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

Introduction

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learning and Acting

Learning is used to gain knowledge. In this course, we look into *machine* learning, so knowledge is a formal data structure. Where possible, we focus on knowledge representations that are *understandable* also to people, such as graph or rule-based representations. These are commonly termed *symbolic*.

Knowledge is used to act well. While human-understandability is only optional, a necessary condition for knowledge to be useful is that it should enable the machine (agent) or the human to perform good *actions* (=make good decisions). We first need to formalize what that means.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Actions

Let us use the following notation for any variable v and a number $k \in \mathbb{N}$:

$$v_{<k} = v_1, v_2, \ldots v_{k-1}$$

$$v_{\leq k} = v_1, v_2, \ldots v_k$$

Agent executes **actions** $y_k \in Y$ in discrete time $k \in \mathbb{N}$.

Y may be infinite but all $y \in Y$ must have a final description (i.e., Y is countable).

(Otherwise the agent could not communicate the chosen action in finite time.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Rewards

From its environment, it receives **rewards** $r_k \in R$, where R is a subset of a *bounded* real interval, i.e. $R = [a, b]$, $a, b \in \mathbb{R}$. Again, all $r \in R$ have to have a finite description (e.g., finite-precision decimal representation).

Reward at time k depends probabilistically on rewards up to time $k-1$, and actions up to time k:

$$r_k|_{y_{\leq k}, r_{<k}} \sim P(r_k \mid y_{\leq k}, r_{<k}) \tag{1}$$

Where the symbol $\sim$ means *'distributed as'*.

(Actions may influence rewards far into the future - consider a chess game with $R = \{0, 1\}$ and 1 indicating a win.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Actions and Rewards

The expression (1) describes the distribution of the random variable r_k assuming the history $y_{\leq k}, r_{<k}$. As it is clumsy, we adopt a simpler notation

$$r_k \overset{c}{\sim} P(r_k \mid y_{\leq k}, r_{<k}) \tag{2}$$

where $\overset{c}{\sim}$ means *'distributed as', assuming the conditions after the bar sign*.
This is a convenience notation only for this class, not used generally in literature!

The marginal probability of r_k is thus

$$r_k \sim P(r_k) = \sum_{y_{\leq k}, r_{<k}} P(r_k \mid y_{\leq k}, r_{<k})$$

(where the sum is over all pairs of sequences $y_{\leq k}, r_{<k}$) and the probability of a particular reward sequence $r_{\leq k}$ given action sequence $y_{\leq k}$ is

$$P(r_{\leq k} \mid y_{\leq k}) = P(r_1 \mid y_1) \cdot P(r_2 \mid r_1, y_{\leq 2}) \ldots P(r_m \mid r_{<m}, y_{\leq m})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Multi-Armed Bandit

Two-armed bandit

- k Game number (we play repeatedly)
- y_k One of two actions (pulling left or right lever)
- r_k Cash received minus cash thrown in at k. Depends on action at k but also on the history of actions of rewards. For example, after too much cash given out for the left-pull, the bandit lowers the mean reward for that action.

wiki

Utility

The optimal action sequence $\overline{y}_1, \overline{y}_2, \ldots$ maximizes the **utility**, which is the expected (discounted) sum of rewards:

- for a *finite* time horizon $m \in \mathbb{N}$:

$$U^{y_{\leq m}} = \mathbb{E}\left(\sum_{k=1}^{m} r_k \,\middle|\, y_{\leq m}\right) = \sum_{r_{\leq m}} \left(P(r_{\leq m} \mid y_{\leq m}) \sum_{k=1}^{m} r_k\right) \tag{3}$$

- for an *infinite* horizon, we need to include a **discount** so that the sum converges. Typically, an exponential discount with base $0 < \gamma < 1$:

$$U^{y_{\leq \infty}} = \lim_{m\to\infty} \mathbb{E}\left(\sum_{k=1}^{m} r_k \gamma^{k-1} \,\middle|\, y_{\leq m}\right) = \lim_{m\to\infty} \sum_{r_{\leq m}} \left(P(r_{\leq m} \mid y_{\leq m}) \sum_{k=1}^{m} r_k \gamma^{k-1}\right) \tag{4}$$

Note: $\sum_{r_{\leq m}}$ *sums over all possible reward sequences* $r_{\leq m}$.

Exercises: finite, infinite

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Instant Rewards

In the general case, there is no obvious way to compute $\overline{y}_1, \overline{y}_2, \ldots$ without knowing the distribution $P(r_k \mid y_{\leq k}, r_{<k})$, e.g., without knowing the bandit's internals.

But consider the special case of **instant rewards**, which do not depend on anything before time k (a "memoryless" bandit). So instead of (2), we have

$$r_k \overset{c}{\sim} P(r_k \mid y_k) \tag{5}$$

Now utility ((3) or (4)) is maximized by constantly repeating the action

$$\overline{y} = \arg\max_y \mathbb{E}\left(r_k \mid y\right) \tag{6}$$

which is independent of k.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Exploration vs. Exploitation

Agent cannot follow (6) without knowing the distribution (5). It can however 'probe' the environment in time $k = 1, 2, \ldots K$ through random actions $y_1, y_2, \ldots, y_K \in Y$, collecting the rewards $r_1, r_2, \ldots, r_K$. K must be large enough so that each $x \in X$ occurs at least once in the action sequence.

For $k \geq K$, the agent switches from random actions to actions

$$y_{k+1} = \arg\max_y \widehat{\mathbb{E}}_k \left(r_{k+1} \mid y \right)$$

where the conditional expectation estimate $\widehat{\mathbb{E}}_k \left(r_{k+1} \mid y \right)$ is the average of all rewards obtained for action y from time 1 to time k.

The larger K, i.e. the longer the *exploration* period,

- the more accurate the estimate is
- the more time is spent behaving randomly, i.e. non-optimally

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Exploration vs. Exploitation (cont'd)

Dilemma: when to switch from exploration to *exploitation*?

Another option is to choose y_1 at random and then for $k \in \mathbb{N}$

- Make a random (exploration) action $y_{k+1} \in Y$ with probability $\epsilon > 0$.
- With prob. $(1 - \epsilon)$ use action that seems optimal:

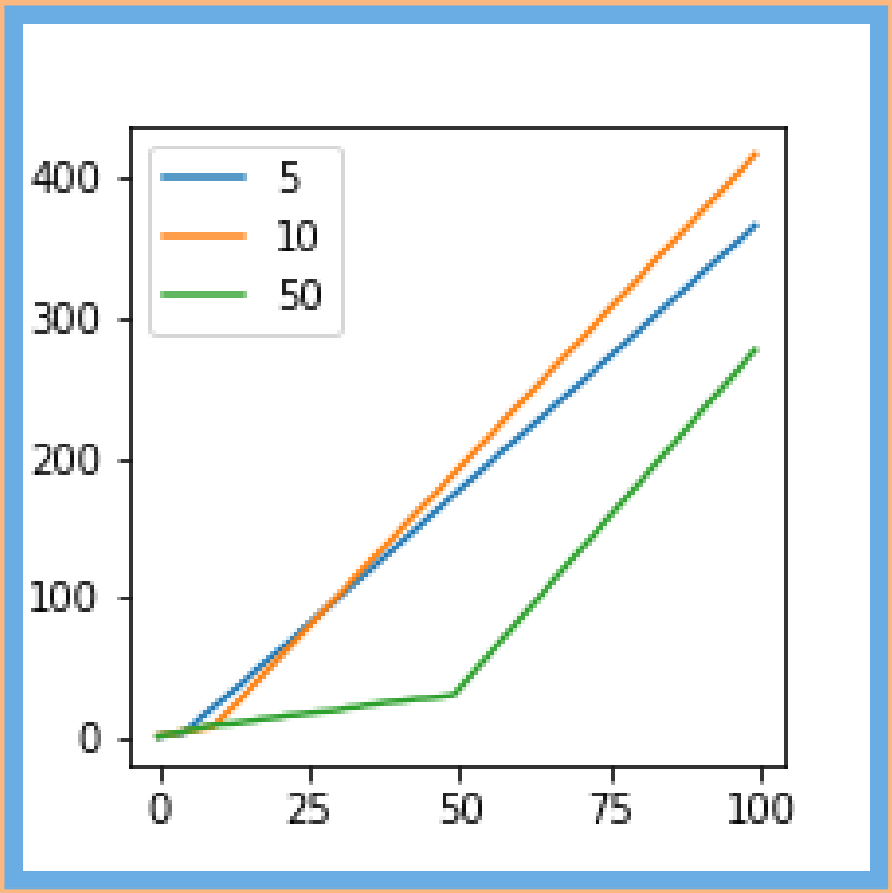
$$y_{k+1} = \arg\max_y \widehat{\mathbb{E}}_k \left(r_{k+1} \mid y \right)$$

Update the estimate $\widehat{\mathbb{E}}_k \left(. \mid . \right)$ at each k to the average of all rewards obtained for action y from time 1 to time k (use e.g. 0 if average undefined. i.e., y has not been used yet).

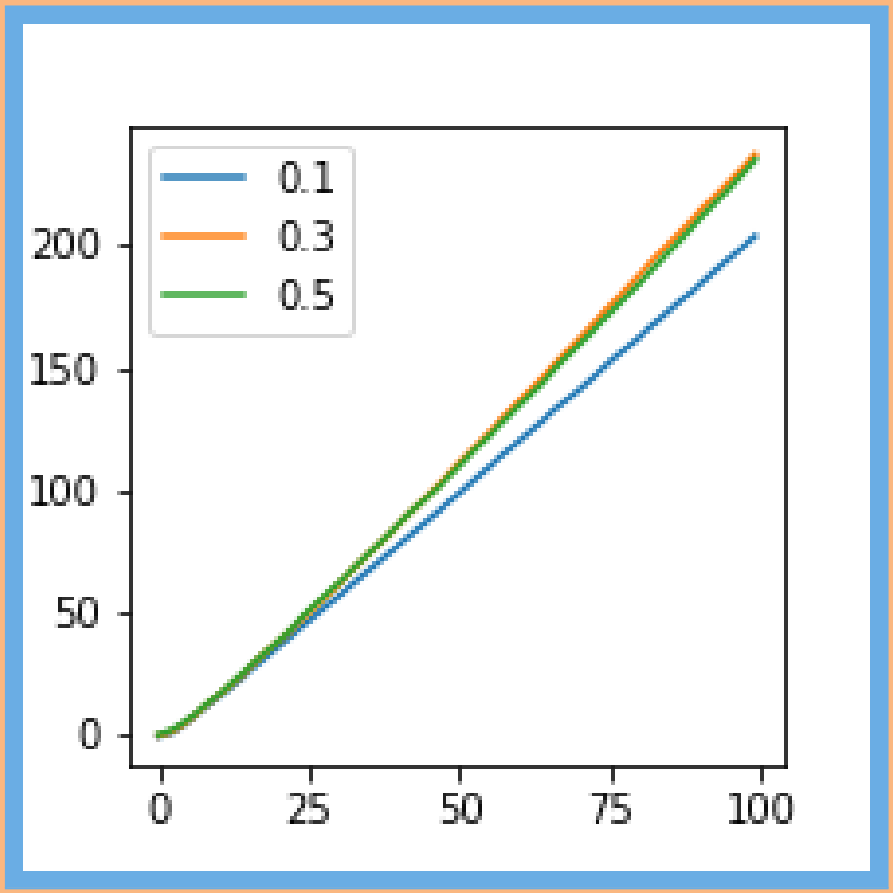
The dilemma is now how large ϵ could be. For an infinite horizon, an option is to let $\epsilon \to 0$ as $k \to \infty$ but keeping $\epsilon > 0$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Exploration vs. Exploitation: Example



Total reward for agent switching from exploration to exploitation after $K = 5, 10, 50$ steps.



Total reward for agent making exploration actions with probability $\epsilon = 0.1, 0.3, 0.5$ steps.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Observations

So far we have considered a 'blind' agent which does not observe anything about its environment, except for the rewards. Now we consider that at each k, it also receives an **observation** x_k from some set X. X may be infinite but its elements must have a finite representation.

The tuple (x_k, r_k) received by the agent is called **percept**, written xr_k for short. Analogically to (2), the percept generally depends on the entire history:

$$xr_k \overset{c}{\sim} P(xr_k \mid y_{\leq k}, xr_{<k}) \tag{7}$$

Despite the short notation xr_k, the above is a *joint* probability of x_k and r_k, which can be written using the probability chain rule as the product

$$P(x_k \mid r_k, xr_{<k}, y_{\leq k}) \tag{8}$$
$$\cdot P(r_k \mid xr_{<k}, y_{\leq k}) \tag{9}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Stock Market

k Day number

x_k Available market data on day k at closing time

y_k Investor's action (buy, sell, ..) on day k before closing time

r_k Cash earned or lost by investor on day k before closing time

$P(x_k \mid r_k, xr_{<k}, y_{\leq k})$ Current market data depend on the entire history of market data and investor's actions and rewards *(as well as other, unobserved factors - that is why it is a probability, not a function.).*

$P(r_k \mid xr_{<k}, y_{\leq k})$ Current reward depends on entire history of actions *(think a Bitcoin bought 5 years ago)*, market data except current x_k, and rewards.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Utility with Observations

Involving observations, the utility definitions (3) and (4) change.

- for a *finite* time horizon $m \in \mathbb{N}$, (3) turns into:

$$U^{y_{\leq m}} = \mathbb{E}\left(\sum_{k=1}^{m} r_k \;\middle|\; y_{\leq m}\right) = \sum_{xr_{\leq m}} \left(P(xr_{\leq m} \mid y_{\leq m}) \sum_{k=1}^{m} r_k\right) \tag{10}$$

- for an *infinite* time horizon, (4) turns into:

$$U^{y_1,y_2,\cdots} = \lim_{m\to\infty} \mathbb{E}\left(\sum_{k=1}^{m} r_k \gamma^{k-1} \;\middle|\; y_{\leq m}\right) = \lim_{m\to\infty} \sum_{xr_{\leq m}} \left(P(xr_{\leq m} \mid y_{\leq m}) \sum_{k=1}^{m} r_k \gamma^{k-1}\right) \tag{11}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Instant Rewards with Observations

We now revisit the instant rewards assumption, this time with observations x_k.

With this assumption, the agent is rewarded only for how well it reacted to the *last seen observation* so for $k > 1$, (9) is replaced by

$$r_k \overset{c}{\sim} P(r_k \mid x_{k-1}, y_k)$$

i.e., we have for $k \in \mathbb{N}$:

$$\begin{aligned} r_1 &\overset{c}{\sim} P(r_1 \mid y_1) \\ r_{k+1} &\overset{c}{\sim} P(r_{k+1} \mid x_k, y_{k+1}) \end{aligned} \tag{12}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Instant Rewards with Observations (cont'd)

The instant rewards assumption enabled to identify the optimal action (6) when observations were not part of the framework.

With observations, there is no longer an obvious way to compute optimal actions even if we know or can estimate the distribution (12). In particular,

$$y_{k+1} = \arg\max_{y \in Y} \mathbb{E}\left(r_{k+1} \mid x_k, y\right) \tag{13}$$

does *not* necessarily yield the optimal action y_{k+1}.

This is because we still assume in (8) that actions influence future observations. An effect of choosing y_{k+1} by (13) may be that the agent will receive 'worse' observations (allowing smaller rewards) in the future.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Who Wants to Be a Millionaire?

k Question number

x_k Question

y_k Answer (one of a list of options) to question x_{k-1} *Formally, the k 'clock' ticks between a question and the subsequent answer.*

r_k Cash earned or lost

$P(x_k \mid r_k, xr_{<k}, y_{\leq k})$ Current question depends on history: correct answers cause more difficult questions to come. (Also: high rewards entail such questions to come.)

$P(r_k \mid x_{k-1}, y_k)$ Current reward depends only on the last question (in particular, the difficulty of it) and the answer to it (in particular, on its correctness w.r.t. the question).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

i.i.d. Observations

We will now consider one more assumption: observations are **i**ndependent of history and at each k they are sampled from the **i**dentical **d**istribution $P(x_k)$.

$$x_k \sim P(x_k) \tag{14}$$

This prevents the environment from 'playing tricks' on the agent. In the millionaire example, this would mean that questions are drawn randomly from a bucket and do not get progressively harder.

With the assumptions of instant rewards and i.i.d. observations, the optimal action is

$$\overline{y}_{k+1} = \arg\max_{y \in Y} \mathbb{E}\left(r_{k+1} \mid x_k, y\right) \tag{15}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Policy

Without observations, the optimal action (6) was a constant. Here, the optimal action in (15) is a *function* of x_k. In general, a function

$$\pi : X \to Y \tag{16}$$

mapping an observation to an action is called a **policy**. The optimal policy will be denoted $\overline{\pi}(x)$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Sequential vs. Non-Sequential: Summary

Decision processes where rewards depend on history are called **sequential**.

No observations	
sequential	**non-sequential**
$r_k \overset{c}{\sim} P(r_k \mid y_{\leq k}, r_{<k})$ (2)	$r_k \overset{c}{\sim} P(r_k \mid y_k)$ (5)
	$\overline{y} = \arg\max_y \mathbb{E}(r_k \mid y)$ (6)

With observations	
sequential	**non-sequential**
$r_k \overset{c}{\sim} P(r_k \mid xr_{<k}, y_{\leq k})$ (9)	$r_1 \overset{c}{\sim} P(r_1 \mid y_1)$
	$r_{k+1} \overset{c}{\sim} P(r_{k+1} \mid x_k, y_{k+1})$ (12)
$x_k \overset{c}{\sim} P(x_k \mid r_k, xr_{<k}, y_{\leq k})$ (8)	$x_k \sim P(x_k)$ (14)
	$\overline{y}_{k+1} = \arg\max_y \mathbb{E}(r_{k+1} \mid x_k, y)$ (15)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Scenarios in this course

In increasing generality:

1. *Concept and distribution learning from i.i.d. data*: non-sequential
2. *Online concept learning*: sequential (but additional assumptions)
3. *Reinforcement learning*: sequential (but additional assumptions)
4. *Universal AI*: sequential ("fully")

For didactic reasons, we will proceed in the 2, 1, 3, 4 order.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Classification

Classification is a special case of the agent-environment interaction defined by two assumptions

1. Y is finite
2. rewards are instant (12)

Elements of Y are called *classes*.

Similarly, for 'regression' we would replace the first condition with $Y = \mathbb{R}$. We do not elaborate regression in this course.

(exercise problem)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Classification of Handwritten Numbers

$X = \mathbb{R}^{16\times 16}$ (x are pixel vectors)
$Y = \{0, \ldots, 9\}$

When training a classification agent, if we know the 'true' classes $\overline{y}(x)$ of observations, we may use them to prescribe rewards by function $r : Y \times Y \to R$, such as

$$r_{k+1} = r(y_{k+1}, \overline{y}(x_k)) = -|y_{k+1} - \overline{y}(x_k)|$$

$$r_{k+1} = r(y_{k+1}, y^*(x_k)) = \begin{cases} 0 \text{ if } y_{k+1} = \overline{y}(x_k) \\ -1 \text{ otherwise} \end{cases} \quad \text{(Unit reward)} \qquad (17)$$

The negative reward function $-r(.,.)$ is called a **loss function**.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Concept Learning

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Concept Classification

We will now focus on the simplest interesting form of classification: only two classes and no "noise".

Formally, any subset $C \subseteq X$ is called a **concept on X** and we define **concept classification** as a special case of classification where $Y = \{0, 1\}$ there is a **target concept** C on X instantiating the rewards (12) as $(k \in \mathbb{N})$

$$r_1 = 0$$
$$r_{k+1} = \begin{cases} y_{k+1} - 1 \text{ if } x_k \in C \\ -y_{k+1} \text{ otherwise} \end{cases} \tag{18}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Concept Classification (cont'd)

In other words, the agent decides by $y_{k+1} = 1$ ($y_{k+1} = 0$) that $x_k \in C$ ($x_k \notin C$, respectively) and gets reward $r_{k+1} = 0$ ($r_{k+1} = -1$) if the decision was right (wrong, respectively).

(Exercise problem)

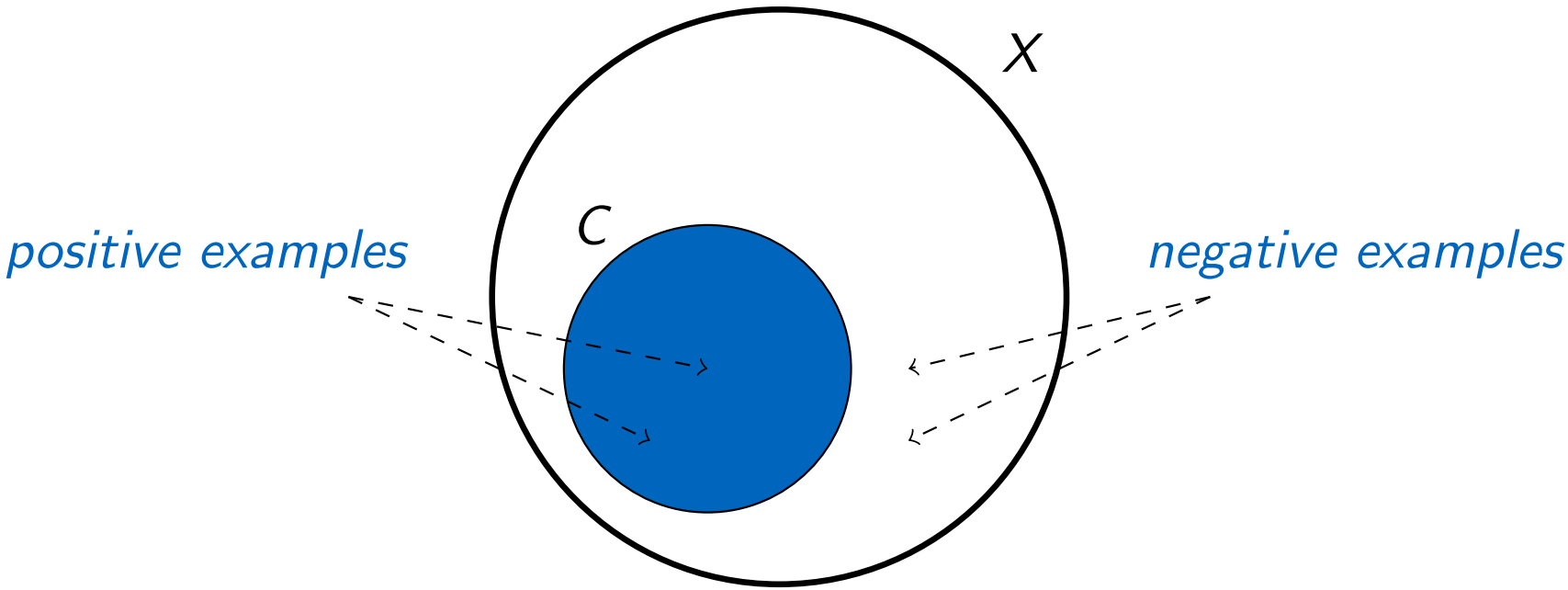
Rewards r_{k+1} here depend deterministically on x_k and y_{k+1}, hence no "noise".

Note that arbitrary classification can be done by a finite number of concept classification agents. Indeed, since Y is finite, each $y \in Y$ can be represented by a binary number with $n \approx \lg |Y|$ digits. So we just let the target concept for the i's agent contain all $x \in X$ for which the i's digit of optimal action $\overline{y}$ is 1. This agent will learn to predict the i's digit of the optimal action for x.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Positive and Negative Examples

Observations $x \in C$ ($x \notin C$, respectively) received by the agent are called **positive (negative) examples** of C.



Example: $X \sim$ descriptons of animals. $C \sim$ same for mammals. Positive example: description of a cat, negative example: same for a chicken.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Hypothesis

From the examples, the agent learns a **hypothesis** h, which is a *finite-size* description of a binary policy. The hypothesis also induces a concept

$$C(h) = \{ x \in X \mid h(x) = 1 \} \tag{19}$$

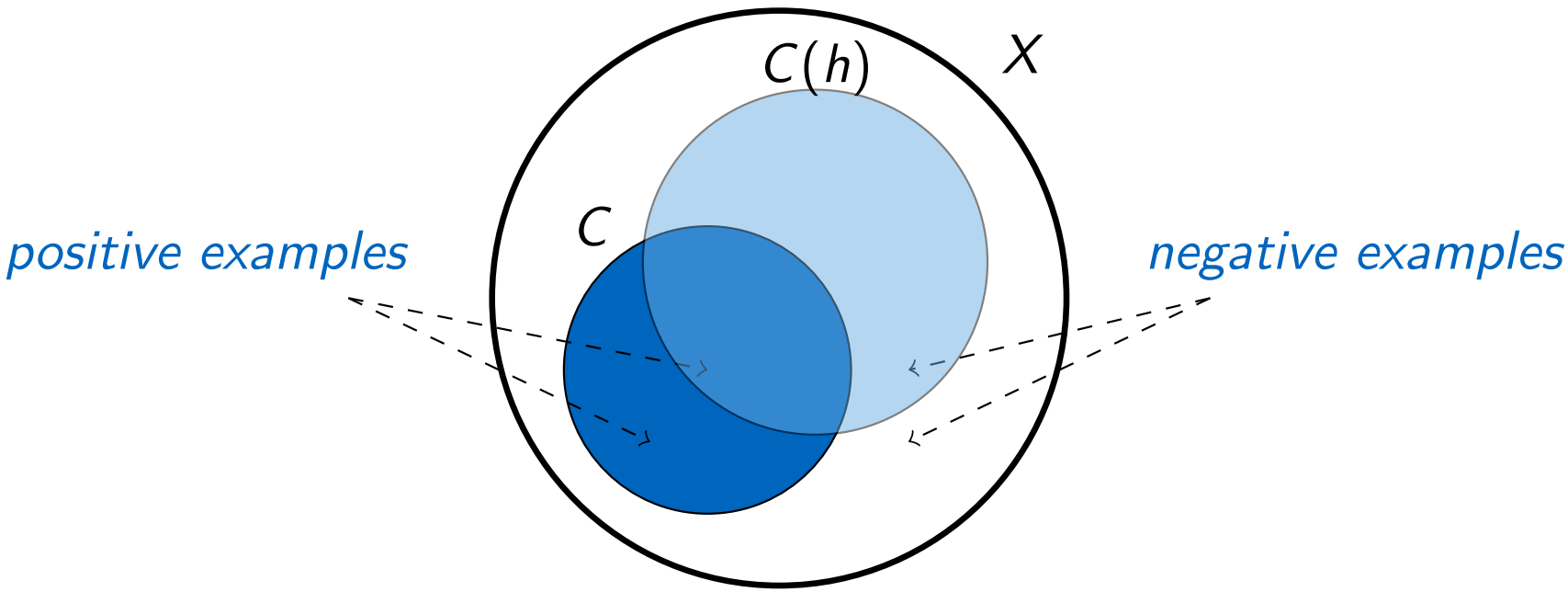
Note that we overload the h symbol to denote both the description of the hypothesis and the policy function it defines.

(19) depends on the way the function $h(x)$ is determined from the description h. We do not make this dependence explicit as we will only be interested in hypotheses with an obvious functional interpretation.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Hypothesis vs. Concept

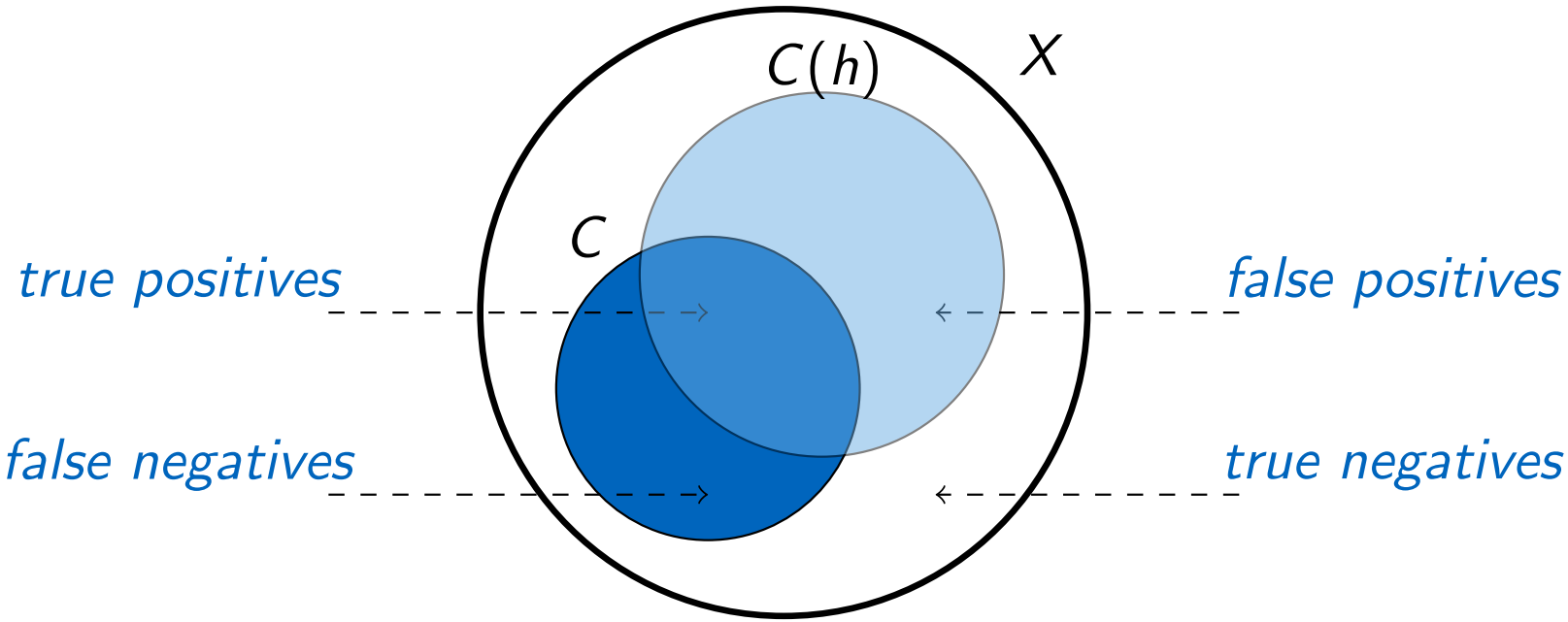
The goal of learning is to find h such that $C(h) = C$.



Example (logical): $h = \text{milk} \wedge \neg\text{feathers}$, $h(x) = 1$ iff $x \models h$.

Error Types

Until $C(h) = C$, there are four kinds of observations



False positives and false negatives form the *error region*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Concept Class

With an unlimited supply of non-repeated examples, can we always learn the target concept, i.e. find a h such that

$$C(h) = C \tag{20}$$

In general, *no*. There are $2^{|X|}$ possible concepts on X. If X is infinite (e.g. $X = \mathbb{N}$), there is an uncountable ($|\mathbb{R}|$) number of such concepts. A hypothesis is a finite description so there is a countable number ($|\mathbb{N}|$) of hypotheses. Thus there are more concepts than hypotheses.

To allow any learnability results, we will always have to assume that C is not arbitrary ($C \in 2^X$) but belongs to a smaller **concept class on** X

$$\mathcal{C} \subset 2^X \tag{21}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Hypothesis Class

A **hypothesis class** $\mathcal{H}$ is a set of hypotheses. For example, a set of *propositional-logic conjunctions*.

$\mathcal{C}(\mathcal{H})$ denotes the concept class on X induced by hypotheses in $\mathcal{H}$, i.e.

$$\mathcal{C}(\mathcal{H}) = \{\ C(h) \mid h \in \mathcal{H}\ \} \tag{22}$$

So if $\mathcal{H} =$ propositional-logic conjunctions then $\mathcal{C}(\mathcal{H})$ means the set of all concepts on X that can be described by such conjunctions.

Terminology: when there is no risk of confusion, we will call $\mathcal{C}(\mathcal{H})$ the same as $\mathcal{H}$, e.g. "conjunctions" rather than "concept class on X induced by conjuctions". If $C(\overline{h})$ is the target concept, we will call $\overline{h}$ the **target hypothesis**.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Mistake-Bound Learning Model

Given a concept class $\mathcal{C}$ we want to study whether a learning agent can learn concepts from $\mathcal{C}$. What does "can learn concepts from $\mathcal{C}$" exactly mean? One definition is provided by the **mistake-bound** learning model also known as the **online learning** model.

Due to (18), maximizing the utilities (10) or (11) means minimizing the number of mistakes, i.e., actions followed immediately by reward $r = -1$. But what utility value is considered a success?

In the mistake-bound model, we request that if the target concept $C \in \mathcal{C}$, the number of mistakes is *finite* even for an infinite time horizon m, and this is true for *any distribution of observations* (8).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Mistake-Bound Learning Model (cont'd)

Given (18), the total number of mistakes in one possible history $xr_{\leq k}$ is $\sum_{k=1}^{\infty} |r_k|$. Since the latter must be finite, $\sum_{k=1}^{\infty} r\gamma^k$ converges even with $\gamma = 1$ (we will keep $\gamma = 1$ unless stated otherwise).

As this must be true for any distribution of observations (8), the expectation in the infinite utility (11) also converges and $|U^{y_1,y_2,\cdots}|$ is the expected total number of mistakes.

Recall that the sequence of observations determined by distribution (8) is the only source of randomness in the agent-environment interaction in concept classification as (9) is set deterministically by (18).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Mistake-Bound Learning Model (cont'd)

Moreover, the model requests that the number of mistakes is not just finite, but reasonably small. In particular, it should grow at most *polynomially* with the *size (descriptive complexity)* of observations $x \in X$, denoted n_X. When observations are feature tuples of dimension n, we will always set $n_X = n$.

The model also defines when concept learning is time-efficient.

Mistake-bound model (or Online learning model)

In the concept classification protocol, an agent **learns $\mathcal{C}$ from X online** if for any target concept from $\mathcal{C}$ on X and an arbitrary distribution (8), it makes a sequence of decisions $y_1, y_2, \ldots$ such that $\sum_{k=1}^{\infty} |r_k| \leq poly(n_X)$. It learns $\mathcal{C}$ online from X **efficiently**, if in addition, the time taken to compute an action from an observation is also at most polynomial in n_X.

(*Exercise problem*)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

The Winnow Algorithm

Winnow assumes Boolean-tuple observations, i.e. $X = \{0,1\}^n, n \in \mathbb{N}$ and tries to identify the target concept $C \subseteq X$ by a hyperplane in X.

The agent's hypothesis at time k is an n-tuple of integers $h_k = (h_k^1, h_k^2, \ldots, h_k^n)$ specifying the hyperplane, initially set to

$$h_1 = (1, 1, \ldots, 1) \tag{23}$$

Its policy $y_{k+1} = h_k(x_k)$ $(k \in \mathbb{N})$ is given by

$$h_k(x_k) = 1 \text{ iff } \sum_{i=1}^{n} h_k^i x_k^i > \frac{n}{2} \tag{24}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

The Winnow Algorithm (cont'd)

On each mistake, h_k is updated to h_{k+1} with a simple learning rule:

- On a false negative x_k ($y_{k+1} = 0$, $r_{k+1} = -1$), *promote* each component h_k^i where $x_k^i = 1$ by doubling its value:

$$h_{k+1}^i = 2h_k^i \tag{25}$$

- On a false positive ($y_{k+1} = 1$, $r_{k+1} = -1$), *eliminate* each component h_k^i where $x_k^i = 1$ by zeroing it:

$$h_{k+1}^i = 0 \tag{26}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow Learning Monotone Disjunctions

Using hyperplane separation, Winnow can learn only classes of linearly separable concepts. One example is the class $\mathcal{C}(\mathcal{H})$ of *monotone disjunctions* made out of up to n propositional variables $\mathrm{p}_1, \mathrm{p}_2, \ldots \mathrm{p}_n$, i.e.,

$$\mathcal{H} = \{ \mathrm{p}_{i_1} \vee \mathrm{p}_{i_2} \vee \ldots \mathrm{p}_{i_s} \mid 1 \leq i_j \leq n \}$$

So, for example, when $n = 4$ and the target hypothesis is $\mathrm{p}_1 \vee \mathrm{p}_3$, the agent's hypothesis $h = [2, 0, 1, 0]$ will not make any mistakes because the target disjunction is true iff

$$2x^1 + x^3 > 2$$

We are putting component indexes to the superscript of x and h_k, reserving the subscript for a time index.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Conjunctive Observations

For Winnow, we assumed $X = \{0,1\}^n, n \in \mathbb{N}$, so an example $x \in X$ specifies *each* of the n Boolean values. We want to allow the case that x does not specify some of them. This could be done by leting $X = \{0,1,?\}^n$ where ? stands for *value uknown*.

Another way is to let X be the set of **contingent** (not tautologically true or false) *conjunctions* of propositional literals made of atoms selected from $\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_n$. For example, with $n = 3$ the observation

$$\mathrm{p}_1 \wedge \neg \mathrm{p}_3$$

represents the same information as

$$(1, ?, 0)$$

We define the complexity n_X of such observations to be n.

(*Exercise problem*)

Conjunctive Hypotheses

Unless stated otherwise, the term conjunction (disjunction) will mean a conjunction (disjunction) of *propositional literals*, excluding e.g. a conjunction of disjunctions (or the reverse). Non-propositional cases will be marked explicitly.

We will be interested in hypotheses which are conjunctions, providing the following decision policy $y = h(x)$ for conjunctive examples

$$h(x) = 1 \text{ iff } x \models h \tag{27}$$

where $\models$ is *tautological consequence*. So e.g. the observation

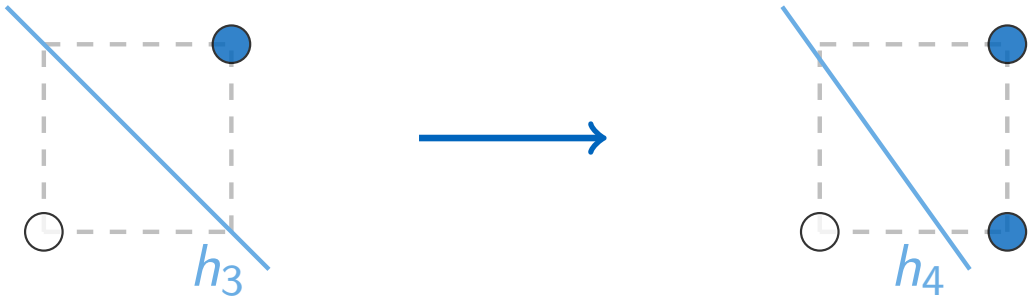
$$x = \text{milk} \wedge \neg\text{feathers} \wedge \neg\text{flies}$$

is decided positively ($h(x) = 1$) by $h = \text{milk} \wedge \neg\text{feathers}$.

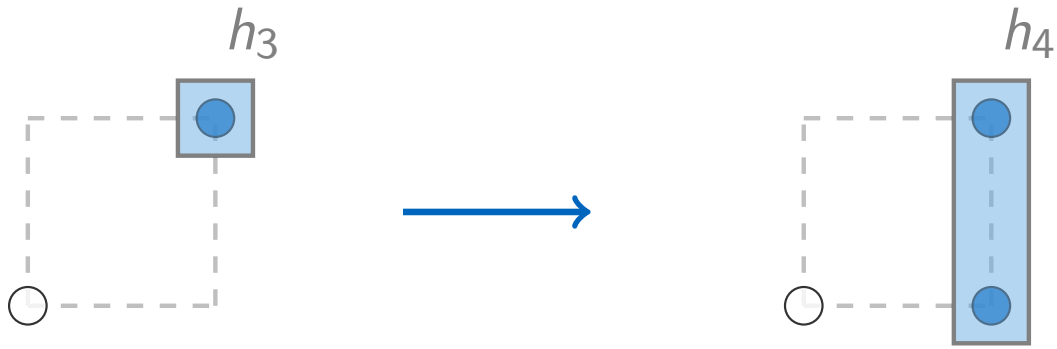
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Separation vs. Generalization

Winnow uses the popular learning technique of *separation*



An alternative is the "covering" approach seeking the smallest joint *generalization* of positive examples



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generality and Subsumption Order

Let π, π' be two policies $X \to \{0,1\}^n$. We say that y is **at least as general as** y' if $\pi(x) = 1$ for any $x \in X$ such that $\pi'(x) = 1$.

Let h, h' be conjunctions that prescribe policies by (27). If $h' \models h$ then h is at least as general as h'. *(exercise problem)*

Let h, h' be two conjunctions or two disjunctions. We say that h **subsumes** h' (written $h \subseteq h'$) if $\text{Lits}(h) \subseteq \text{Lits}(h')$ where $\text{Lits}(c)$ denotes the set of literals in c. We say that h **strictly subsumes** (written $h \subset h'$) h' if $\text{Lits}(h) \subset \text{Lits}(h')$.

Theorem 1

Let h, h' be conjunctions. If $h \subseteq h'$ then $h' \models h$. Let furthermore h' not be tautologically false. Then $h \subseteq h'$ if and only if $h' \models h$.

(exercise problem)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Least General Generalization

Let h, h' be two conjunctions (two disjunctions, respectively). We say that g is a **least general generalization** of h and h' if $g \subseteq h$, $g \subseteq h'$, and there is no conjunction (disjunction) g' such that $g \subset g'$, $g' \subseteq h$, $g' \subseteq h'$.

Let h, h' be two conjunctions (two disjunctions, respectively) and let us define:

$$\mathrm{lgg}(\mathbf{h}, \mathbf{h}') = \mathrm{Lits}(h) \cap \mathrm{Lits}(h') \tag{28}$$

Easy to verify: $\mathrm{lgg}(h, h')$ is a least general generalization of h and h'.

The proof is an exercise problem.

(a further exercise problem)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

LGG Properties and Set-LGG

The subsumption order $\subseteq$ induces a *lattice* where lgg is the *least upper bound* (lup). As any lup, lgg has these properties:

$$\text{lgg}(a, b) = a \text{ if } a \subseteq b \tag{29}$$

$$\text{lgg}(a, b) = \text{lgg}(b, a) \text{ (commutativity)} \tag{30}$$

$$\text{lgg}(a, \text{lgg}(b, c)) = \text{lgg}(\text{lgg}(a, b), c) \text{ (associativity)} \tag{31}$$

Properties 30 and 31 let us extend lgg naturally to *sets* of conjunctions or disjunctions:

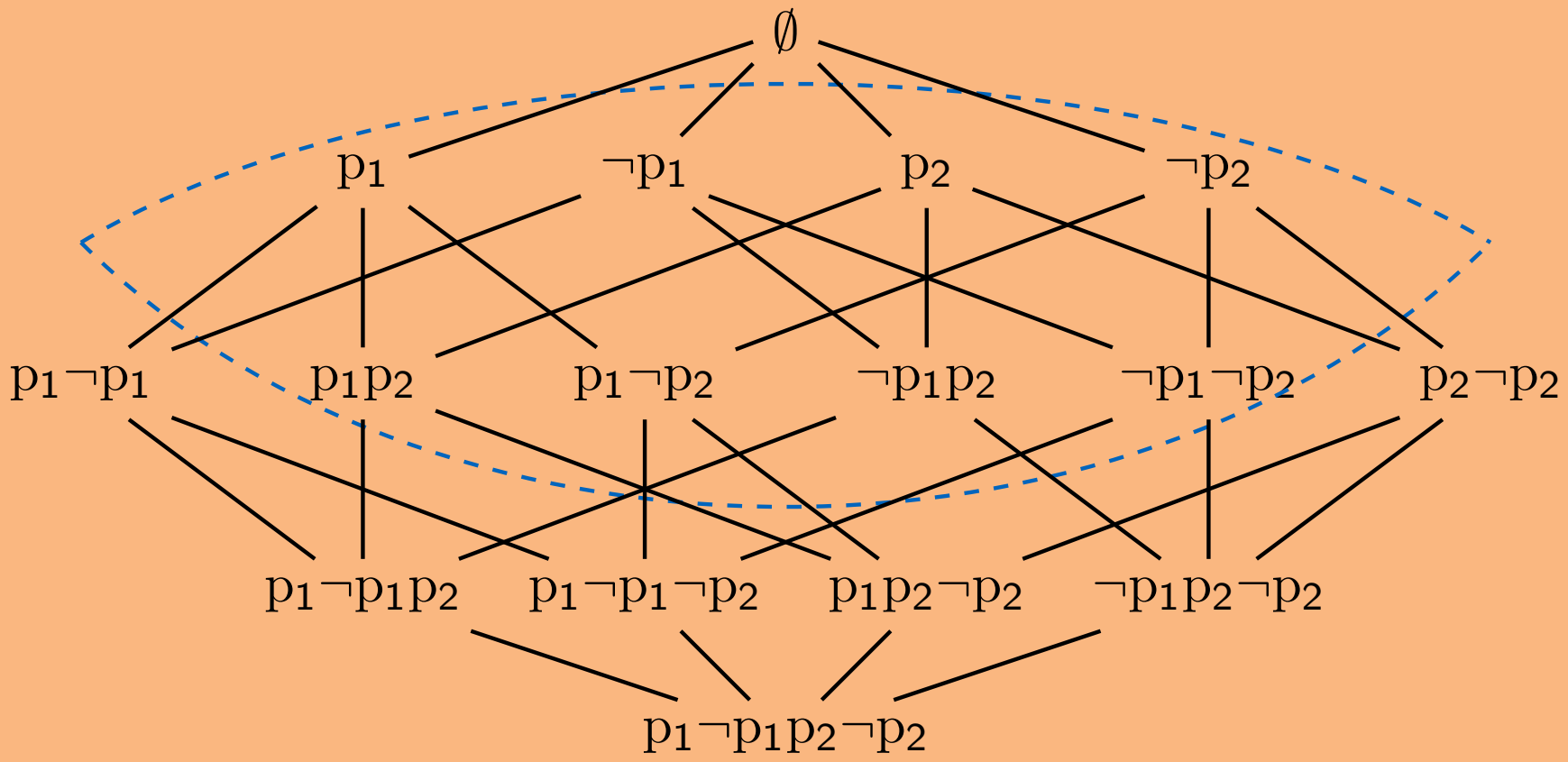
$$\text{lgg}(\{ x_1, x_2, \ldots, x_m \}) = \text{lgg}(\ldots \text{lgg}(\text{lgg}(x_1, x_2), \ldots), x_m) \tag{32}$$

where the order of the x_k on the RHS is irrelevant. For conjunctions, obviously $\text{lgg}(\{ x_1, x_2, \ldots, x_m \}) = \bigcap_{k=1}^{m} x_k$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Subsumption Lattice on Conjunctions

Contingent conjunctions are enclosed by the dashed curve.



The Generalization Algorithm

The generalization algorithm assumes X to consist of contingent conjunctions on variables $\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_n$. It uses the policy (27), which can be written as

$$h(x) = 1 \text{ if } h \subseteq x \tag{33}$$

because $x \in X$ are contingent. It has a simple learning rule:

- Wait for the first positive example. That is, emit actions $y_k = 0$ until $r_k = -1$, then x_{k-1} is a positive example. Set $h_k = x_{k-1}$.
- Continue receiving percepts and on each mistake ($r_{k+1} = -1$), set

$$h_{k+1} = \mathrm{lgg}(h_k, x_k) \tag{34}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow: Mistake Bound

Theorem 2

Let the target hypothesis be a monotone disjunction on $X = \{0,1\}^n$ *and* s *be the number of atoms in it. Winnow makes at most* $2 + 2s \lg n$ *mistakes, i.e.,*

$$\sum_{k=1}^{\infty} |r_k| \leq 2 + 2s \lg n \tag{35}$$

As $s \leq n$, we have $\sum_{k=1}^{\infty} |r_k| \leq poly(n)$, and thus Winnow learns monotone disjunctions online. It also learns them efficiently (easy to check). The $\lg n$ term makes Winnow a fast learner (compared e.g. to the perceptron) when the number of atoms s in the target disjuction is small ($s \ll n$).

To prove the theorem, we first show two lemmas.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow: Mistake Bound (cont'd)

Lemma 1

If x_k $(k \in \mathbb{N})$ *is a* false negative *then* $\sum_{i=1}^{n} h_{k+1}^i - \sum_{i=1}^{n} h_k^i \leq \frac{n}{2}$

Proof of Lemma 1: $h_k(x_k) = 0$ since x_k is a false negative, and by (24),

$$\sum_{i=1}^{n} h_k^i x_k^i \leq \frac{n}{2} \tag{36}$$

Furthermore,

$$\sum_{i=1}^{n} h_{k+1}^i - \sum_{i=1}^{n} h_k^i = \sum_{i=1}^{n} (h_{k+1}^i - h_k^i) = \sum_{i=1}^{n} (h_{k+1}^i - h_k^i) x_k^i \tag{37}$$

where the last equality is because for any i such that $x_k^i = 0$, $h_{k+1}^i = h_k^i$ due to the Winnow learning rule.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow: Mistake Bound (cont'd)

Due to (25),

$$\sum_{i=1}^{n}(h_{k+1}^i - h_k^i)x_k^i = \sum_{i=1}^{n}(2h_k^i - h_k^i)x_k^i = \sum_{i=1}^{n} h_k^i x_k^i$$

Therefore

$$\sum_{i=1}^{n} h_{k+1}^i - \sum_{i=1}^{n} h_k^i = \sum_{i=1}^{n} h_k^i x_k^i \leq \frac{n}{2}$$

where the inequality is given by (36). This proves the lemma.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow: Mistake Bound (cont'd)

Lemma 2

If x_k $(k \in \mathbb{N})$ *is a* false positive *then* $\sum_{i=1}^{n} h_k^i - \sum_{i=1}^{n} h_{k+1}^i > \frac{n}{2}$

Proof of Lemma 2: $h_k(x_k) = 1$ since x_k is a false positive, and by (24),

$$\sum_{i=1}^{n} h_k^i x_k^i > \frac{n}{2} \tag{38}$$

The lemma is proven by the equation below, where the first equality is due to the Winnow learning rule as in (37), the second equality due to the same rule prescribing $h_{k+1}^i = 0$ for each i such that $x_k^i = 1$, and the last inequality is from (38):

$$\sum_{i=1}^{n} h_k^i - \sum_{i=1}^{n} h_{k+1}^i = \sum_{i=1}^{n} (h_k^i - h_{k+1}^i) x_k^i = \sum_{i=1}^{n} h_k^i x_k^i > \frac{n}{2}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow: Mistake Bound (cont'd)

Lemma 3

For $\forall k \in \mathbb{N}, i \in [1; n] : h_k^i \leq n$

Proof of Lemma 3: From (23) $h_1^i = 1$. For contradiction, assume the lemma is not true and $k+1$ ($k \in \mathbb{N}$) is the smallest index for which $h_{k+1}^i > n$. Since $h_k^i \leq n$, h_k was promoted to h_{k+1} by (25), implying $x_k^i = 1$ (otherwise h_k^i would not have been promoted) and $h_k^i > n/2$ (promotion doubles the value). But then $\sum_{i=1}^{n} x^i h_k^i > n/2$ so by (24), $y_{k+1} = h_k(x_k) = 1$ so h_k^i was not promoted. This contradiction proves the lemma.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow: Mistake Bound (cont'd)

Proof of Theorem 2: Let $\mathrm{FN_k}$ ($\mathrm{FP_k}$, respectively) be the number of false negatives (false positives) up to time k so $\mathrm{FN_k} + \mathrm{FP_k}$ is the total number of mistakes up to k. From (23), we have

$$\sum_{i=1}^{n} h_1^i = n$$

From this and Lemmas (1) and (2) we have

$$\sum_{i=1}^{n} h_k^i \leq n + \frac{n}{2}\mathrm{FN_k} - \frac{n}{2}\mathrm{FP_k} \tag{39}$$

Furthermore, since $h_1^i = 1$ and any decrease is only through elimination, which zeros the component, we have for $\forall k \in \mathbb{N}, i \in [1; n]$

$$h_k^i \geq 0 \tag{40}$$

Winnow: Mistake Bound (cont'd)

From (39) and (40), it holds for $\forall k \in \mathbb{N}$:

$$0 \leq \sum_{i=1}^{n} h_k^i \leq n + \frac{n}{2}\mathrm{FN_k} - \frac{n}{2}\mathrm{FP_k}$$

implying

$$\frac{n}{2}\mathrm{FP_k} \leq n + \frac{n}{2}\mathrm{FN_k}$$

and since $n > 0$, we can multiply this by $\frac{2}{n}$, obtaining

$$\mathrm{FP_k} \leq 2 + \mathrm{FN_k} \tag{41}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow: Mistake Bound (cont'd)

Each promotion doubles h^i where i is one of the s indexes corresponding to the s atoms in the target disjunction. First assume $s > 0$. So after $\mathrm{FN_k}$ promotions, at least one of them was doubled $\frac{\mathrm{FN_k}}{s}$ times or more, thus for $\forall k \in \mathbb{N}, \exists i \in [1; n]$: $h_k^i \geq 2^{\mathrm{FN_k}/s}$, i.e.,

$$\lg h_k^i \geq \frac{\mathrm{FN_k}}{s}$$

From Lemma (3) we further have for $\forall k \in \mathbb{N}, i \in [1; n]$: $\lg h_k^i \leq \lg n$. Thus for $\forall k \in \mathbb{N}, \exists i \in [1; n]$

$$\frac{\mathrm{FN_k}}{s} \leq \lg h_k^i \leq \lg n \tag{42}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Winnow: Mistake Bound (cont'd)

Since we assumed $s > 0$, (42) can be written as

$$\mathrm{FN}_k \leq s \lg n$$

If on the other hand $s = 0$ then the target disjunction is tautologically false, so there are no false negatives, therefore $\mathrm{FN}_k = 0$ and the inequality is satisfied trivially. Combining this with (41) we get for the total number of mistakes

$$\mathrm{FP}_k + \mathrm{FN}_k \leq 2 + 2s \lg n$$

and since this value holds for any $k \in \mathbb{N}$, we can write

$$\sum_{k=1}^{\infty} |r_k| \leq 2 + 2s \lg n$$

which completes the proof. *(exercise problem)*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalization: Mistake Bound

Theorem 3

Let X be contingent conjunctions made of up to n variables and let the target hypothesis be a conjunction. The generalization algorithm makes at most n mistakes, i.e.,

$$\sum_{k=1}^{\infty} |r_k| \leq n \tag{43}$$

Thus the generalization algorithm learns conjunctions from contingent conjunctions online. It also learns them efficiently (easy to check).

As any Boolean tuple can be represented through a contingent conjunction, the theorem implies that the generalization algorithm learn conjunctions online from $X = \{ 0, 1 \}^n$ as well.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalization: Mistake Bound (cont'd)

Proof of Theorem 3. By (33), $x \in X$ is classified positive iff $h \subseteq x$. Let $\overline{h}$ be the target conjunction. So any $x \in X$ is a positive example iff $\overline{h} \subseteq x$.

1. $\overline{h} \subseteq h_1$ because h_1 is set to be the first *positive* example. (Without loss of generality, we start indexing from the instant immediately after receiving the first positive example, thus skipping the waiting stage.)
2. $\forall k : \overline{h} \subseteq h_k$ *implies* $\overline{h} \subseteq h_{k+1}$ If x_k is classified correctly, then $h_{k+1} = h_k$ and the implication holds trivially. If x_k is a false positive, i.e., $\overline{h} \not\subseteq x_k$, and $h_k \subseteq x_k$ then from (34) and (29), $h_{k+1} = h_k$, and again the implication holds trivially. Lastly, if x_k is a false negative, i.e., $\overline{h} \subseteq x_k$, and $h_k \not\subseteq x_k$ then from (34) $h_{k+1} = \mathrm{lgg}(h_k, x_k)$. If $\overline{h} \subseteq h_k$ then both arguments of the lgg are subsumed by $\overline{h}$ and since lgg is a least general generalization, $\overline{h} \subseteq \mathrm{lgg}(h_k, x_k)$. Thus the implication again holds.
3. $\forall k \in \mathbb{N} : \overline{h} \subseteq h_k$ – by induction using (1) and (2).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalization: Mistake Bound (cont'd)

4. On each mistake at k, $h_k \not\subseteq x_k$, i.e x_k is a false negative. This is because if x_k was a false positive, then $\overline{h} \not\subseteq x_k$ and due to (3) also $h_k \not\subseteq x$, but then x is not classified as positive. So *mistakes are made only on positive examples*.
5. On each mistake at k, h_{k+1} has strictly fewer literals than h_k. This is because due to (34), $h_{k+1} = \text{lgg}(h_k, x_k)$, and since lgg is a least general generalization, it must be that $h_{k+1} \subseteq x_k$. From (4), we have $h_k \not\subseteq x_k$. Thus some literals of h_k are not in h_{k+1}.
6. Since examples are contingent, they have at most n literals (each of the n atoms is included either as a positive or negative literal but not both) and since h_1 is the first positive example, it also has at most n literals. Due to (5), at least one literal is removed on each mistake, so the *maximum number of mistakes is n*, which completes the proof.

(*exercise problem*)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability

A concept class $\mathcal{C}$ on X is (efficiently) **learnable from X online** if *there is an algorithm* that learns $\mathcal{C}$ from X (efficiently) online.

If for some hypothesis class $\mathcal{H}$, $\mathcal{C}(\mathcal{H})$ is (efficiently) learnable from X online, we say that $\mathcal{H}$ is (efficiently) learnable from X online. We have seen that

- monotone disjunctions are efficiently learnable online from truth-value assignments by the Winnow algorithm.
- conjunctions are efficiently learnable online from contingent conjunctions (incomplete observations) by the generalization algorithm.

From each of these two results, learnability of general *disjunctions*, which are also called **clauses**, can be proven. We will look at two techniques to achieve that.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Attribute Expansion

(Efficient) online learnability of *monotone* disjunctions from $X = \{ 0, 1 \}^n$ by an algorithm implies the same for clauses:

- Use the algorithm with $X' = \{ 0, 1 \}^{2n}$, presenting to it each x as

$$x'(x) = x^1, x^2, \ldots, x^n, 1 - x^1, 1 - x^2, \ldots, 1 - x^n$$

Reminder: superscripts are component indexes, not powers!

- The disjunction h' learned from X' is monotone but corresponds to the (non-monotone) disjunction

$$h = \bigvee_{\substack{i \leq n \\ \mathrm{p}_n \in \mathsf{Lits}(h')}} \mathrm{p}_i \bigvee_{\substack{i > n \\ \mathrm{p}_n \in \mathsf{Lits}(h')}} \neg \mathrm{p}_i$$

on X, i.e., $\forall x \in X : h(x) = h'(x'(x))$.

(*Exercise problem*)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Concept Inversion

(Efficient) online learnability of *conjunctions* from any X implies the same for clauses.

- Use the algorithm with inverted policy h', i.e.

$$h'(x) = 1 - h(x)$$

- When $h = \mathcal{L}_1 \wedge \mathcal{L}_2 \wedge \ldots \wedge \mathcal{L}_s$ (where $\mathcal{L}_i$ are literals) is a conjunction on X, i.e., then $h'(x)$ is the policy prescribed by

$$\neg h = \neg\mathcal{L}_1 \vee \neg\mathcal{L}_2 \vee \ldots \vee \neg\mathcal{L}_s$$

  which is a clause.

The reverse implication can be shown with the same reasoning.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability of Conjunctions and Clauses

With the two techniques, we can prove additional learnability results:

1. *clauses* are efficiently learnable online from $X = \{ 0, 1 \}^n$. (Proof: use Winnow + attribute expansion.)
2. *Conjunctions* are efficiently learnable online from $X = \{ 0, 1 \}^n$. (Proof: use Winnow + attribute expansion + concept inversion).
3. *clauses* are efficiently learnable online from $X =$ *contingent conjunctions*. (Proof: use generalization + concept inversion.)

Since a truth-value tuple can be represented by a contingent conjunction, assertion 2 already follows from the mistake bound of the generalization algorithm and assertion 3 implies assertion 1. However, Winnow used in 1 and 2 gives a better bound ($\mathcal{O}(\lg n)$) than generalization ($\mathcal{O}(n)$).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

s-DNF

An **s-conjunction** is a conjunction with at most s literals. An **s-DNF** is a disjunction of s-conjunctions.

For example, the "accepted form of authentication" concept description

$$\text{password} \vee (\text{fingerprint} \wedge \text{pin}) \vee (\text{facescan} \wedge \text{pin})$$

or the "accepted form of payment" concept description

$$\text{cash} \vee (\text{creditcard} \wedge \neg\text{expired})$$

are both 2-DNF but not a 1-DNF.

We will use the name s-DNF also to denote the *hypothesis class* of s-DNF's.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability of s-DNF

Let $c_1, c_2, \ldots c_{n'}$ be all s-conjunctions on n variables. Using a variant of the attribute expansion technique, we can reduce learning s-DNF from $X = \{ 0,1 \}^n$ to learning monotone disjunctions from $X' = \{ 0,1 \}^{n'}$.

- Use an algorithm for learning a monotone disjunction on n' variables, presenting to it each $x \in X$ as $x'(x) = x'^1, x'^2, \ldots x'^{n'}$ where

$$x'^i(x) = 1 \text{ iff } x \models c_i \tag{44}$$

- The hypothesis h' learned from x' is a monotone disjunction but corresponds to the s-DNF

$$\bigvee_{\{ i | \mathrm{p}_i \in \mathsf{Lits}(h') \}} c_i$$

on X, i.e., $\forall x \in X : h(x) = h'(x'(x))$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability of s-DNF (cont'd)

Assume we used an algorithm with mistake bound $poly(n')$. So s-DNF is learnable online from $X = \{0,1\}^n$ if $n' \leq poly(n)$.

The number n' of all s-conjunctions on n variables is the sum of the number of conjunctions with exactly $0, 1, \ldots s$ literals and the number of literals is twice the number of atoms, i.e. $2n$:

$$1 + \binom{2n}{1} + \binom{2n}{2} + \ldots + \binom{2n}{s} = \mathcal{O}(n^s) \leq poly(n)$$

Here we used an exponential upper bound for binomial coefficients.

So the class s-DNF (for any constant $s \in \mathbb{N}$) is efficiently learnable online from $X = \{0,1\}^n$. Determining (44) for each of the n' conjunctions takes linear time (verify) so s-DNF is also learnable *efficiently*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability of s-CNF

An ***s*-clause** is a clause with at most s literals. An ***s*-CNF** is a conjunction of s-clauses.

The class s-CNF is efficiently learnable online from $X = \{0,1\}^n$.

The proof is analogical to that of s-DNF learnability (the $c_1, c_2, \ldots c_{n'}$ are then all s-clauses).

(Learning an s-CNF from conjunctive observations is the subject of Project 1.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

s-term DNF and s-clause CNF

An **s-term DNF** is a disjunction of at most $s$conjunctions. An **s-clause CNF** is a conjunction of at most s clauses. The two names are also used to refer to the respective classes of formulas.

From the De Morgan's laws, it follows that each s-term DNF can be written as a tautologically equivalent s-CNF, by "multiplying out". Example with 2-term DNF:

$$(p_1 \wedge p_2 \wedge p_3) \vee (p_4 \wedge p_5) =$$
$$(p_1 \vee p_4) \wedge (p_1 \vee p_5) \wedge (p_2 \vee p_4) \wedge (p_2 \vee p_5) \wedge (p_3 \vee p_4) \wedge (p_3 \vee p_5)$$

Therefore

$$\mathcal{C}(s\text{-term DNF}) \subseteq \mathcal{C}(s\text{-CNF}) \tag{45}$$

Analogically, $\mathcal{C}(s\text{-clause CNF}) \subseteq \mathcal{C}(s\text{-DNF})$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learnability of s-term DNF and s-clause CNF

Since s-DNF is efficiently learnable online from $X = \{0,1\}^n$, so is s-clause CNF.

This is because due to (45), for each target s-clause CNF, the agent can learn an equivalent s-DNF.

For an analogical reason, s-clause CNF is also efficiently learnable online from $X = \{0,1\}^n$, with the agent using s-DNF.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Clausal Observations and Hypothesis

Consider a conjunction $h = h_{k+1}$ learned from contingent conjunctions $x_{\leq k}$, i.e.,

$$h = \text{lgg}(\{ x_{k_i} \}) \tag{46}$$

where $\{ x_{k_i} \}$ are exactly the positive examples from $x_{\leq k}$. (46) can be equivalently written as

$$\neg h = \text{lgg}(\{ \neg x_{k_i} \})$$

where $\neg h$ and all of $\neg x_{k_i}$ are *clauses* and the policy (27) is

$$h(x) = 1 \text{ iff } \neg h \models \neg x \tag{47}$$

because $\neg h \models \neg x$ is equivalent to $x \models h$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Disjunctive Observations and Hypothesis (cont'd)

For clausal hypotheses h and clausal observations x we define the decision policy

$$h(x) = 1 \text{ iff } h \models x \tag{48}$$

which is consistent with (47) where $\neg h$ and $\neg x$ are clauses.

Analogically to Theorem (1) we have the following evident principle

Theorem 4

Let h, h' be clauses. If $h \subseteq h'$ then $h \models h'$. Let furthermore h not be tautologically false. Then $h \subseteq h'$ if and only if $h \models h'$.

Thus if h is not tautologically false, (48) is equivalent to (33).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Disjunctive Observations and Hypothesis (cont'd)

The policy (33) is the same for for disjuctions and conjunctions, despite the difference btw. (48) and (27).

The mistake bound of the generalization algorithm asserted by Theorem (3) remains true even when X is assumed to consist of clauses. *(exercise problem.)*

So clauses can be learned from clauses online by the generalization algorithm with no modification of it.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Clausal Observations and Hypothesis

Clauses can be written as implications, e.g. $\neg \mathrm{p} \vee \mathrm{q} \models\!\!\models \mathrm{p} \rightarrow \mathrm{q}$, and interpreted as *rules*. Consider two positive clausal examples

$$x_1 = \mathrm{man} \wedge \mathrm{adult} \wedge \mathrm{young} \rightarrow \mathrm{married} \vee \mathrm{bachelor}$$
$$x_2 = \mathrm{man} \wedge \mathrm{adult} \rightarrow \mathrm{married} \vee \mathrm{bachelor} \vee \mathrm{nerd}$$

$h = \mathsf{lgg}(x_1, x_2) = \mathsf{Lits}(x_1) \cap \mathsf{Lits}(x_2) =$

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

The correct rule is thus learned from observed rules which are true but insufficiently general.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

First-Order Logic Observations

We will study the case when observations X are *first-order logic* (**FOL**, for short) conjunctions or clauses, i.e., conjunctions or disjunctions of FOL literals made using

- a finite set $\mathcal{P}$ of *predicates*
- a finite set $\mathcal{F}$ of *functions* (including constants)

Symbols x, y, z (typed in italics, possibly with indexes) will denote FOL *variables*.

Note: confusion of a FOL variable *x or y with an* observation *x resp.* action *y is excluded: FOL variables occur only as terms in predicates.*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

First-Order Logic Conjunctions and Clauses

If a FOL conjunction or clause is shown *without quantifiers* then all its variables are implicitly quantified

- *existentially*, for a *conjunction*
- *universally*, for a *clause*

So the negation of a FOL conjunction is a FOL clause and vice versa.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

FOL Subsumption

We now extend subsumption to FOL.

Let h, h' be two FOL conjunctions or two FOL clauses. We say that h **subsumes** h' (written $h \subseteq_\theta h'$) if there is a substitution θ such that $\text{Lits}(h\theta) \subseteq \text{Lits}(h')$. We say that h **strictly subsumes** h' (written $h \subset_\theta h'$) if $h \subseteq_\theta h'$ and $h' \not\subseteq_\theta h$.

Propositional formulas are special cases of FOL formulas with propositional variables corresponding to zero-arity predicates. So subsumption as defined above is identical to this definition for propositional conjunctions or clauses.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Subsumption

"There exist two vertices not connected by an edge":

$$h = \mathrm{vertex}(x) \wedge \mathrm{vertex}(y) \wedge \neg \mathrm{edge}(x, y)$$

"There exists a vertex not connected by an edge to vertex a":

$$h' = \mathrm{vertex}(x) \wedge \mathrm{vertex}(\mathsf{a}) \wedge \neg \mathrm{edge}(x, \mathsf{a})$$

$h \subseteq_\theta h'$ because for $\theta = \{ y \mapsto \mathsf{a} \}$, $\mathrm{Lits}(h) \subseteq \mathrm{Lits}(h')$.

$h' \not\subseteq_\theta h$ because the literal $\mathrm{vertex}(\mathsf{a})$ is not in h.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Equivalence and Reduction

Two FOL conjunctions or FOL clauses h, h' **are equivalent** if $h \subseteq_\theta h'$ and $h' \subseteq_\theta h$. We write $h \approx_\theta h'$.

Let h be a FOL conjunction or clause. We say that h **is reduced** if for no FOL conjunction (clause) h' such that $\text{Lits}(h') \subset \text{Lits}(h)$ it holds $h' \approx_\theta h$. We say that h' **is a reduction of** h if $h \approx_\theta h'$ and h' is reduced.

Algorithm returning a reduction of h:

- If there is a literal $\mathcal{L} \in h$ such that $h \subseteq_\theta h \setminus \mathcal{L}$ for some θ, set $h := h\theta$ and repeat, else return h.

$h \setminus \mathcal{L}$ denotes the result of removing literal $\mathcal{L}$ from h.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Reduction

$$h := \mathrm{vertex}(x) \wedge \mathrm{vertex}(y) \wedge \mathrm{vertex}(z) \wedge \neg\mathrm{edge}(x, y) \wedge \neg\mathrm{edge}(x, z)$$

$$\mathcal{L} = \mathrm{vertex}(z),\ h \subseteq_\theta h \setminus \mathcal{L} \text{ with } \theta = \{ z \mapsto y \},$$

$$h := h\theta = \mathrm{vertex}(x) \wedge \mathrm{vertex}(y) \wedge \neg\mathrm{edge}(x, y)$$

Note that *two* literals were removed by setting $h := h\theta$ above.

On the other hand,

$$h := \mathrm{vertex}(x) \wedge \mathrm{vertex}(y) \wedge \mathrm{vertex}(z) \wedge \neg\mathrm{edge}(x, y) \wedge \neg\mathrm{edge}(y, z)$$

is already reduced (cannot reduce further).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Subsumption vs. Consequence in FOL

We call a FOL conjunction or FOL clause **self-resolving** if it contains a positive literal and a negative literal both with the same predicate.

The relationship between $\subseteq_\theta$ and $\models$ for conjunctions (Theorem (1)) and clauses (Theorem (4)) is extended to FOL as follows:

Theorem 5

Let h, h' be FOL conjunctions. If $h \subseteq_\theta h'$ then $h' \models h$. Let furthermore h' not be self-resolving. Then $h \subseteq_\theta h'$ if and only if $h' \models h$.
Let h, h' be FOL clauses. If $h \subseteq_\theta h'$ then $h \models h'$. Let furthermore h not be self-resolving. Then $h \subseteq_\theta h'$ if and only if $h \models h'$.

Simple corollary: the theorem holds when $\subseteq_\theta$ is replaced by $\approx_\theta$ and $\models$ is replaced by $\models\!\!\models$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Subsumption vs. Consequence for Conjunctions

- Both $h' \models h$ and $h \subseteq_\theta h'$:

$$h = \mathrm{vertex}(x) \wedge \mathrm{vertex}(y) \wedge \neg\mathrm{edge}(x, y)$$
$$h' = \mathrm{vertex}(x) \wedge \mathrm{vertex}(\mathrm{a}) \wedge \neg\mathrm{edge}(x, \mathrm{a})$$

- $h' \models h$ but $h \not\subseteq_\theta h'$ because h is self-resolving (exercise problem):

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

- Similarly for clauses (just negating the h, h' above): $h \models h'$ but $h \not\subseteq_\theta h'$ because h is self-resolving:

$$h = \mathrm{p}(x, y) \wedge \mathrm{p}(y, z) \rightarrow \mathrm{p}(x, z)$$
$$h' = \mathrm{p}(\mathrm{a}, \mathrm{b}) \wedge \mathrm{p}(\mathrm{b}, \mathrm{c}) \wedge \mathrm{p}(\mathrm{c}, \mathrm{d}) \rightarrow \mathrm{p}(\mathrm{a}, \mathrm{d})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Decision Policy for FOL Clauses and Disjunctios

In generalization, we used policy (33) because the assumption of *contingent* propositional examples implies the assumptions of Theorem (5) so (33) is equivalent to (48) ((27), respectively) for clausal (conjunctive) examples.

The advantage of (33) is that it is decidable and enables us to use the least general generalization, and specifically the lgg operator for learning.

Similarly, when generalizing from FOL clausal (FOL conjunctive, respectively) hypotheses and examples, we use the policy

$$h(x) = 1 \text{ if } h \subseteq_\theta x \tag{49}$$

on the assumption that examples x are *not self-resolving*, which implies the assumptions of Theorem (5) making (49) equivalent to (48) ((27)). *(exercise)*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Least General Generalization in FOL

The definition of propositional least general generalization is extended to FOL simply by replacing the respective symbols $\subseteq, \subset$ by $\subseteq_\theta, \subset_\theta$.

For a pair of FOL conjunctions (FOL clauses, respectively) h, h', we say that g is a **least general generalization** of h and h' if $g \subseteq_\theta h$, $g \subseteq_\theta h'$, and there is no FOL conjunction (FOL clause) g' such that $g' \subseteq_\theta h$, $g' \subseteq_\theta h'$ and $g \subset_\theta g'$.

Example: $\mathrm{p}(x)$ is a least general generalization of clauses $\mathrm{p}(\mathrm{a})$ and $\mathrm{p}(\mathrm{b}) \vee \mathrm{p}(\mathrm{c})$. Other least general generalizations of the same literals include e.g. $\mathrm{p}(y)$ or $\mathrm{p}(x) \vee \mathrm{p}(y)$.

Let g, g' be two different least general generalizations of h and h'. Then $g \approx_\theta g'$. (*Exercise problem*)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg in FOL

Unlike in propositional logic, where the least general generalization is just set intersection, in FOL the computation is more involved.

We will develop the lgg operator which yields a least general generalization for FOL conjunctions or FOL clauses and reduces to set intersection in the propositional case.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg of FOL terms

For FOL terms t_1, t_2:

$$\mathrm{lgg}(t_1, t_2) = \begin{cases} s\left(\mathrm{lgg}(u_1, v_1), \mathrm{lgg}(u_2, v_2), \ldots, \mathrm{lgg}(u_a, v_a)\right) \\ \quad \text{if } t_1 = s(u_1, u_2, \ldots u_n) \text{ and } t_2 = s(v_1, v_2, \ldots v_n), \\ \quad \text{where } n \in \{0\} \cup \mathbb{N} \text{ and } s \text{ is a function symbol.} \\ v \text{ if } t_1, t_2 \text{ have already been matched to variable } v \\ \text{new variable otherwise} \end{cases}$$

Note that the definition above applies also to FOL constants since a constant is a function of arity 0.

Examples:

$$\mathrm{lgg}(\mathrm{john}, \mathrm{mary}) = x_1$$
$$\mathrm{lgg}\left(\mathrm{father\_of}(\mathrm{john}), \mathrm{father\_of}(\mathrm{mary})\right) = \mathrm{father\_of}(x_1)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg of FOL Literals

For FOL literals $\mathcal{L}_1, \mathcal{L}_2$:

$$\text{lgg}(\mathcal{L}_1, \mathcal{L}_2) = \begin{cases} p\left(\underline{\text{lgg}(u_1, v_1)}, \underline{\text{lgg}(u_2, v_2)}, \ldots, \underline{\text{lgg}(u_a, v_a)}\right) \\ \quad \text{if } \mathcal{L}_1 = p(u_1, u_2, \ldots u_n) \text{ and } \mathcal{L}_2 = p(v_1, v_2, \ldots v_n), \\ \quad \text{where } n \in \{0\} \cup \mathbb{N} \text{ and } p \text{ is a signed (negated or not) predicate symbol.} \\ \\ \text{undefined otherwise} \end{cases}$$

Examples:

$$\text{lgg}(\neg\text{p}, \neg\text{p}) = \neg\text{p}$$

$$\text{lgg}(\text{p}, \text{q}) \text{ undefined}$$

$$\text{lgg}\left(\begin{array}{c} \text{met}\left(\text{father\_of}(\text{john}), \text{mother\_of}(\text{john})\right), \\ \text{met}\left(\text{father\_of}(\text{mary}), \text{mother\_of}(\text{mary})\right) \end{array}\right) = \text{met}\left(\text{father\_of}(\text{x}), \text{mother\_of}(\text{x})\right)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Igg of FOL Literals in Pseudo-Code

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

Require: Literals a, b with same sign, symbol and arity, and not sharing a variable (if they share a variable, replace all occurrences of it in a with a new variable)

$i = 0$; $\theta := \emptyset$; $\sigma := \emptyset$ ▷ *a counter and two substitutions*
$v_1, v_2, \ldots$: 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

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg of FOL Conjunctions or Clauses

For FOL conjunctions (clauses, respectively) h, h', define $\text{lgg}(h, h')$ as the conjunction (disjunction) of

$$\text{lgg}(\mathcal{L}_i, \mathcal{L}_j) \tag{50}$$

for all $\mathcal{L}_i \in h$, $\mathcal{L}_j \in h'$ such that (50) is defined.

Theorem 6

Let h, h' be two FOL clauses or two FOL conjunctions. Then $\text{lgg}(h, h')$ is a least general generalization of h and h'.

(proof - deals with clauses but the case for conjunctions is analogical.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: lgg of Clauses

lgg of

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

and

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

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

θ	σ	
y	x	v_1
x	ann	v_2

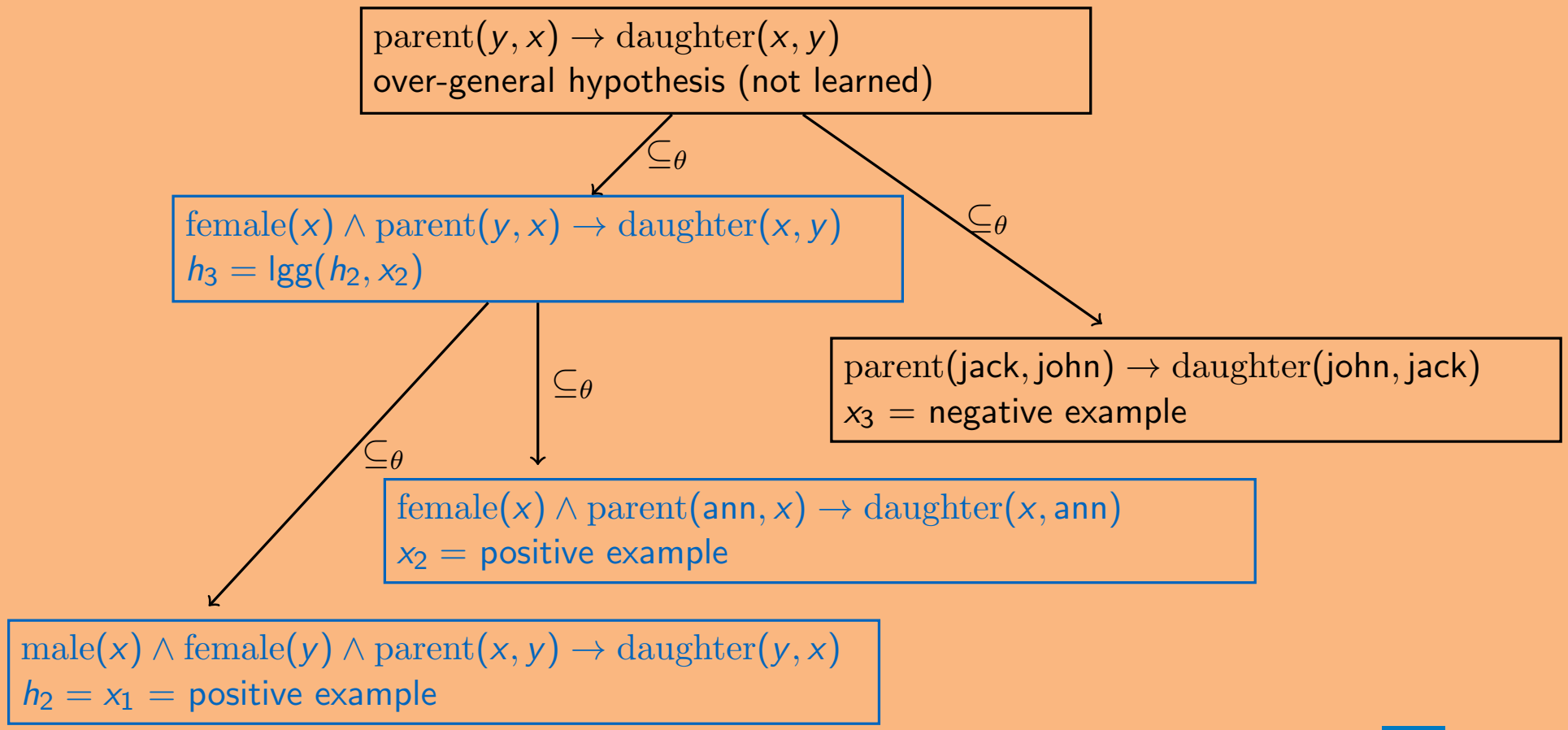
is

$$\text{female}(v_1) \wedge \text{parent}(v_2, v_1) \rightarrow \text{daughter}(v_1, v_2)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Generalizing FOL Clauses

The Generalization Algorithm with X = FOL clauses



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing FOL Conjunctions or Clauses

Because the lgg operator was extended to FOL, the generalization algorithm can be used with policy (49), assuming

$$X = \text{non-self-resolving FOL conjunctions} \tag{51}$$

or

$$X = \text{non-self-resolving FOL clauses} \tag{52}$$

producing the lgg of all positive examples. *(exercise problem)*

However, in the FOL case, we *cannot* prove a polynomial mistake bound analogical to the propositional case. In other words, Theorem (3) (where $n = |\mathcal{P}|$) does not hold if its assumption on X is replaced by (51) or (52), even if $|\mathcal{F}| = \emptyset$. *(Showing this is an exercise problem.)*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Background Knowledge: Motivation

Following the definition, $\text{lgg}(x_1, x_2)$ of

$$x_1 = \text{female}(x) \wedge \text{father}(y, x) \rightarrow \text{daughter}(x, y) \tag{53}$$

and

$$x_2 = \text{female}(x) \wedge \text{mother}(y, x) \rightarrow \text{daughter}(x, y) \tag{54}$$

is

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

This does not seem satisfactory since the 'natural' generalization would be

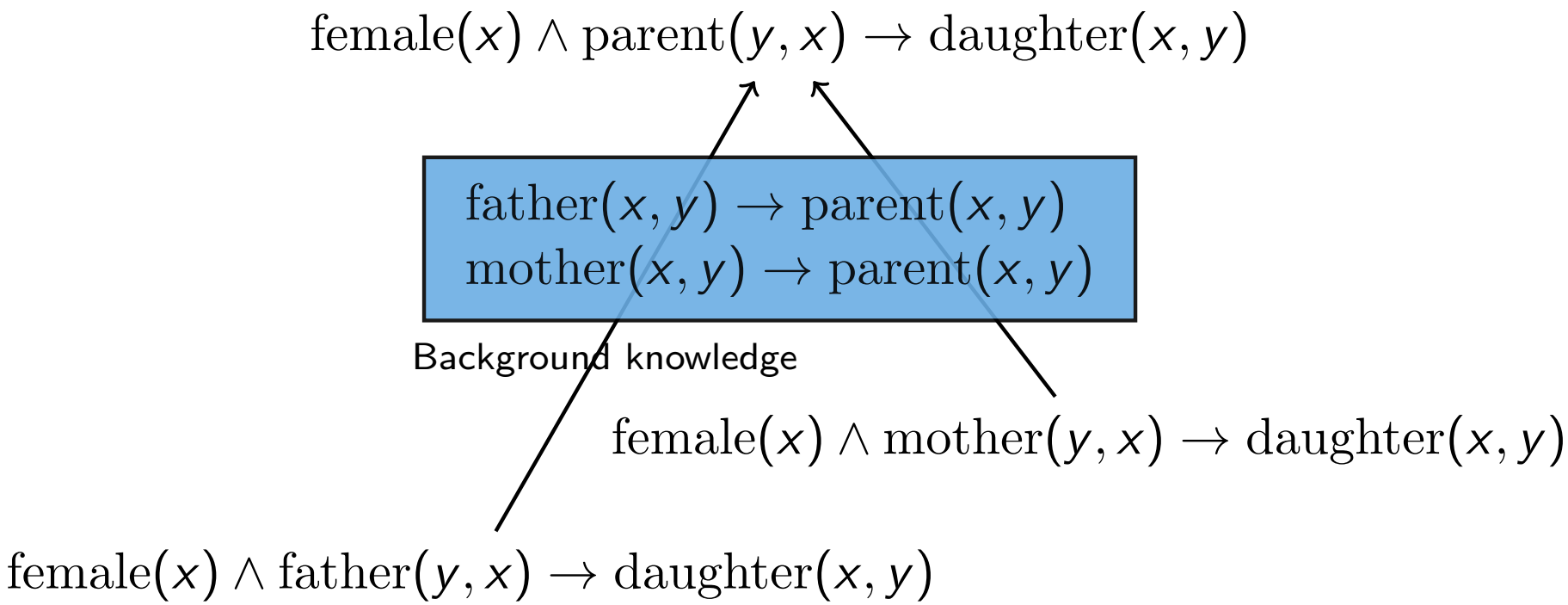
$$\text{female}(x) \wedge \text{parent}(y, x) \rightarrow \text{daughter}(x, y) \tag{55}$$

The learning agent would need to 'know' beforehand that father and mother are special cases of parent.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Background Knowledge: Motivation (cont'd)

We aim at generalization with respect to **background knowledge**, i.e., formal knowledge available prior to learning. Schematically:



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Decision Policy with Background Knowledge

For clausal hypotheses h, clausal observations x, *and background knowledge B*, we extend the consequence-based policy (48) to **relative consequence** policy

$$h(x) = 1 \text{ iff } B \wedge h \models x \tag{56}$$

Note that with $B =$

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

we have $B \wedge h \models x_1$, $B \wedge h \models x_2$ for x_1 (68), x_2 (69), h (55) as expected.

Just like we reduced (48) to (49) under the assumption on non-self-resolving examples, we investigate on which assumptions on x and B we reduce the relation in (56) to one based on $\subseteq_\theta$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Relative Subsumption

A formula is **ground** if it contains no variables.

Let h, h' be FOL clauses and B a ground FOL conjunction. We say that

- h **subsumes** h' **relative to** B (written $h \subseteq_\theta^B h'$) if

$$h \subseteq_\theta (B \to h') \tag{57}$$

- h **strictly subsumes** h' **relative to** B, if $h \subseteq_\theta^B h'$ but $h' \not\subseteq_\theta^B h$.

Theorem 7

Let h, h' be FOL clauses and B a ground FOL conjunction. If $h \subseteq_\theta h'$ then $h \subseteq_\theta^B h'$.

The proof is an exercise.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Relative Subsumption vs. Relative Consequence

Theorem 8

Let h, h' be FOL clauses. If $h \subseteq_\theta^B h'$ then $B \wedge h \models h'$. Let furthermore h not be self-resolving. Then $h \subseteq_\theta^B h'$ if and only if $B \wedge h \models h'$.

Proof: $h \models h'$ iff $\models h \rightarrow h'$ (i.e. iff the implication is tautologically true), thus $B \wedge h \models h'$ iff $h \models B \rightarrow h'$. Since B is a conjunction and h' a clause, $B \rightarrow h'$ is a clause. The asserted relationships between $B \wedge h \models h'$ (i.e., $h \models B \rightarrow h'$) and $h \subseteq_\theta^B h'$ (i.e., $h \subseteq_\theta B \rightarrow h'$) thus follow from Theorem (5).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Relative Subsumption: Example

$$B = \mathrm{male}(\mathsf{john}) \wedge \mathrm{parent}(\mathsf{ann}, \mathsf{john})$$
$$h = \mathrm{male}(x) \wedge \mathrm{parent}(y, x) \rightarrow \mathrm{son}(x, y)$$

We have

$$B \wedge h \models \mathrm{son}(\mathsf{john}, \mathsf{ann})$$

which can be shown through a resolution proof. But also

$$h \subseteq_\theta^B \mathrm{son}(\mathsf{john}, \mathsf{ann})$$

because

$$h \subseteq_\theta (\mathrm{male}(\mathsf{john}) \wedge \mathrm{parent}(\mathsf{ann}, \mathsf{john}) \rightarrow \mathrm{son}(\mathsf{john}, \mathsf{ann}))$$

Verify with $\theta = \{ x \mapsto \mathsf{john}, y \mapsto \mathsf{ann} \}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Relative Equivalence and Relative Reduction

For FOL clauses, we extend the equivalence and reduction definitions straightforwardly to account for background knowledge.

Let h, h' be FOL clauses and B a ground FOL conjunction. We say that

- h, h' **are equivalent relative to** B (written $h \approx_\theta^B h'$) if $h \subseteq_\theta^B h'$ and $h' \subseteq_\theta^B h$.
- h **is reduced relative to** B if for no FOL clause g such that $\text{Lits}(g) \subset \text{Lits}(h)$ it holds $g \approx_\theta^B h$.
- h' **is a reduction of** h **relative to** B if $h \approx_\theta^B h'$ and h' is reduced relative to B.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Relative Equivalence and Relative Reduction (cont'd)

To obtain a reduction of

$$h = h^1 \vee h^2 \vee \ldots h^{n_h}$$

relative to

$$B = b^1 \wedge b^2 \wedge \ldots b^{n_B}$$

first realize that any literal of h that is also in B with the opposite sign can be removed from h (obtaining h'). To see this for $h^i = \neg b^j$,

$$B \rightarrow h = b^1 \wedge \ldots \wedge b^j \wedge \ldots b^{n_B} \rightarrow h^1 \vee \ldots \vee h^i \vee \ldots h^{n_h}$$
$$\models\!\models \neg b^1 \vee \ldots \vee \neg b^j \vee \ldots \neg b^{n_B} \vee h^1 \vee \ldots \vee h^i \vee \ldots h^{n_h}$$

so the literal $h^i = \neg b^j$ occurs twice in $B \rightarrow h$. Thus $B \rightarrow h \approx_\theta B \rightarrow h'$ and so by definition, also $h \approx_\theta^B h'$. After the removal of all such literals, use the reduction algorithm on h'.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Relative Least General Generalization

Let h, h' be FOL clauses and B a ground FOL conjunction. g is a **least general generalization of h and h' relative to B** if $g \subseteq_\theta^B h$, $g \subseteq_\theta^B h'$, and there is no FOL clause g' such that $g \subset_\theta^B g'$, $g' \subseteq_\theta^B h$, $g' \subseteq_\theta^B h'$.

In other words, g is a least general generalization of h and h' relative to B iff it is a least general generalization of $B \to h$ and $B \to h'$. *To see this, check the definitions of $\subseteq_\theta^B$, $\subset_\theta^B$.*

Define $\mathrm{rlgg}_B(h, h')$ as $\mathrm{lgg}(B \to h, B \to h')$. Corollary of Theorem (6): $\mathrm{rlgg}_B(h, h')$ is a least general generalization of h and h' relative to B.

(Implementing rlgg *with a subsequent clause reduction is the subject of Project 2.)*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Relative Least General Generalization

$$B = \text{female}(\mathsf{a}) \wedge \text{parent}(\mathsf{a}, \mathsf{b}) \wedge \text{male}(\mathsf{b}) \wedge \text{parent}(\mathsf{b}, \mathsf{c}) \wedge \text{male}(\mathsf{c}) \quad (58)$$

$$x_1 = \text{son}(\mathsf{b}, \mathsf{a})$$
$$x_2 = \text{son}(\mathsf{c}, \mathsf{b})$$
$$\text{rlgg}_B(x_1, x_2) = \text{lgg}(B \to x_1, B \to x_2)$$

Rewrite $B \to x_1$ as

$$\text{son}(\mathsf{b}, \mathsf{a}) \vee \neg\text{female}(\mathsf{a}) \vee \neg\text{parent}(\mathsf{a}, \mathsf{b}) \vee \neg\text{male}(\mathsf{b}) \vee \neg\text{parent}(\mathsf{b}, \mathsf{c}) \vee \neg\text{male}(\mathsf{c})$$

Rewrite $B \to x_2$ as

$$\text{son}(\mathsf{c}, \mathsf{b}) \vee \neg\text{female}(\mathsf{a}) \vee \neg\text{parent}(\mathsf{a}, \mathsf{b}) \vee \neg\text{male}(\mathsf{b}) \vee \neg\text{parent}(\mathsf{b}, \mathsf{c}) \vee \neg\text{male}(\mathsf{c})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Relative Least General Generalization (cont'd)

Now compute the lgg as in the older example. Below, only the first letters of predicate symbols are shown.

	$s(c,b)$	$\neg f(a)$	$\neg p(a,b)$	$\neg m(b)$	$\neg p(b,c)$	$\neg m(c)$
$s(b,a)$	$s(v_1,v_2)$					
$\neg f(a)$		$\neg f(a)$				
$\neg p(a,b)$			$\neg p(a,b)$		$\neg p(v_2,v_1)$	
$\neg m(b)$				$\neg m(b)$		$\neg m(v_1)$
$\neg p(b,c)$			$\neg p(v_3,v_4)$		$\neg p(b,c)$	
$\neg m(c)$				$\neg m(v_4)$		$\neg m(c)$

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

So $\mathrm{rlgg}_B(x_1,x_2) =$

$$s(v_1,v_2) \vee \neg f(a) \vee \neg p(a,b) \vee \neg p(v_2,v_1) \vee \neg m(b) \vee$$
$$\neg m(v_1) \vee \neg p(v_3,v_4) \vee \neg p(b,c) \vee \neg m(v_4) \vee \neg m(c) \qquad (59)$$

Now reduce the clause.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Relative Clause Reduction

Reduce $h =$ (59) relative to $B =$ (58).

First remove literals of h that are in B with the opposite sign, obtaining

$$h := \mathrm{s}(v_1, v_2) \vee \neg\mathrm{p}(v_2, v_1) \vee \neg\mathrm{m}(v_1) \vee \neg\mathrm{p}(v_3, v_4) \vee \neg\mathrm{m}(v_4)$$

Now follow the reduction algorithm

$$h \subseteq_\theta (h \setminus \neg\mathrm{p}(v_3, v_4)) \text{ with } \theta = \{ v_3 \mapsto v_1, v_4 \mapsto v_2 \}$$
$$h := h\theta = \mathrm{s}(v_1, v_2) \vee \neg\mathrm{p}(v_2, v_1) \vee \neg\mathrm{m}(v_1)$$

Rewriting h into the implication form and plugging in the full predicate symbols, we get

$$\mathrm{parent}(v_2, v_1) \wedge \mathrm{male}(v_1) \rightarrow \mathrm{son}(v_1, v_2)$$

which is the expected generalization. *(exercise problem)*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Herbrand Semantics

For a finite set $\mathcal{P}$ of predicates and a finite set $\mathcal{F}$ of functions (including constants)

- the **Herbrand base** HB is the set of all ground atoms made using $\mathcal{P}$ and $\mathcal{F}$.
- a **Herbrand interpretation** HI is a subset of the HB.

A HI is a special case of FOL interpretation so we write $HI \models \phi$ whenever HI is a model of FOL formula ϕ.

In propositional logic as a special case of FOL, a HI is just a truth assignment to all the unary atoms, more precisely, the set of all atoms assigned the True value.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

st-CNF

A positive learnability result for a FOL hypothesis class can be established if we impose *size bounds* on the target hypothesis, like we did for *s*-DNF, *s*-CNF, *s*-term DNF, and *s*-clause CNF.

An ***st*-clause** is a FOL clause with at most s literals and at most t term occurrences in each literal. So e.g.

$$\mathrm{born}(x) \rightarrow \mathrm{reproduced}(\mathsf{mother}(x), \mathsf{father}(x))$$

has 2 literals with 1 term occurrence in the LHS literal and 4 term occurrences in the RHS literal. So it is a 2,4-clause but not e.g. a 2,3-clause.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability of *st*-CNF

Consider learning from Herbrand interpretations, i.e. FOL analogies of truth assignments $X = \{0, 1\}^n$ to propositional variables.

Theorem 9

Let X contain Herbrand interpretations for a finite set of $\mathcal{P}$ predicates and a finite set $\mathcal{F}$ of functions, and the observation complexity n_X be the tuple $(|\mathcal{P}|, |\mathcal{F}|)$. The hypothesis class st-CNF is learnable online from X.

The proof is an exercise.

As observations have to be finite, Herbrand interpretations are strictly *less expressive* than clausal or conjunctive observations. For example, with $\mathcal{P} = \{\mathrm{p}/1\}$ and $\mathcal{F} = \{\mathrm{a}/0, \mathrm{f}/1\}$, $\mathrm{p}(x)$ has exactly one Herbrand model, which is the *infinite* Herbrand interpretation $\{\mathrm{p(a)}, \mathrm{p(f(a))}, \mathrm{p(f(f(a)))}, \ldots\}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Version Space

Theorem 10

A finite hypothesis class $\mathcal{H}$ is learnable online from X if $\lg|\mathcal{H}| \leq poly(n_X)$.

Proof: the **version space** algorithm below has the mistake bound $\lg|\mathcal{H}|$ so if $\lg|\mathcal{H}| \leq poly(n_X)$ the version space learns $\mathcal{H}$ online from X.

Define a decision policy for a *set of hypotheses* as the *majority vote* among the hypotheses in it

$$\mathcal{H}(x) = \begin{cases} 1 \text{ if } |\{\, h \in \mathcal{H} \;:\; h(x) = 1 \,\}| > |\mathcal{H}|/2 \\ 0 \text{ otherwise} \end{cases} \tag{60}$$

Instead of a single hypothesis h_k, the version-space agent maintains a set $\mathcal{H}_k$ of hypotheses at each time $k \in \mathbb{N}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Version Space (cont'd)

At $\forall k \in \mathbb{N}$

$$y_{k+1} = \mathcal{H}_k(x_k)$$

where $\mathcal{H}_1 = \mathcal{H}$ and $\mathcal{H}_{k+1}$ keeps exactly those hypotheses from $\mathcal{H}_k$ which gave the correct decision for x_k:

$$\mathcal{H}_{k+1} = \{\, h \in \mathcal{H}_k : h(x_k) = \overline{y}_k \,\} \tag{61}$$

where $\overline{y}_k = |y_k + r_{k+1}|$ is the true class according to the target hypothesis.

At least half of the hypotheses from $\mathcal{H}_k$ is deleted by (61) on each mistake ($r_k = -1$), because at least half of them were wrong according to (60). In the worst case, the single remaining hypothesis is the target hypothesis, which will no longer make mistakes. So $\lg |\mathcal{H}_1| = \lg |\mathcal{H}|$ is the maximum number of mistakes.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability due to Polynomial lg $|\mathcal{H}|$

Theorem (10) implies online learnability of the hypothesis classes

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3$$

$\mathcal{H}_1$= monotone conjunctions (clauses, respectively): $|\mathcal{H}| = 2^n$ (each atom present or absent in it).

$\mathcal{H}_2$= contingent conjunctions (clauses, respectively): $|\mathcal{H}| = 3^n$ (each atom absent, present in a positive literal or present in a negative literal)

$\mathcal{H}_3$= conjunctions (clauses, respectively): $|\mathcal{H}| = 2^{2n}$ (each of $2n$ literals present or absent)

from $X = \{0,1\}^n$ when $n_X = n$ due to $\lg |\mathcal{H}| \leq poly(n)$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Version Space vs. Algorithms for Specific $\mathcal{H}$

We already know these classes are *efficiently* learnable online from $X = \{0,1\}^n$ by Winnow with mistake bound $\mathcal{O}(\lg n)$. Version space has bound $\lg |\mathcal{H}| = \mathcal{O}(n)$ in all these cases and is *not efficient* as it loops over an exponential-size set $\mathcal{H}$ in (60) and (61).

Contingent conjunctions (clauses) are also *efficiently* learnable online from contingent conjunctions (clauses) by generalization with bound $\mathcal{O}(n)$ where $n_X = n$ is the maximum number of variables in the conjunctions (clauses). The version space bound $\lg |\mathcal{H}| = \mathcal{O}(n)$ is the same but again, version space is *not effificent*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability due to Polynomial $\lg|\mathcal{H}|$ (cont'd)

Further classes learnable online from $X = \{0,1\}^n$:

$\mathcal{H} =$ s-conjunctions or s-clauses: not only $\lg|\mathcal{H}| \leq poly(n)$ but $|\mathcal{H}| \leq poly(n)$. So version space learns them online with logarithmic mistake bound and *efficiently* because (60) and (61) can be evaluated in $poly(n)$-time.

$\mathcal{H} =$ s-DNF or s-CNF: These include a subset of the $poly(n)$-number of s-conjunctions (s-clauses, respectively), so $|\mathcal{H}| = 2^{poly(n)}$, i.e. $\lg|\mathcal{H}| = poly(n)$. So version space learns them online with logarithmic mistake bound but not efficiently. In contrast, they can be learned online efficiently by reduction to monotone disjunctions (conjunctions).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

VC Dimension

We say that concept class $\mathcal{C}$ **shatters** a set of observations $X' \subseteq X$ if for every subset $X'' \subseteq X'$ there is a concept $C \in \mathcal{C}$ such that $C \cap X' = X''$.

In other words, X' is shattered by $\mathcal{C}$ if it can be split by concepts from $\mathcal{C}$ in all $2^{|X'|}$ possible ways.

Vapnik-Chervonenkis Dimension

The **VC-dimension of concept class $\mathcal{C}$ on X** denoted $\mathrm{VC}(\mathcal{C})$ is

$$\max\{\ |X'|\ :\ \mathcal{C} \text{ shatters } X',\ X' \subseteq X\ \}$$

For a hypothesis class $\mathcal{H}$, we abbreviate $\mathrm{VC}(\mathcal{C}(\mathcal{H}))$ as $\mathrm{VC}(\mathcal{H})$ and call the latter the VC-dimension of $\mathcal{H}$.

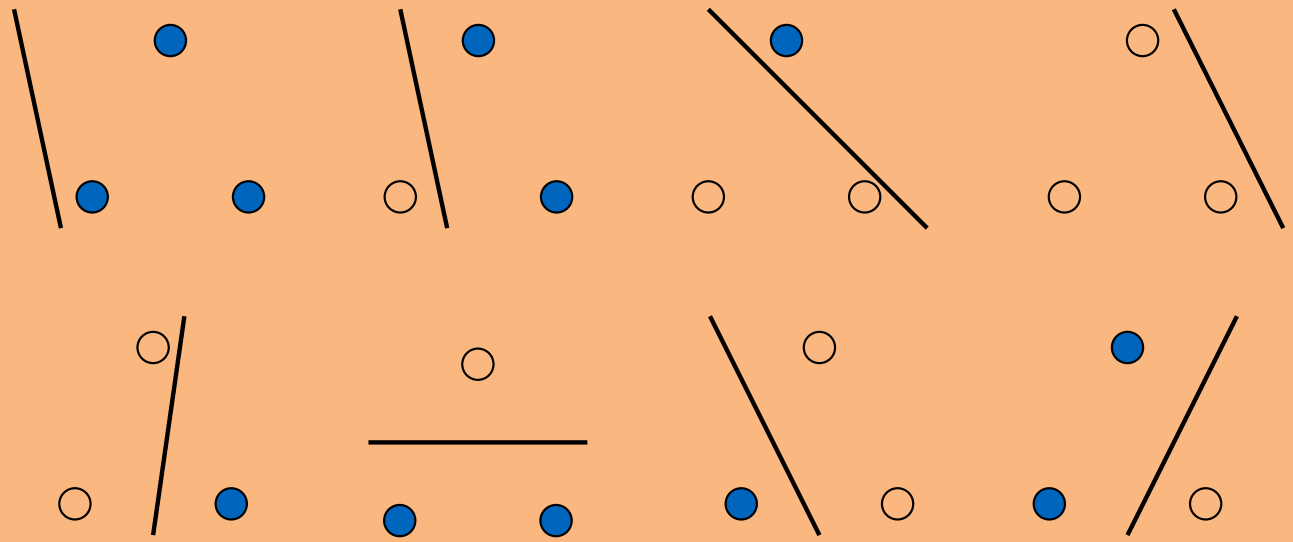
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Determining VC-Dimension

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

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 colored 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 $\text{VC}(\mathcal{C}) = 3$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Poly VC-Dimension Necessary for Learnability

Theorem 11

Concept class $\mathcal{C}$ on X is learnable online from X only if $\mathrm{VC}(\mathcal{C}) \leq poly(n_X)$.

Proof: There exists a set of $\mathrm{VC}(\mathcal{C})$ observations from X shattered by $\mathcal{C}$ so there exists a sequence $x_{\leq \mathrm{VC}(\mathcal{C})}$ of observations such that for any sequence of the learner's decisions $y_2, y_3, \ldots, y_{\leq \mathrm{VC}(\mathcal{C})+1}$ there is a target concept $C \in \mathcal{C}$ on which this sequence results in $\mathrm{VC}(\mathcal{C})$ mistakes, i.e., $r_2, r_3, \ldots, r_{\mathrm{VC}(\mathcal{C})+1} = \underbrace{(-1, -1, \ldots -1)}_{\mathrm{VC}(\mathcal{C})}$, thus $\sum_{k=1}^{\infty} |r_k| \geq \mathrm{VC}(\mathcal{C})$. So if $\mathcal{C}$ is learnable, i.e. if $\sum_{k=1}^{\infty} |r_k|$ is polynomially bounded, $\mathrm{VC}(\mathcal{C})$ is also polynomially bounded.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

$\lg|\mathcal{H}|$ vs. $\mathrm{VC}(\mathcal{H})$

Theorem (11) holds for any concept class $\mathcal{C}$, including a concept class $\mathcal{C}(\mathcal{H})$ induced by a hypothesis class $\mathcal{H}$. So if $\mathcal{H}$ is learnable online from X then $\mathrm{VC}(\mathcal{H}) \leq poly(n_X)$.

Combining this with Theorem (10), we get that $\mathrm{VC}(\mathcal{H})$ must be polynomial whenever $\lg|\mathcal{H}|$ is.

Note that in Theorem (10), we cannot replace "hypothesis class" with "concept class" and $\mathcal{H}$ with $\mathcal{C}$, because there may be concepts in $\mathcal{C}$ that have no finite description by a hypothesis, thus the assumption in the Theorem would not be sufficient for online learnability.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Standard Agent

An agent is called **standard** if it changes its hypothesis iff an error is made, i.e. $h_{k+1} \neq h_k$ iff $r_{k+1} = -1$. So Winnow and the generalization agent are both standard while version space is not.

Lemma 4

Let $r_{\leq K}$ be a reward sequence of a standard agent and $h_{\leq K}$ be its sequence of hypotheses. Denote $M = \sum_{k=1}^{K} |r_k|$. There is a hypothesis h retained for at least $\frac{K}{M+1}$ consecutive steps in $h_{\leq K}$, i.e.

$$h_{\leq K} = h_1, h_2, \ldots \underbrace{h, h, \ldots h}_{\text{at least } \frac{K}{M+1} \text{ times}}, \ldots h_K$$

Proof: exercise

If $\frac{K}{M+1}$ is non-integer than "at least $\frac{K}{M+1}$" means the same as "at least the nearest integer largest than $\frac{K}{M+1}$".

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

I.I.D. Observations and Hypothesis Error

Assume (14) for the rest of the concept-learning chapter.

Given target concept $\overline{C}$, define the **error of hypothesis** h as

$$\text{err}(h) = P(\overline{C} \triangle C(h))$$

where $\triangle$ is the symmetric set difference. *Recall the definition of C(h).*

So err(h) is the

- total probability of the error region on X with respect to the distribution (14).
- probability that policy $h(x)$ is not the true class of x, i.e., not the optimal action (15).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

The PAC Learning Model

Unlike the mistake bound model, the PAC-learning model does not require a finite bound on mistakes. It requires finding a *low-error hypothesis* with *high probability* using a *polynomial number of observations*.

Probably Approximately Correct (PAC) Learning Model

In the concept classification protocol, an agent **probably approximately correctly (PAC) learns $\mathcal{C}$ from X** if for any target concept from $\mathcal{C}$ on X, an arbitrary distribution (14) and arbitrary numbers $0 < \epsilon, \delta < 1$, there is a $K < poly(n_X, 1/\epsilon, 1/\delta)$ such that with probability at least $1 - \delta$, the agent's hypothesis h_K has $\text{err}(h_K) \leq \epsilon$. It PAC-learns $\mathcal{C}$ from X **efficiently**, if in addition, the time taken to compute an action from an observation is also at most polynomial in $n_X, 1/\epsilon, 1/\delta$.

$\mathcal{C}$ is **PAC-learnable** if there is an algorithm that PAC-learns it.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability Implies PAC-Learnability

Call a hypothesis h **consistent** with $x \in X$ if $h(x)$ is the true class of x, i.e. $h(x) = 1$ iff x is in the target concept.

Theorem 12

If $\mathcal{C}$ is learnable online from X by a standard agent, then $\mathcal{C}$ is PAC-learnable from X.

The standard-agent assumption is not needed in the theorem but it is a rather weak assumption making the proof much easier.

Proof: Online learnability of $\mathcal{C}$ from X means $M = \sum_{k=1}^{\infty} |r_k| \leq poly(n_X)$. Due to Lemma (4), for an arbitrary $K \in \mathbb{N}$, some hypothesis h was retained by the agent for $q = \frac{K}{M+1}$ steps before time K. Since the learning agent is standard, h is consistent with q consecutive observations.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability Implies PAC-Learnability (cont'd)

To show that the agent PAC-learns $\mathcal{C}$, we want to show that $\text{err}(h) \leq \epsilon$ with probability at least $1 - \delta$ when the observations are i.i.d. from (14).

Call h **bad** if $\text{err}(h) > \epsilon$. The probability that a h is consistent with an observation from (14) is $1 - \text{err}(h)$, so if h is bad, it is at most $(1 - \epsilon)$. Because h is consistent with q such i.i.d. consecutive observations, the probability that h is bad is at most $(1 - \epsilon)^q$.

Note that it is important for the above reasoning that h is consistent with q observations, which are consecutive. For example, we could not 'pick' an arbitrary selection of q correctly classified observation from an arbitrarily long history $x_{\leq k}$ (that would not be a series of Bernoulli trials).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Online Learnability Implies PAC-Learnability (cont'd)

We want the probability $(1-\epsilon)^q$ to be less than δ. Use the upper bound

$$(1-\epsilon)^q < e^{-\epsilon q} \tag{62}$$

which holds for all $\epsilon > 0$. So $(1-\epsilon)^q < \delta$ if $e^{-\epsilon q} \leq \delta$. The latter is satisfied if

$$q = \frac{1}{\epsilon} \ln \frac{1}{\delta} \tag{63}$$

because $e^{-\epsilon \frac{1}{\epsilon} \ln \frac{1}{\delta}} = \delta$. Since $q = \frac{K}{M+1}$, we have $K = (M+1)q$ where $M+1 \leq poly(n_X)$ and $q \leq poly(\frac{1}{\epsilon}, \frac{1}{\delta})$, therefore $K \leq poly(n_X, \frac{1}{\epsilon}, \frac{1}{\delta})$.

Thus with probability *at least* δ, h_K is *not bad* ($\text{err}(h) \leq \epsilon$) for $K \leq poly(n_X, \frac{1}{\epsilon}, \frac{1}{\delta})$ so the agent PAC-learns $\mathcal{C}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learnability Implies Consistent Learning

The following theorem states that in the definition of PAC-Learning, the hypothesis h_K is necessarily consistent with all observations up to $K-1$.

Theorem 13

Let an agent PAC-learn $\mathcal{C}$ from X. Then for any target concept from $\mathcal{C}$ on X, an arbitrary distribution (14) and arbitrary numbers $0 < \epsilon, \delta < 1$ and $K \in \mathbb{N}$, the condition $\mathrm{err}(h_K) \leq \epsilon$ with probability at least $1-\delta$ implies that h_K is consistent with all observations in $x_{<K}$.

Proof: exercise

Consistent Agent

$\mathcal{H}$-consistent learning agent is an agent whose h_{k+1} ($k \in \mathbb{N}$) is an arbitrary hypothesis from $\mathcal{H}$ that is consistent with $x_{\leq k}$ if such a hypothesis exists.

When the target concept is chosen from a concept class $\mathcal{C}$, the condition

$$\mathcal{C} \subseteq \mathcal{C}(\mathcal{H}) \tag{64}$$

(where $\mathcal{C}(\mathcal{H})$ is defined by (22)) guarantees that a hypothesis consistent with any $x_{\leq k}$ exists in $\mathcal{H}$.

Note that an $\mathcal{H}$-consistent agent may make mistakes even when (64) is satisfied but after each mistake ($r_{k+1} = -1$), h_{k+1} is chosen such that it is consistent with all of $x_{\leq k}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Consistent Agent (cont'd)

Recall we are assuming observations x_k, $k \in \mathbb{N}$ to be i.i.d.!

Lemma 5

For the hypothesis h_{k+1} of a $\mathcal{H}$-consistent agent satisfying (64), $\mathrm{err}(h_{k+1}) \leq \epsilon$ *with probability at least* $1 - \delta$ *if*

$$k \geq \frac{1}{\epsilon} \ln \frac{|\mathcal{H}|}{\delta} \tag{65}$$

Proof: Due to the assumption (64), $\mathcal{H}$ contains a hypothesis consistent with an arbitrary sequence of observations $x_{\leq k}$, so the agent can pick one at each $k+1$ and the agent has at most $|\mathcal{H}|$ possible choices. The probability that one such chosen hypothesis $h_{k+1} \in \mathcal{H}$ consistent with $x_{\leq k}$ is bad ($\mathrm{err}(h_{k+1}) > \epsilon$) is $(1 - \mathrm{err}(h_{k+1}))^k < (1 - \epsilon)^k$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Consistent Learning with with Polynomial lg $|\mathcal{H}|$

The probability that *some* of the $|\mathcal{H}|$ possible choices is bad is thus at most

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

where we used the bound (62). This is smaller than δ if (65) holds. This completes the proof. We get the following theorem as a corollary.

Theorem 14

An $\mathcal{H}$-consistent learning agent satisfying (64) PAC-learns from X any concept class $\mathcal{C}$ on X if $\lg|\mathcal{H}| \leq poly(n_X)$.

Proof: By Lemma (5), an $\mathcal{H}$-consistent learning agent satisfying (64) has $\mathrm{err}(h_k) \leq \epsilon$ with probability at least $1-\delta$ for k polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}$ (65). The requirement $\lg|\mathcal{H}| \leq poly(n_X)$ means that k is also polynomial in $poly(n_X)$, thus the agent PAC-learns $\mathcal{C}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Consistent Learning with with Polynomial VC($\mathcal{H}$)

Lemma 6

For the hypothesis h_{k+1} of a $\mathcal{H}$-consistent agent satisfying (64), $\mathrm{err}(h_{k+1}) \leq \epsilon$ *with probability at least* $1 - \delta$ *if*

$$k \geq \max\left\{\frac{4}{\epsilon}\lg\frac{2}{\delta}, \frac{8 \cdot \mathrm{VC}(\mathcal{H})}{\epsilon}\lg\frac{13}{\epsilon}\right\} \tag{66}$$

Proof omitted due to length and technicality.

$\mathrm{VC}(\mathcal{H})$ may be finite and polynomial even when $|\mathcal{H}|$ is infinite and $\lg|\mathcal{H}|$ is not defined. Even for a finite $|\mathcal{H}|$, (66) *may give a better (smaller) bound than* (65). *(Exercise problem)*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Consistent Learning with with Polynomial VC($\mathcal{H}$) (cont'd)

A corollary of Lemma (6) is analogical to Theorem (14).

Theorem 15

An $\mathcal{H}$-consistent learning agent satisfying (64) PAC-learns from X any concept class $\mathcal{C}$ on X if $\mathrm{VC}|\mathcal{H}| \leq poly(n_X)$.

The proof is also analogical to that of Theorem (14), except it relies on Lemma (6) instead of Lemma (5).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Proper PAC-Learning

We have shown s-term DNF (s-clause CNF, respectively) to be learnable online from $X = \{0,1\}^n$ because it is a subset of the class s-CNF (s-DNF) which is learnable (by a standard agent) from X.

So by Theorem (12), the two classes are also PAC-Learnable from X, which means that the agent finds a hypothesis h_K such that $\mathrm{err}(h_K) < \epsilon$ with probability at least $1 - \delta$ where $K \leq poly(\frac{1}{\epsilon}, \frac{1}{\delta}, n_X)$.

h_K is a s-CNF (s-DNF) and generally, *it cannot be rewritten into an equivalent s-term DNF (s-clause CNF)*.

We will define a stronger version of PAC-learning which requires that h_K belongs to the hypothesis class from which the target hypothesis is chosen.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Proper PAC-Learning

Proper PAC Learning Model

Let $\mathcal{H}$ be a hypothesis class. An agent (efficiently) **properly PAC-learns** $\mathcal{H}$ from an observation class X if all conditions for (efficient) PAC-learning of $\mathcal{C}(\mathcal{H})$ are satisfied, and, in addition, for the h_K in the definition it holds $h_K \in \mathcal{H}$. A hypothesis class $\mathcal{H}$ is (efficiently) **properly PAC-learnable** from X if there is an agent (efficiently) properly PAC-learning $\mathcal{H}$ from X.

Proper PAC-learning is important e.g. when h_K is to be interpreted by a human and its membership in $\mathcal{H}$ guarantees readability.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Efficiently Properly PAC-Learnable Classes

Given Theorem (12), a hypothesis class $\mathcal{H}$ is efficiently properly PAC-learnable from X if there is a standard agent that efficiently learns $\mathcal{H}$ online from X and the hypotheses h_k the agent uses as decision policies are all from $\mathcal{H}$.

For example, *conjunctions* (*clauses*, respectively) are efficiently properly PAC-learnable from $X = \{0,1\}^n$ or from $X =$ contingent conjunctions (clauses) because they are learnable online efficiently with the generalization algorithm, and all h_k are conjunctions (clauses). *(Unlike Winnow, where h_k are hyperplanes!)*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

(Non)-Learnability of s-term DNF and s-clause CNF

We already know that s-term DNF is efficiently learnable online from $X = \{0,1\}^n$ by a standard agent thus also *efficiently* PAC-learnable from X. It is also *properly* PAC-learnable from X due to Theorem (14) and the fact that $\lg |s\text{-CNF}| \leq poly(n)$ and $\mathcal{C}(s\text{-term DNF}) \subseteq \mathcal{C}(s\text{-CNF})$.

The same holds analogically for s-clause CNF. Are these classes also *efficienty properly* learnable?

Theorem 16

None of s-term DNF and s-clause CNF is efficiently properly PAC-learnable from $X = \{0,1\}^n$

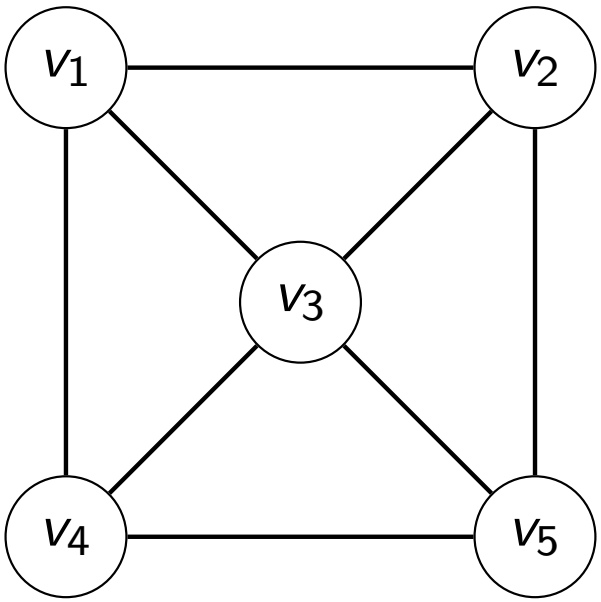
Proof: We will show the proof only for the special case of 3-term DNF. The NP-complete graph 3-coloring problem can be reduced in poly-time to finding an 3-term DNF consistent with a finite set of observations.

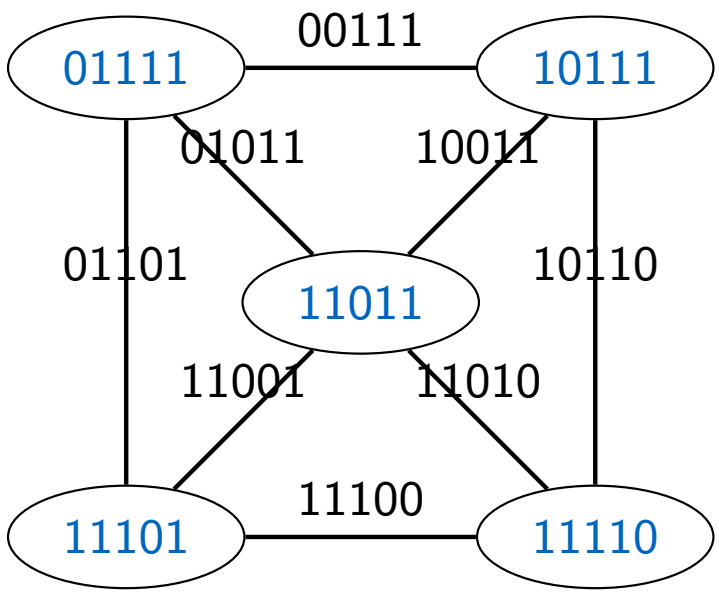
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

3-term DNF's are not efficiently properly PAC-learnable.

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

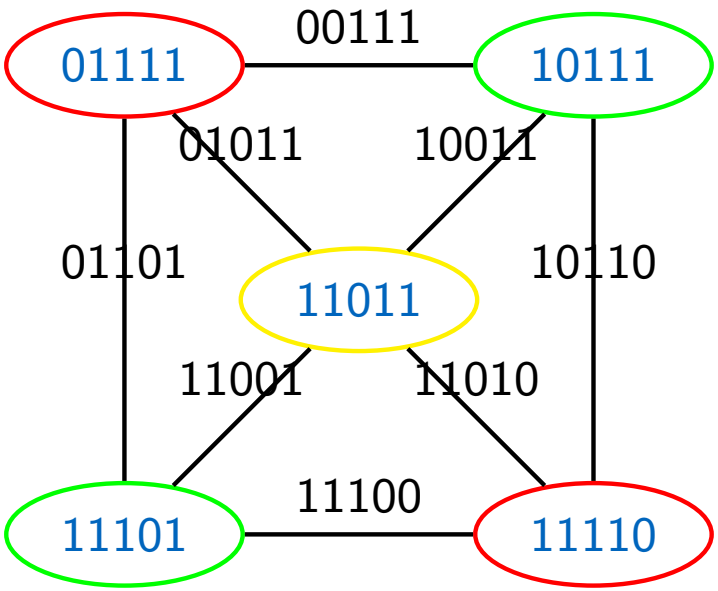
edge e_{ij} $\leftrightarrow$ neg. example x, $x^l = \begin{cases} 0 \text{ if } l = i \text{ or } l = j \\ 1 \text{ otherwise} \end{cases}$





3-term DNF's are not efficiently properly PAC-learnable.

Graph 3-colorable iff a 3-term DNF consistent with the observations.



$\Rightarrow$

$\bigvee$ **color** $\in$ $\{ R, G, Y \}$ $\quad$ $\bigwedge_i$ v_i not of **color** $\quad$ p_i

$\Leftarrow$

color of any consistent term

$p_2 \wedge p_3 \wedge p_4$

$\vee$

$p_1 \wedge p_3 \wedge p_5$

$\vee$

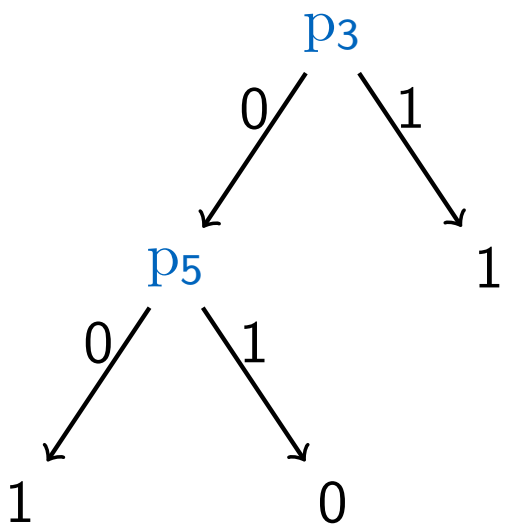
$p_1 \wedge p_2 \wedge p_4 \wedge p_5$

3-colorability NP-hard $\rightarrow$ finding a consistent 3-term DNF NP-hard.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

s-Decision Trees

Example:



3-Decision Tree

A **decision tree** on $X = \{0,1\}^n$ is a binary tree graph where each non-leaf vertex indicates one of the n components, each leaf is a class from Y, and each edge is labeled 0 or 1.
It prescribes a policy for $x \in X$: go from the root, always following one of the two outgoing edges that is labeled with the value of the component in the last vertex, until in a leaf.
The leaf is the decision.
For example, $x = (0,1,0,1,1)$ is decided as $y = 0$ by the tree on the left.
An ***s*-decision tree** has *depth* s or less.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learnability of s-Decision Trees

Theorem 17

The class s-Decision trees is PAC-learnable from $X = \{0,1\}^n$ efficiently or properly but not efficiently properly.

Proof: For any s-DT, there is an equivalent s-DNF: create an s-conjunction for each tree's path from the root to a "1" leaf. E.g. this tree corresponds to the 3-DNF $\mathrm{p}_3 \vee (\neg \mathrm{p}_3 \wedge \neg \mathrm{p}_5)$. So

$$\mathcal{C}(s\text{-DT}) \subseteq \mathcal{C}(s\text{-DNF}) \tag{67}$$

s-DNF is efficently learnable online by a standard agent and thus also efficiently PAC-learnable. So the agent can efficiently PAC-learn s-DT using s-DNF. Thus s-DT is *efficiently* PAC-learnable.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learnability of s-Decision Trees (cont'd)

s-DT is also *properly* PAC-learnable by a s-DT-consistent agent according to Theorem (14) due to $\lg |s\text{-DT}| \leq poly(n_X)$ where $n_X = n$. Indeed, $|1\text{-DT}| = 2$ because there are exactly two options $\{0, 1\}$ for the single vertex (leaf) of it. So

$$\lg |1\text{-DT}| = \lg 2 = 1 \tag{68}$$

For $s > 1$, $|(s+1)\text{-DT}| = n|s\text{-DT}|^2$ (n options for the vertex and $|s\text{-DT}|$ options for each of the two subtrees). Take the logarithm of the equation:

$$\lg |(s+1)\text{-DT}| = \lg n + 2 \lg |s\text{-DT}| \tag{69}$$

(68) and (69) form a recursive prescription of a geometric series whose solution is $\lg |s\text{-DT}| = (2^s - 1)(1 + \lg n) + 1 \leq poly(n)$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learnability of s-Decision Trees (cont'd)

Finally, finding an s-tree consistent with a finite set of observations is an NP-complete problem. We omit the part of the proof showing this but refer to the analogical proof for s-term DNF following Theorem (16).

Thus the class s-DT is not efficiently properly PAC-Learnable, which completes the proof.

Note: similarly to (67), we also have

$$\underline{\mathcal{C}(s\text{-DT})} \subseteq \underline{\mathcal{C}(s\text{-CNF})} \tag{70}$$

Given an s-DT, one creates a clause for each path from root to a "0" leaf, e.g. this tree corresponds to the single-clause 3-CNF $\mathrm{p}_3 \vee \neg \mathrm{p}_5$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

s-Decision Lists

Example:

c	y
$\mathrm{p}_1 \wedge \neg \mathrm{p}_3$	0
p_2	1
$\neg \mathrm{p}_1$	1
$\emptyset$	0

2-Decision list

An **s-Decision list** on $X = \{0,1\}^n$ is a list of pairs (c, y) where c is an s-conjunction using variables from $\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_n$ and $y \in Y$.

The last conjunction in the list is empty and the corresponding y is called the *default class*.

It classifies an $x \in X$ into class y_i where (c_i, y_i) is the first pair in the list such that $x \models c_i$.

For example, $x = (1,1,1)$ is classified into 1 by the decision list on the left.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learnability of s-Decision Lists

Theorem 18

The class s-Decision lists is efficiently properly PAC-learnable from $X = \{0,1\}^n$.

We will present an s-DL-consistent algorithm known as the *covering algorithm* for efficient finding of an s-DL hypothesis h_{k+1} consistent with $x_{\leq k}$.

Let $T_{k+1} = \{(x_1, \overline{y}_1), (x_2, \overline{y}_2), \ldots, (x_k, \overline{y}_k)\}$ where $\overline{y}_i$ $(1 \leq i \leq k)$ is the true class of x_i. T_{k+1} is called a **training set** (at time $k+1$).

Note that the agent knows all elements of T_{k+1} because it has seen all of the x_i and the $\overline{y}_i$ can be determined as $\overline{y}_i = |y_i + r_{i+1}|$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Finding a Consistent s-Decision List

Require: training set T

$L := []$ (empty list)
while $T \neq \emptyset$ **do**
    $c =$ any s-conjunction true for some positive and no negative example in T, *or* some negative and no positive example in T (*respectively*)
    Remove samples covered by c: $T := T \setminus \{(x, y) \in T : x \models c\}$
    if $T = \emptyset$ **then**
        append $(\emptyset, 1)$ or $(\emptyset, 0)$ (*respectively*) to L.
    else
        append $(c, 1)$ or $(c, 0)$ (*respectively*) to L
    end if
end while

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learnability of s-Decision Lists (cont'd)

$$|s\text{-DL}| = 3^{|s\text{-conjunctions}|}!$$

because each s-conjunction can be absent from the list, present with $y = 0$ or present with $y = 1$ (hence the base 3), and they can be arranged in an arbitrary order (hence the factorial).

We know that $|s\text{-conjunctions}| \leq poly(n)$. So we have

$$\lg |s\text{-DL}| < poly(n)$$

So by Theorem (14), the s-DL-consistent covering algorithm PAC-learns s-DL. Since it is efficient and the output is an s-DL, it does so efficiently and properly, which finishes the proof.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

s-Decision Lists (cont'd)

Every s-DNF has an equivalent s-DL constructed as follows

- for each s-conjunction c from the s-DNF, add $(c, 1)$ to the s-DL
- add $(\emptyset, 0)$ to the s-DL

so

$$\mathcal{C}(s\text{-DNF}) \subset \mathcal{C}(s\text{-DL})$$

s-DL is closed under negation, i.e., for any $h \in s$-DL, also $\neg h \in s$-DL (just flip the zeros and ones for all the y_i in h). Each s-CNF is the negation of some s-DNF. Therefore also

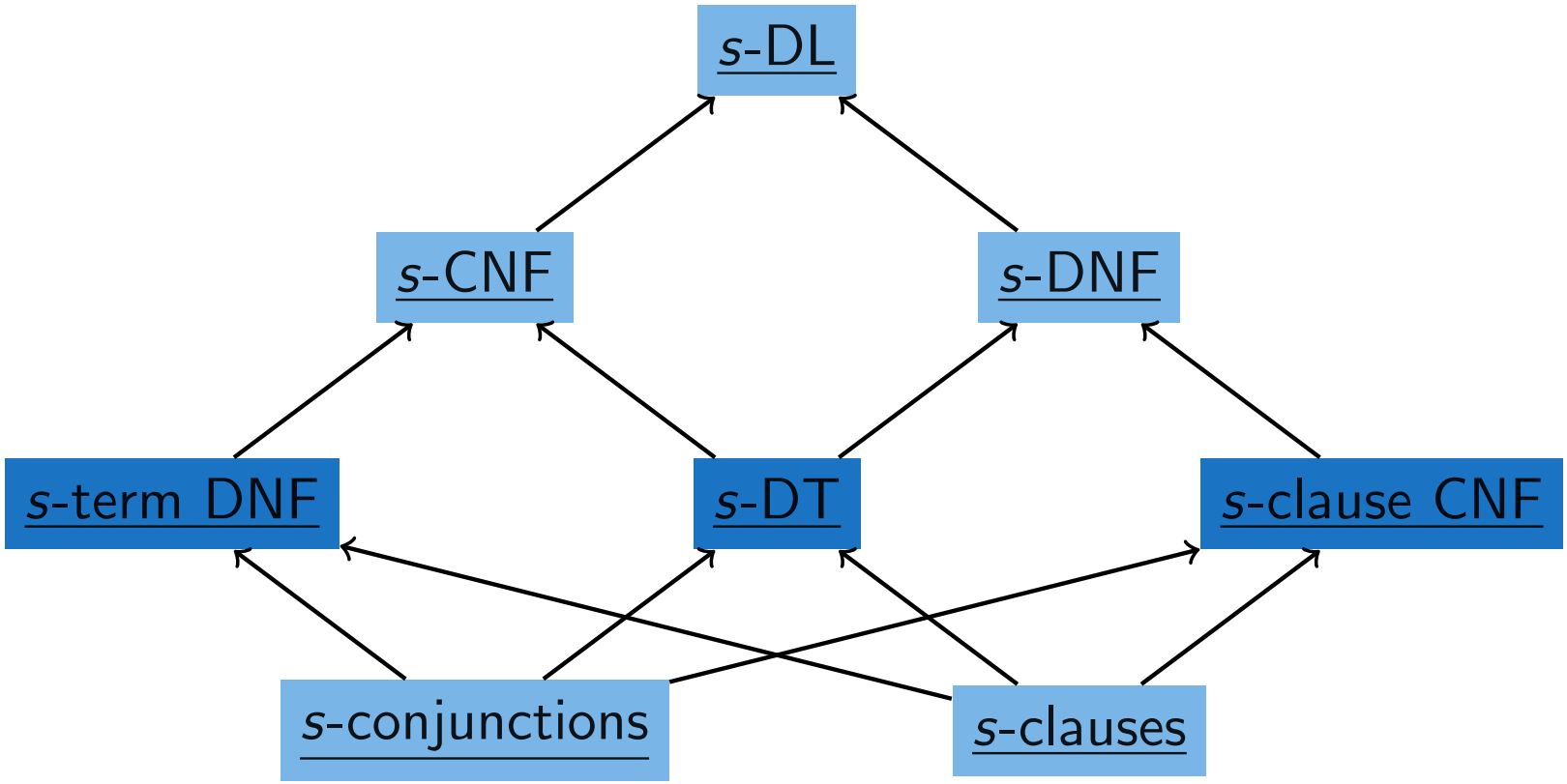
$$\mathcal{C}(s\text{-CNF}) \subset \mathcal{C}(s\text{-DL})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Hierarchy of Size-Bounded Propositional Classes

efficiently properly PAC-learnable

efficiently **or** properly PAC-learnable



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Inconsistent Learning

Consistent learning may not be possible when (64) does not hold or when rewards r_{k+1} are not deterministic as in (18) but depend only *probabilistically* on x_k and y_{k+1}. *The latter case corresponds to learning from "noisy data."*

Define the **training error** $\widehat{\mathrm{err}}(h_{k+1})$ $(k \in \mathbb{N})$ of hypothesis h_{k+1} as

$$\widehat{\mathrm{err}}(h_{k+1}) = \frac{1}{k} \sum_{i=1}^{k} |h_{k+1}(x_i) - \overline{y}_i| \tag{71}$$

where $\overline{y}_i$ is the true class of x_i. So $\widehat{\mathrm{err}}(h_{k+1})$ is the proportion of observations from $x_{\leq k}$ that h_{k+1} is not consistent with.

Note that $\widehat{\mathrm{err}}(h_{k+1})$ is in general not equal to $\frac{1}{k}\sum_{i=1}^{k}|r_i|$ since actions y_i, $1 \leq i \leq k$ were decided by hypotheses other than h_{k+1}.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Inconsistent Learning (cont'd)

The following lemma a direct consequence of the well-known Hoeffding inequality.

Lemma 7

Let $\{ z_1, z_2, \ldots, z_m \}$ *be a set of i.i.d. samples from* $P(z)$ *on* $\{ 0, 1 \}$*. Then the probability that* $\left| P(1) - \frac{1}{m} \sum_{i=1}^{m} z_i \right| > \epsilon$ *is at most* $2e^{-2\epsilon^2 m}$*.*

Theorem 19

Let $h_{k+1} \in \mathcal{H}$ $(\forall k \in \mathbb{N})$ *where* $\mathcal{H}$ *is a hypothesis class. With probability at least* $1 - \delta$

$$\left| \mathrm{err}(h_{k+1}) - \widehat{\mathrm{err}}(h_{k+1}) \right| \leq \sqrt{\frac{1}{2k} \ln \frac{2|\mathcal{H}|}{\delta}} \tag{72}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Inconsistent Learning (cont'd)

Proof of Theorem (72): by assumption, $x_1, x_2, \ldots, x_k$, are i.i.d. from (14), thus for a given $h_{k+1} \in \mathcal{H}$,

$$|h_{k+1}(x_1) - \overline{y}_1|, |h_{k+1}(x_1) - \overline{y}_2|, \ldots |h_{k+1}(x_k) - \overline{y}_k|$$

where $\overline{y}_i$ are the true classes of x_1 is an i.i.d. sample from a $P(.)$ on $\{0, 1\}$ where $P(1) = \text{err}(h_{k+1})$. Thus given (71) and Lemma (7), the probability that

$$|\text{err}(h_{k+1}) - \widehat{\text{err}}(h_{k+1})| > \epsilon$$

is at most $2e^{-2\epsilon^2 k}$. The probability that the above is true for *some* $h_{k+1} \in \mathcal{H}$ is thus at most $|\mathcal{H}|2e^{-2\epsilon^2 k}$. Setting $|\mathcal{H}|2e^{-2\epsilon^2 k} = \delta$ yields

$$\epsilon = \sqrt{\frac{1}{2k} \ln \frac{2|\mathcal{H}|}{\delta}}$$

which completes the proof.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Empirical Risk Minimization

When consistent learning is not possible, the best the agent can do is to minimize the *training error* (71), which is also called the **empirical risk**.

Notice the dilemma following from Theorem (19). A *larger* $\mathcal{H}$ will

- allow to achieve a smaller training error (we are choosing among more hypotheses)
- loosen the bound on the discrepancy (72) between error and the training error

Given a training set T, the dilemma is usually solved *empirically*, e.g., by cross-validating different $\mathcal{H}$ on T and then using the best $\mathcal{H}$ to learn from T.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Classification with Noise

Real-world concepts are often not "crisp" subsets of X as assumed by our current assumption.

Consider a "soft" alternative assuming that $x \in X$ belongs to class $y \in Y$ *with probability* $P(y \mid x)$. An appropriate replacement for the prescription (18) of the unit rewards ($R = \{0, 1\}$) is then ($k \in \mathbb{N}$)

$$r_1 = 0$$
$$r_{k+1} = \begin{cases} 0 \text{ with probability } P(y_{k+1} \mid x_k) \\ -1 \text{ otherwise} \end{cases} \tag{73}$$

Such rewards are probabilistic and we have already considered that. The agent can resort to empirical risk minimization using a class $\mathcal{H}$ of "crisp" hypotheses.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learning a Probability Distribution

But consider an alternative. Instead of learning a binary-policy hypothesis h_k from training set T_k ($k > 1$), learn from T_k an estimate p_k of the distribution $P(x, y)$ and use the policy

$$y_k = \arg\max_{y \in Y} p_k(y \mid x_{k-1}) \tag{74}$$

where $p_k(y \mid x_k) = p_k(x_k, y)/p_k(x_k)$ and $p_k(x_k) = \sum_{y \in Y} p_k(x_k, y)$.

This approach is appropriate for example when we do not know which class $\mathcal{H}$ contains a low-error hypothesis but we know the class of distributions (e.g., normal) containing P.

Being able to learn a distribution allows us to design agents for agent-environment interactions beyond classification.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learning a Probability Distribution (cont'd)

For example, let $V_1, V_2, \ldots, V_n$ be a set of discrete random variables distributed by some $P(V_1, V_2, \ldots, V_n)$. Let observations x_k be sampled from P but conveyed to the agent with *missing values* for some of the variables, i.e. only some of the n values are given to the agent.

Given x_k, the agent then predicts through y_{k+1} the most probable values of the rest of the variables according to its current model, i.e., ($k \in \mathbb{N}$)

$$y_{k+1} = \arg \max_{\{x_k^i\}_{i \in I}} p_{k+1}(\{x_k^i\}_{i \in I} \mid \{x_k^j\}_{j \in J}) \tag{75}$$

where J (I, respectively) contains the indexes of the observed (unobserved) variables.

Example: predicting occluded pixels given the surrounding pixels in images.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learning a Probability Distribution (cont'd)

Computing (74) or (75) is called **maximum aposteriori probability** (MAP) inference. (74) can be viewed as a special case of (75), in which the argument of maximization is fixed to the class variable.

Conceptually, (74) and (75) are the same problem, requiring the agent to learn a model of a joint distribution from samples from the distribution, possibly with missing values.

More precisely, the agent receives observations in the following way. Let $\mathcal{V}$ be a set of discrete random variables jointly distributed by P. Each observation *is sampled i.i.d from P* and then an arbitrary subset of its components is set to value '?' (indicating 'missing value').

From such observations the agent should learn an estimate p of P, i.e., for each $k > 1$, p_k is estimated from $x_{<k}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Dimensionality Problem

The task can be accomplished with a variant of the EM algorithm. At each $k+1$ ($k \in \mathbb{N}$), first set $p := p_k$, and then loop over two steps

1. Fill in missing values: Estimate the most probable values of unobserved components in all of $x_{\leq k}$ by MAP inference using p, yielding $\widehat{x}_{\leq k}$ with no missing values.
2. Re-estimate p by *relative frequencies*: for each value tuple v of $\mathcal{V}$ set $p(v) := m/k$ where m is the number of times v occurs in $\widehat{x}_{\leq k}$.

until p converges. Then set $p_{k+1} = p$.

The problem with the relative-frequency estimate is that when the dimension n grows linearly, to keep the accuracy of the estimate unchanged, the number of samples k must *grow exponentially*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Independence to the Rescue

The dimensionality problem vanishes in the special case where the n variables are pairwise independent, so P **factorizes** (i.e., is equal to a product of smaller factors) as

$$P(V_1, V_2, \ldots, V_n) = P_1(V_1)P_1(V_2)\ldots P_1(V_n) \tag{76}$$

Then instead of estimating P of dimension n, the agent estimates $P_1, P_2, \ldots P_n$, each of dimension 1. *This is trivial: e.g. $P(v) \approx$ the proportion of value v among all non-missing values $x^i_{\leq k}$.*

The problem with assumption (76) is that it is too strong making it irrelevant to real-life machine learning problems.

However, a weaker form of independence can be defined and exploited.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Conditional Independence

Let P be a joint probability distribution of a set of random variables $\mathcal{V}$. Let $A, B \in \mathcal{V}$ and $\mathcal{E} \subseteq \mathcal{V}$. We say that A **and** B **are conditionally independent given** $\mathcal{E}$ (under P) if $P(A, B \mid \mathcal{E}) = P(A \mid \mathcal{E})P(B \mid \mathcal{E})$. We denote this as $A \perp\!\!\!\perp_P B \mid \mathcal{E}$.

We will drop the set delimiters $\{\}$ in the conditional part when there is only one variable in the condition, i.e. will we write $A \perp\!\!\!\perp_P B \mid C$ rather than $A \perp\!\!\!\perp_P B \mid \{C\}$ to denote that A is conditionally independent of B given C.

It is obvious from the definition that $A \perp\!\!\!\perp_P B \mid \mathcal{E}$ implies

- $B \perp\!\!\!\perp_P A \mid \mathcal{E}$ (i.e., $\perp\!\!\!\perp_P$ is symmetric)
- $P(A|B, \mathcal{E}) = P(A|\mathcal{E})$ (hint: use the chain rule)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Conditional Independence

Three random variables:

T	outdoor temperature
I	ice-cream sales
H	heart-attack rate

I and H are not independent:

$$P(I, H) \neq P(I)P(H)$$

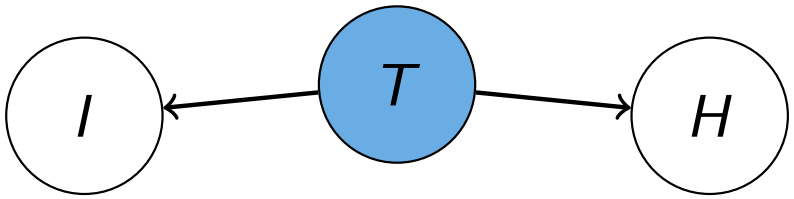
but they are *conditionally independent*:

$$P(I, H \mid T) = P(I \mid T)P(H \mid T)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

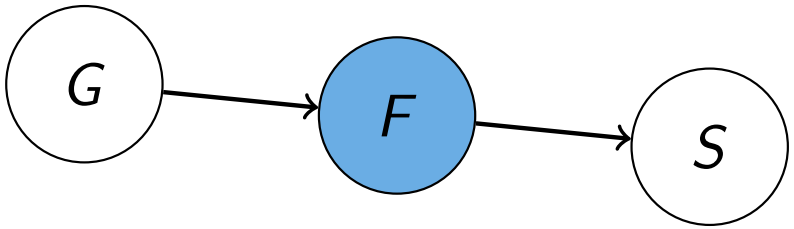
Conditional Independence in Cause-Effect Graphs

Heart attack rate and ice-cream sales independent if temperature known:



$$P(I, H \mid T) = P(I \mid T)P(H \mid T)$$

Son and grandfather's high IQ independent if same known for father:



$$P(S, G \mid F) = P(S \mid F)P(G \mid F)$$

In both cases: *any vertex is conditionally independent of all of its non-descendants given all its parents.*

This principle motivates the framework of *Bayesian networks*, which are a special case of probabilistic graphical models.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Networks

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayes Graph

Denote $\text{par}_G(V)$ the set of all parents of vertex V in an oriented graph G.

Bayes Graph

A **Bayes Graph** for a set $\mathcal{V}$ of random variables is an acyclic directed graph G with vertex set $\mathcal{V}$. A Bayes G is **correct** for a distribution P on $\mathcal{V}$ if $\forall V, V' \in \mathcal{V} : V \perp\!\!\!\perp_P V' \mid \text{par}_G(V)$ whenever V' is not a descendant of V in G.

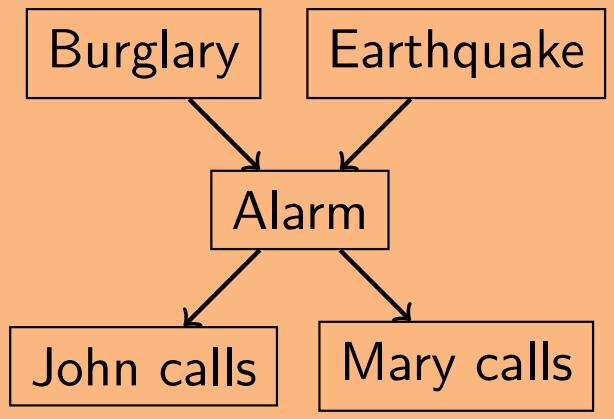
(Exercise problem)

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: a variable is conditionally independent of all its non-descendants if exactly all its parents are given.

There may by multiple BG's for one P.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Bayes Graph for Binary Variables (from AIMA)



From this Bayes graph, we can infer:

- $P(B, E) = P(B)P(E)$
- $P(J \mid X, A) = P(J \mid A)$ for all of $X \in \{ B, E, M \}$
- $P(M \mid X, A) = P(M \mid A)$ for all of $X \in \{ B, E, J \}$

By the chain rule of probability

$$P(B, E, A, M, J) = P(J \mid B, E, A, M)P(M \mid B, E, A)P(A \mid B, E)P(B, E)$$

but this simplifies using the inferred equalities:

$$P(B, E, A, J, M) = P(J \mid A)P(M \mid A)P(A \mid B, E)P(B)P(E)$$

(See more in a tutorial.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Probability Factorization by a Bayes Graph

The following theorem is a general statement of the factorization shown in the example. Its validity follows directly from the definition.

Theorem 20

Let G be a Bayes Graph correct for distribution P on variables $V_1, V_2, \ldots, V_n$. Then

$$P(V_1, V_2, \ldots, V_n) = \prod_{i=1}^{n} P\left(V_i \mid \mathrm{par}_G(V_i)\right) \tag{77}$$

Similarly to (76), the Theorem enables to express a high-dimensional distribution as a product of low-dimensional distributions provided that the variables V_i's have a low number of parents in G. *This assumption is more realistic than pairwise independence.*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Conditional Probability Table

Let us abbreviate $V = 1$ ($V = 0$, respectively) as v ($\neg v$) for any binary random variable V.

To store an estimate of $P(B, E, A, J, M)$ from the example, we need an array of size $2^5 = 32$ to store a probability for each value combination of the 5 variables. *More precisely, we need to store only 31 parameters as the 32 of them sum to 1.*

To specify its factorization $P(J \mid A)P(M \mid A)P(A \mid B, E)P(B)P(E)$, we need a *conditional probability table (CPT)* for each of the factors. E.g. for $P(A \mid B, E)$:

$P(a \mid B, E)$	E	B
0.001	0	0
0.940	0	1
0.290	1	0
0.950	1	1

(Exercise problem)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayes Network

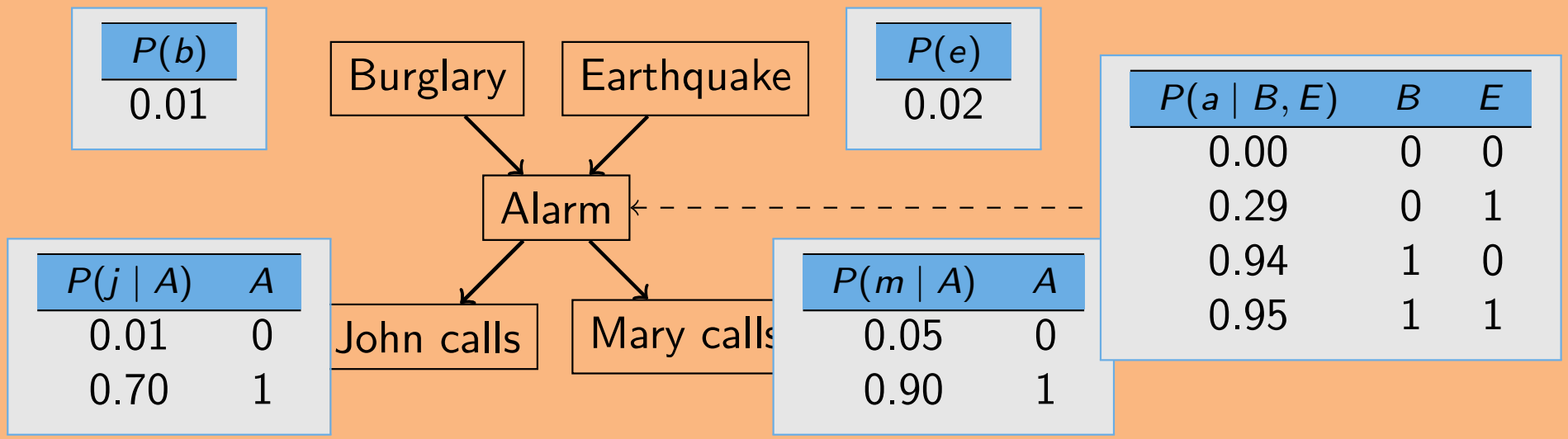
A **Bayes Network** for a distribution P on a set $\mathcal{V}$ of random variables consists of a Bayes graph G for $\mathcal{V}$, and a conditional probability table for each $V \in \mathcal{V}$ containing a number from $[0; 1]$ (i.e., a probability value) for each assignment of values to random variables $\mathrm{par}_G(V)$.

The probabilities in the CPT of any V specify $P(V \mid \mathrm{par}_G(V))$. So due to (77), a BN fully specifies P.

There are in general multiple BN's specifying the same P. More edges mean more parameters.

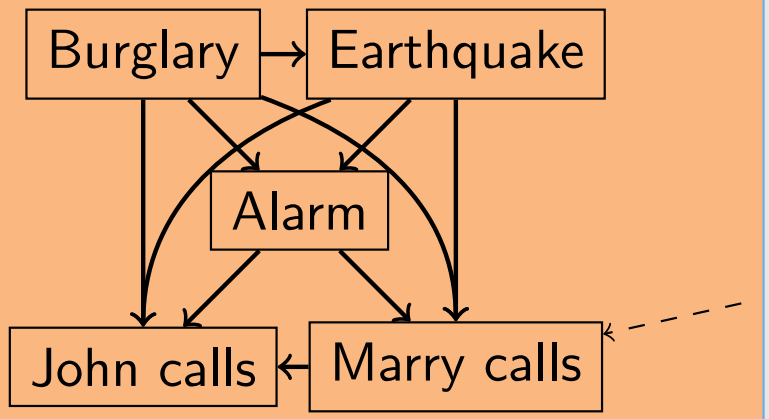
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayes Network Example



10 parameters, less than 31 in the full joint probability table.

A Different Graph for the Same Example



$P(m \mid B, E, A)$	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

$$P(B, E, A, M, J) = P(J \mid B, E, A, M)P(M \mid B, E, A)P(A \mid B, E)P(E \mid B)P(B)$$

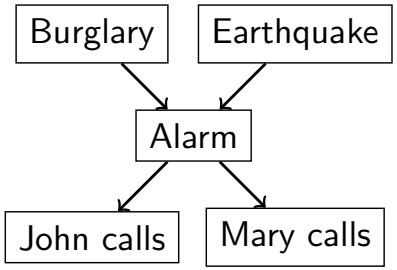
CPT's for the BN with this BG have $2^4 + 2^3 + 2^2 + 2 + 1 = 31$ parameters, same as the full joint probability table.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing Marginal Probabilities from a Bayes Network

So far we know how to compute the full joint distribution from CPT's:

$$P(B, E, A, J, M) = P(J \mid A)P(M \mid A)P(A \mid B, E)P(B)P(E)$$



A straightforward way to compute marginals, e.g. $P(A, J)$ is to *sum out* the remaining variables. Think why it is good below to push the sums as far right as possible! *(implemented in a tutorial)*

$$\begin{aligned} P(A, J) &= \sum_B \sum_E \sum_M P(J \mid A)P(M \mid A)P(A \mid B, E)P(B)P(E) \\ &= P(J \mid A) \sum_M P(M \mid A) \sum_B \sum_E P(A \mid B, E)P(B)P(E) \end{aligned}$$

B under a $\sum$ means summing over b and $\neg b$. Same for other variables.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing Conditional Probabilities from a Bayes Network

Conditional probabilities are just fractions of marginals, e.g.

$$P(A, J \mid B, E) = \frac{P(A, J, B, E)}{P(B, E)}$$

(exercise problem)

Instead of calculating the denominator, we can evaluate the numerator for all assignments to A, J and normalize, since $\sum_A \sum_J P(A, J \mid B, E) = 1$.

$$\alpha \left[P(\neg a, \neg j, B, E) + P(\neg a, j, B, E) + P(a, \neg j, B, E) + P(a, j, B, E) \right] = 1$$

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 $\langle \neg a, j \rangle$

$$P(\neg a, j \mid B, E) = \alpha \cdot P(\neg a, j, B, E)$$

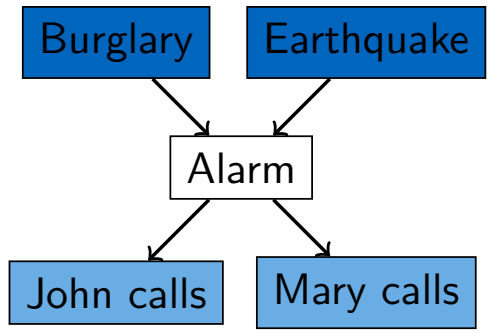
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Evidence and Query Variables

In BN terminology, the variables whose joint conditional probability is computed are called *query* variables; those in the condition part are *evidence* variables.

Example query: *probability that neither John nor Mary will call during a burglary and no earthquake*:

$$P(\underbrace{\neg j, \neg m}_{\text{query}} \mid \underbrace{b, \neg e}_{\text{evidence}})$$

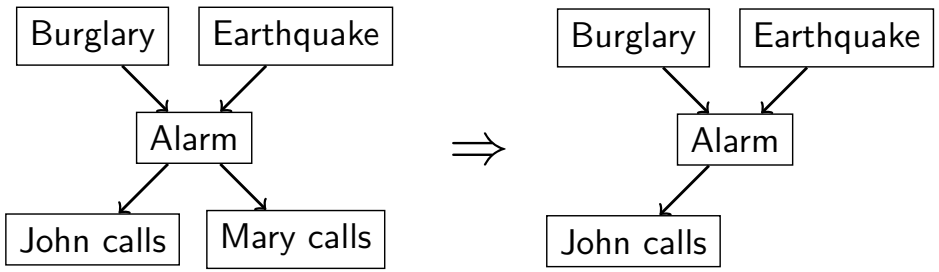


In (75), the query variables include *all* unobserved variables and evidence includes *all* observed variables. But BN's enable more general queries: query and evidence can be arbitrary subsets of all variables.

Removing Irrelevant Variables

Consider
$P(J \mid b) = \alpha P(b) \sum_E P(E) \sum_A P(A \mid b, E) P(J, A) \sum_M P(M \mid A)$

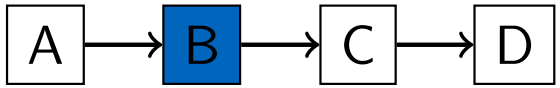
$\sum_M P(M \mid A) = 1$ so it can be left out, i.e. remove the corresponding vertex from the BN.



In general, *any vertex that is not an ancestor to a query variable or evidence variable of a query can be removed from the graph when computing the query*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

d-Separation

Consider the Bayes Graph 

correct for some $P(A, B, C, D)$.

Are A and D independent under P if B is observed? I.e., does G imply

$$A \perp\!\!\!\perp_P D \mid B \ ?$$

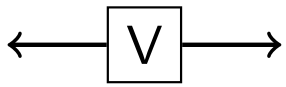
Yes, but this does not immediately follow from the definition because $\mathrm{par}_G(D) = \{ C \}$ is not observed.

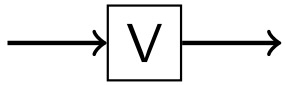
The *d-separation* criterion serves to decide all cases of independence implied by a Bayes Graph.

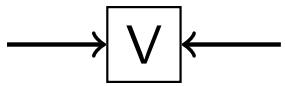
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

d-Separation (cont'd)

Given some evidence $\mathcal{E}$, we say that variables A and B are **d-separated** in the Bayes Graph G by $\mathcal{E}$ if on every undirected path between A and B in G, there is

- either a vertex $V \in \mathcal{E}$ such that the (directed) edges adjacent to V on the path are
  - either *diverging*, i.e., 

  - or *linear*, i.e., 

.
- or a vertex $V \notin \mathcal{E}$ such that $D \notin \mathcal{E}$ *also for all descendants* D of V in G, and the edges adjacent to V on the path are
  - *converging*, i.e., 

.

d-Separation (cont'd)

Theorem 21

Let G be a Bayes graph for a distribution P of a set $\mathcal{V}$ of random vars. Let further $A, B \in \mathcal{V}$ and $\mathcal{E} \subseteq \mathcal{V}$. If A and B are d-separated in G by $\mathcal{E}$ then $A \perp\!\!\!\perp_P B \mid \mathcal{E}$.

Proof (not trivial) can be found in Verma & Pearl, 1998.

(exercise problem)

D-separation can be checked by an efficient algorithm that does not enumerate all paths between the inspected pair of nodes.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Checking d-Separation

To determine if A and B are d-separated in G by $\mathcal{E}$,

1. Extract from G the **ancestral graph** G_{anc} by keeping only vertices in $\{A, B\} \cup \mathcal{E}$ and all their ancestors (edges between kept vertices are kept.)
2. **Moralize** G_{anc} by putting an undirected edge between (i.e., "marrying") each pair of parents of any vertex; then replace in G_{anc} all directed edges by an undirected edge.
3. Delete from G_{anc} all vertices from $\mathcal{E}$ along with their edges.

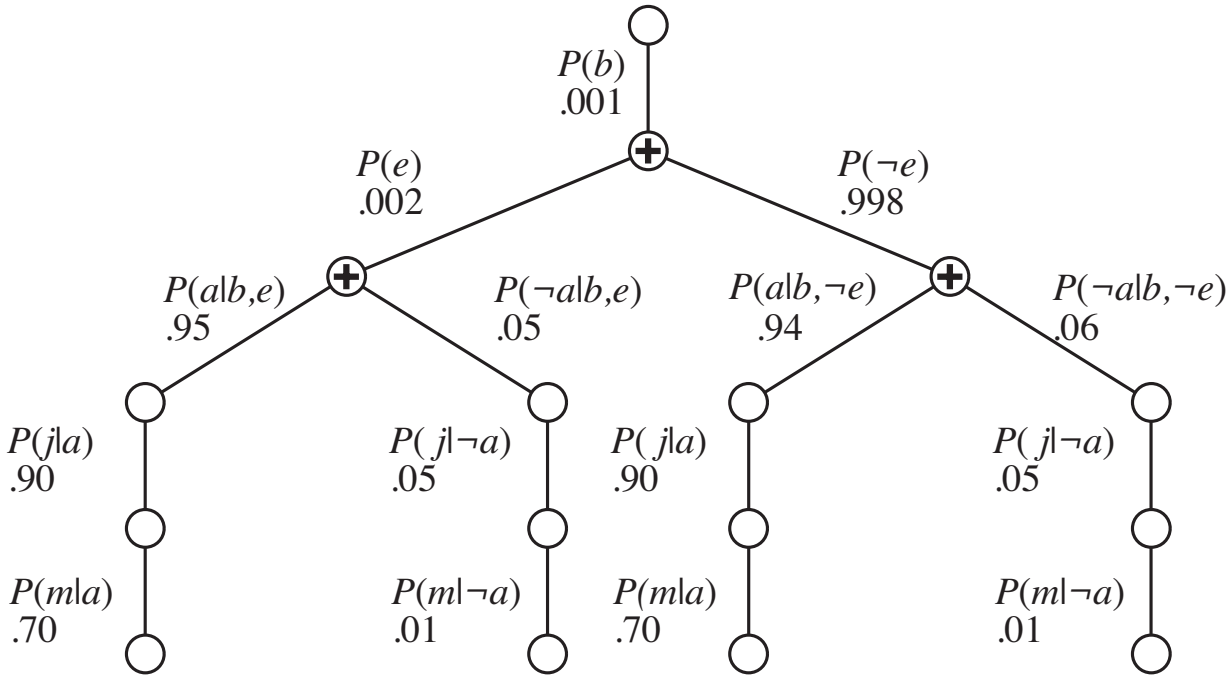
A and B are d-separated in G by $\mathcal{E}$ iff A and B are not connected in the resulting graph.

(Implemented in a tutorial).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Redundancy in Probability Computation

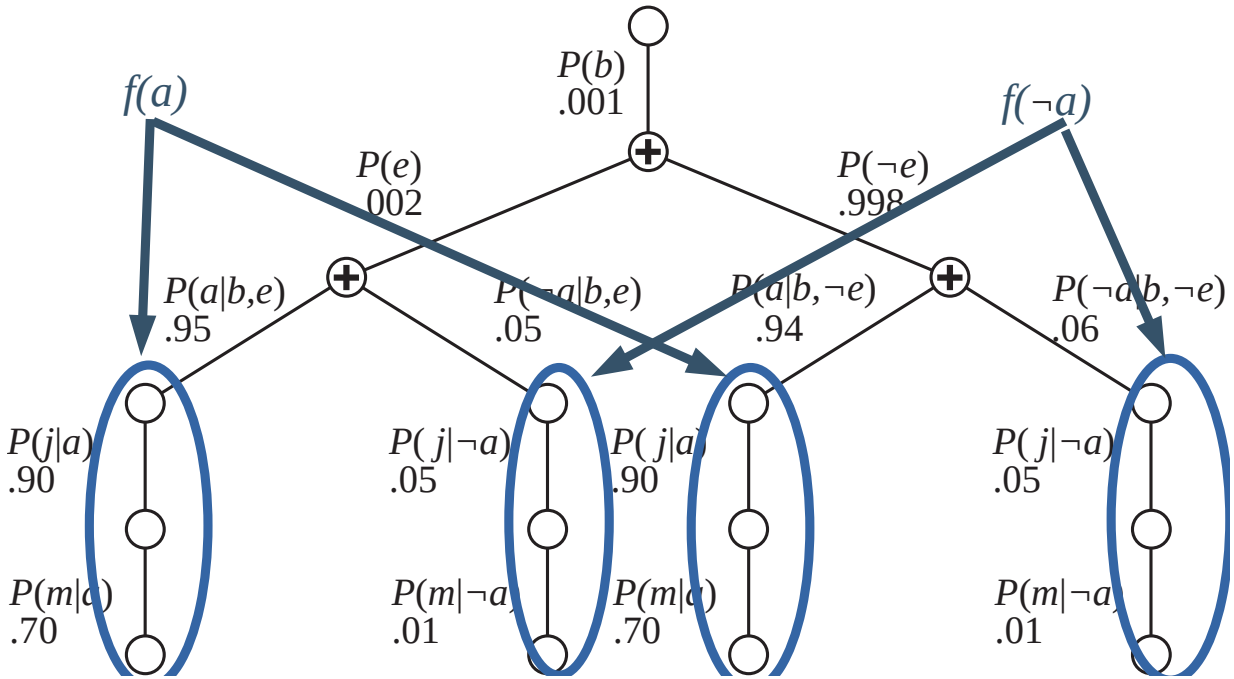
Consider $P(b \mid j, m) = \alpha P(b) \sum_E P(E) \sum_A P(A \mid b, E) P(j \mid A) P(m \mid A)$.
Observe the repeated sub-summing (branches) in its computation:



Factors

This redundancy is removed by using **factors**, which are arrays storing reusable computation results.

$$f(A) = \begin{bmatrix} f(\neg a) \\ f(a) \end{bmatrix} = \begin{bmatrix} P(j \mid \neg a)P(m \mid \neg a) \\ P(j \mid a)P(m \mid a) \end{bmatrix}$$ is an example factor.



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing Probabilities with Factors

Each query generates a set **initial factors**, one for each (conditional) probability in its factorized expression. So $P(B \mid j, m)$ generates 5 initial factors:

$$P(B \mid j, m) = \alpha \underbrace{P(B)}_{f_1(B)} \sum_E \underbrace{P(E)}_{f_2(E)} \sum_A \underbrace{P(A \mid B, E)}_{f_3(A,B,E)} \underbrace{P(j \mid A)}_{f_4(A)} \underbrace{P(m \mid A)}_{f_5(A)} \quad (78)$$

They have a dimension (argument) for each *uninstantiated* (upper-case) variable and the values are given by the corresponding CPT entries. E.g.,

$$f_4(A) = \begin{bmatrix} P(j \mid \neg a) \\ P(j \mid a) \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.70 \end{bmatrix} \quad (79)$$

(Refer to the CPT shown here.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing Probabilities with Factors (cont'd)

We use the initial factors to express (78) as

$$P(B \mid j, m) = \alpha f_1(B) \times \sum_E f_2(E) \times \sum_A f_3(A, B, E) \times f_4(A) \times f_5(A) \quad (80)$$

where

- $\times$ is **factor multiplication** producing a factor from two factors (different from matrix multiplication!)
- $\sum_V$ is **variable elimination** producing a factor without the V variable from a factor with the V variable

Using these two operations, we evaluate (80) from right to left.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Factor Multiplication

Factors are multiplied **point-wise** (similar to a database *join*). Example:

$$f_3(A,B,E) \times f_4(A) =$$

$$\begin{bmatrix} 1.00 & \neg a,\neg b,\neg e \\ 0.71 & \neg a,\neg b,e \\ 0.06 & \neg a,b,\neg e \\ 0.05 & \neg a,b,e \\ 0.00 & a,\neg b,\neg e \\ 0.29 & a,\neg b,e \\ 0.94 & a,b,\neg e \\ 0.95 & a,b,e \end{bmatrix} \times \begin{bmatrix} 0.01 & \neg a \\ 0.70 & a \end{bmatrix} = \begin{bmatrix} 1.00 \cdot 0.01 \\ 0.71 \cdot 0.01 \\ 0.06 \cdot 0.01 \\ 0.05 \cdot 0.01 \\ 0.00 \cdot 0.70 \\ 0.29 \cdot 0.70 \\ 0.94 \cdot 0.70 \\ 0.95 \cdot 0.70 \end{bmatrix}$$

Blue text indicates the array indexes.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Factor Multiplication (cont'd)

For (80), we proceed as follows

$$f_6(A) = f_4(A) \times f_5(A) = \begin{bmatrix} 0.01 & \neg a \\ 0.70 & a \end{bmatrix} \times \begin{bmatrix} 0.05 & \neg a \\ 0.90 & a \end{bmatrix} = \begin{bmatrix} 0.0005 & \neg a \\ 0.63 & a \end{bmatrix}$$

(So f_6 is the example factor f from here.)

$$f_7(A,B,E) = f_3(A,B,E) \times f_6(A) = \begin{bmatrix} 1.00 & \neg a,\neg b,\neg e \\ 0.71 & \neg a,\neg b,e \\ 0.06 & \neg a,b,\neg e \\ 0.05 & \neg a,b,e \\ 0.00 & a,\neg b,\neg e \\ 0.29 & a,\neg b,e \\ 0.94 & a,b,\neg e \\ 0.95 & a,b,e \end{bmatrix} \times \begin{bmatrix} 0.0005 & \neg a \\ 0.63 & a \end{bmatrix}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Variable Elimination

Now we have reduced (80) into

$$P(B \mid j, m) = \alpha f_1(B) \times \sum_E f_2(E) \times \sum_A f_7(A, B, E)$$

where $\sum_A f_7(A, B, E)$ produces a new factor $f_8(B, E)$ defined as

$$f_8(B, E) = \sum_A f_7(A, B, E) = f_7(\neg a, B, E) + f_7(a, B, E) \tag{81}$$

In general the operation $\sum_V f(V, \ldots)$ yielding a new factor without V is called **variable elimination**.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing Probabilities with Factors (cont'd)

Continue interleaving factor multiplication with variable elimination:

- $f_9(B, E) = f_2(E) \times f_8(B, E)$
- $f_{10}(B) = f_9(B, e) + f_9(B, \neg e)$
- $f_{11}(B) = f_1(B) \times f_{10}(B)$

Finally

$$P(B \mid j, m) = \alpha f_{11}(B) \tag{82}$$

where $\alpha = 1/(f_{11}(\neg b) + f_{11}(b))$.

Note that even with factors, complexity of inference in Bayes Networks is obviously exponential: a factor with n variables has 2^n values in it.

(Check out the first part of this exercise problem)

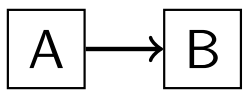
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

MAP inference

We want to compute (75) without computing the probabilities of all combinations of values of the unobserved variables. The joint MAP state need not consist of the values maximizing their marginal probabilities!

Consider e.g.

$P(a)$
0.6



$P(b \mid A)$	A
0.90	0
0.55	1

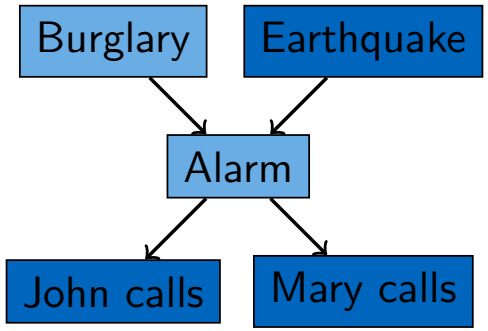
$$\arg\max_A P(A) = 1$$

but

$$\arg\max_{A,B} P(A, B) = (0, 1)$$

MAP Inference (cont'd)

Principle similar to basic probability inference. Example:



First compute the maximum probability:

$$\max_{A,B} P(A, B \mid \neg e, \neg j, \neg m) =$$

$$\alpha \max_{A,B} P(B)P(\neg e)P(A \mid B, \neg e)P(\neg j \mid A)P(\neg m \mid A) =$$

$$\beta \max_{B} P(B) \max_{A} P(A \mid B, \neg e)P(\neg j \mid A)P(\neg m \mid A)$$

where $\beta = \alpha P(\neg e)$ need *not* be evaluated to compute the arg max.

MAP Inference with Factors

Reformulate using factors:

$$\max_{A,B} P(A, B \mid \neg e, \neg j, \neg m) = \beta \max_{B} f_1(B) \times \max_{A} f_2(A, B) \times f_3(A) \times f_4(A)$$
$$= \beta \max_{B} f_1(B) \times \max_{A} f_7(A, B) = \beta \max_{B} f_8(B) \tag{83}$$

where $\max_V f(V, \ldots)$ produces a new factor (without the V dimension) containing the maximal values over the V dimension. This is called **maximizing out** V.

The arguments $(a_{\max}, b_{\max})$ maximizing (83) are determined as

- $b_{\max} = \arg\max_B f_8(B)$
- $a_{\max} = \arg\max_A f_7(A, b_{\max})$

Note that the order of the two steps cannot be switched.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

MAP Involving Variable Elimination

Consider a case where query variables are not the full complement to evidence variables as in here.

Say we want to find the most probable state of B and E when both J and M are true and A is not observed. Now we have both maximization and summation:

$$\max_{B,E} P(B, E \mid j, m) = \alpha \max_{B,E} \sum_{A} P(B)P(E)P(A \mid B, E)P(j \mid A)P(m \mid A) \tag{84}$$

We still can push operators before the first occurrence of their arguments *but* cannot swap the order of a max operator and a $\sum$ operator since

$$\sum_{Y} \max_{X} P(X, Y) \neq \max_{X} \sum_{Y} P(X, Y)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

MAP Involving Variable Elimination (cont'd)

So (84) rewrites to

$$\alpha \max_B f_1(B) \times \max_E f_2(E) \times \sum_A f_3(A, B, E) \times f_4(A) \times f_5(A) =$$

The scope of any max *is everything to the right of it!* Now multiply factors after $\sum_A$ and then eliminate A

$$\alpha \max_B f_1(B) \times \max_E f_2(E) \times \sum_A f_6(A, B, E) = \alpha \max_B f_1(B) \times \max_E f_2(E) \times f_7(B, E)$$

Multiply f_2 with f_7 and then maximize out E:

$$= \alpha \max_B f_1(B) \times \max_E f_8(B, E) = \max_B f_9(B)$$

Finally, $b_{\max} = \arg\max_B f_9(B)$, $e_{\max} = \arg\max_E f_8(b_{\max}, E)$.

(Check out the second part of this exercise problem)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learning a Bayes Network

We now assume that p_{k+1} is a Bayes Network with Bayes graph G and a set $\mathbf{w}$ of parameters instantiating the conditional probability tables.

A usual way to learn $p_{k+1} = (G, \mathbf{w})$ from observations $x_{\leq k}$ is to maximize the likelihood, i.e. choose a model maximizing the probability of $x_{\leq k}$:

$$p_{k+1} = \arg \max_{p=(G,\mathbf{w})} p(x_{\leq k})$$

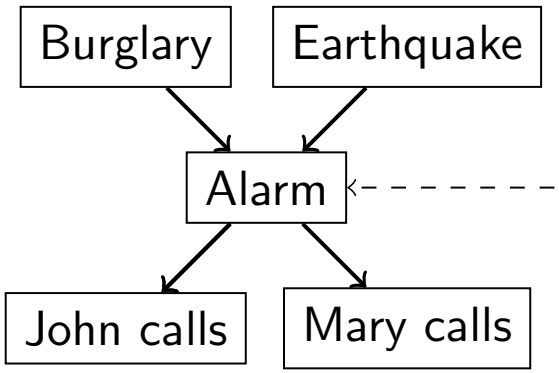
We will only consider the scenario where G is given (i.e. is agent's background knowledge) and the agent only learns $\mathbf{w}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Estimation of CPT Parameters of a Bayes Network

CPT parameters can be estimated using the EM algorithm as in here, except in Step 2 (M step), we estimate the conditional probabilities of the CPT by maximizing the likelihood.

Example of the M step at $k = 7$ after missing data values have been replaced by their expectations (E step) according to the current model:



$P(a \mid B, E)$	B	E
	0	0
$p(a \mid \neg b, e)$	0	1
	1	0
	1	1

	B	E	A	J	M
x_1	0	1	1	0	0
x_2	1	1	1	1	1
x_3	0	1	1	1	0
x_4	0	1	0	0	0
x_5	1	0	0	0	0
x_6	1	0	1	1	1

Here, $p_7(a \mid \neg b, e) := 2/3$ (relative frequency maximizes likelihood.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Fast Estimation of CPT Parameters

The previous approach has high complexity due to:

1. iterating the EM steps until convergence
2. performing MAP inference for every missing value in each E iteration
3. the need to store all of $x_{\leq k}$ in agent's memory

(1) and (2) can be avoided by estimating the CPT probabilities only from the subset of $x_{\leq k}$ where the needed components are not missing.

p_k will then converge slower to P in terms of k but may converge faster in terms of runtime.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Fast Estimation of CPT Parameters (cont'd)

For example, with observations

	B	E	A	J	M
x_1	0	1	0	0	0
x_2	?	1	1	1	1
x_3	0	1	1	1	0
x_4	0	1	?	?	0
x_5	1	0	0	0	?
x_6	0	1	0	1	1

estimate $p_6(a \mid \neg b, e) := 1/3$.

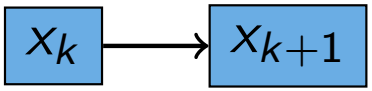
Finally, (3) is also prevented: rather than remembering all of $x_{\leq k}$, update the estimate by the cumulative moving average rule, only keeping the count K of observations used so far for the estimate. Here:

$$p_6(a \mid \neg b, e) = p_5(a \mid \neg b, e) + \frac{A_6 - p_5(a \mid \neg b, e)}{K+1} = \frac{1}{2} + \frac{0 - \frac{1}{2}}{3} = 1/3$$

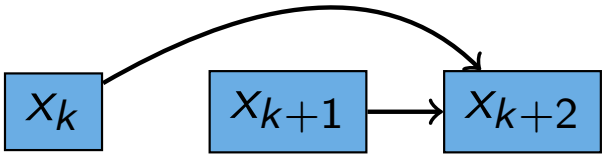
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Temporal Bayesian Networks

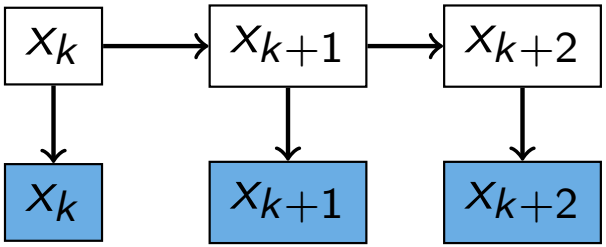
Bayesian networks generalize some well known temporal models.



Markov process (1^{st} order)



Markov process (2^{nd} order)

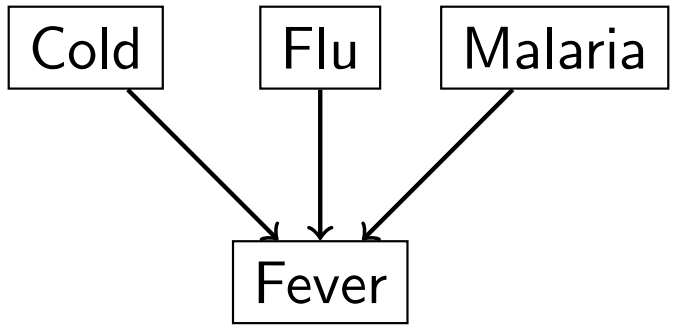


Markov hidden process (1^{st} order, 3 observations)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

BN Encoding Logical Rules

Observe that Bayesian networks can encode propositional logic formulas. Example:



$P(\text{fever} \mid C, F, M)$	C	F	M
1	1	1	1
0	all other cases		

$$\text{fever} \leftarrow \text{cold} \wedge \text{flu} \wedge \text{malaria}$$

$P(\text{fever} \mid C, F, M)$	C	F	M
0	0	0	0
1	all other cases		

$$\text{fever} \leftarrow \text{cold} \vee \text{flu} \vee \text{malaria}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Statistical Relational Learning

Bayes networks extend propositional logic by enabling to express probabilistic dependencies.

First-order logic (FOL) extends propositional logic by enabling to express structural (relational) dependencies.

These aspects are combined in the formalism of *Bayes Logic Programs* studied in the field called Statistical Relational Learning (SRL).

In SRL, also other probabilistic graphical models (e.g., Markov networks) are endowed with FOL expressivness (Markov logic networks).

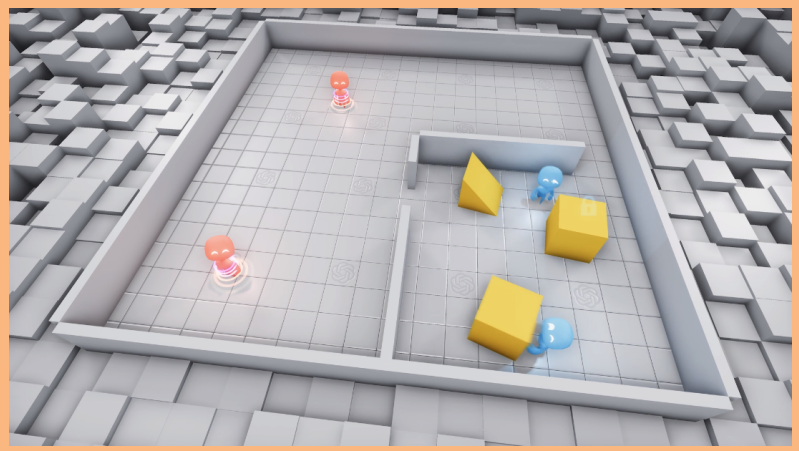
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Reinforcement Learning

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

OpenAI: Hide and Seek Game

Multiple learning agents split in two teams: hiders and seekers



go to web

- Observations: other agents and blocks
- Actions: move, shift blocks
- Rewards to hiders: -1 if any hider seen by a seeker, +1 if all hiders hidden. Seekers get opposite rewards.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Observations = States

How does the example fall into our framework? The *observation* set X is *finite*. Observations in RL are also called **states**.

The initial state x_1 is distributed by $P(x_1)$ independently of y_1 (so y_1 is irrelevant). For $k > 1$, states are distributed as

$$x_k \overset{c}{\sim} P(x_k \mid x_{k-1}, y_k) \tag{85}$$

which is a special case of (8). As the distribution is the same for all $k > 1$, we will also use the index-less form

$$P(x' \mid x, y) \tag{86}$$

where x' is the state immediately following state x.

Terminal States

A state $x \in X$ is **terminal** if

$$P(x' \mid x, y) = P(x')$$

Intuitively, when a terminal state x is reached, the environment is 'restarted', i.e. the next state x' is sampled independently of x, y from the same distribution as the initial state x_1.

An **episode** is the span of interaction

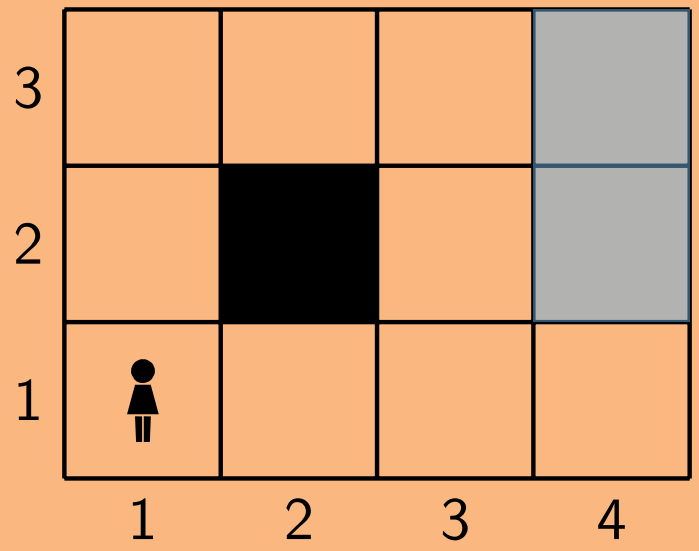
- starting with x_1 or a state immediately following a terminal state
- ending with the next terminal state.

Episodes model e.g. repeated games where terminal states may be 'game lost' or 'game won'.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Grid World - States

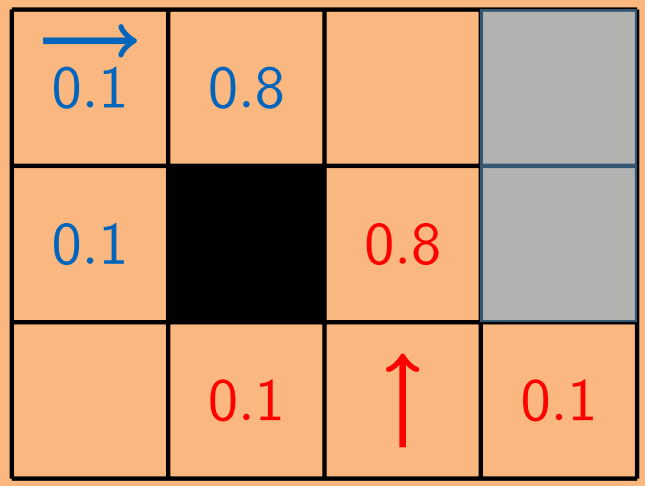
Consider a simpler example with a single agent:



X: 11 possible agent's positions on the grid.
$x_1 = (1, 1)$ with probability 1
$(4, 2), (4, 3)$ are *terminal*, always followed by state $(1, 1)$

Example: Grid World - Actions

Actions $Y = \{ \text{left}, \text{right}, \text{up}, \text{down} \}$



Uncertain effects: prescribed direction with 0.8 probability,
each of perpendicular directions with 0.1 probability

Bouncing: if new state out of grid, agent stays put

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

State Transitions under Policy

Due to (85), the probability of a sequence of states $x_1, x_2, \ldots$ given actions $y_1, y_2, \ldots$ is

$$P(x_1, x_2, \ldots \mid y_1, y_2, \ldots) = P(x_1)P(x_2 \mid x_1, y_2)P(x_3 \mid x_2, y_3)\ldots \tag{87}$$

Define for $k > 1$

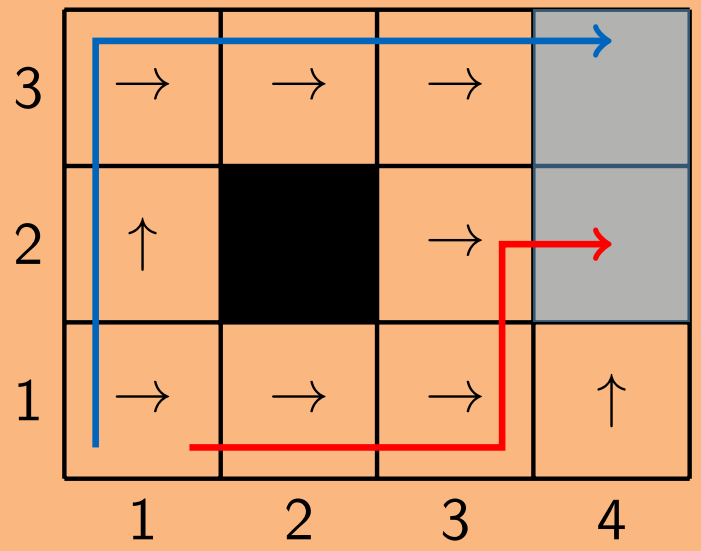
$$P^\pi(x_k \mid x_{k-1}) = P(x_k \mid x_{k-1}, \pi(x_{k-1}))$$

where π is an agent's policy. So (87) can be written as

$$P^\pi(x_1, x_2, \ldots) = P(x_1)P^\pi(x_2|x_1)P^\pi(x_3|x_2)\ldots \tag{88}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: State Transitions under Policy



- Arrows indicate policy π for all states except terminal (irrelevant)
- Probability of blue path and red path:

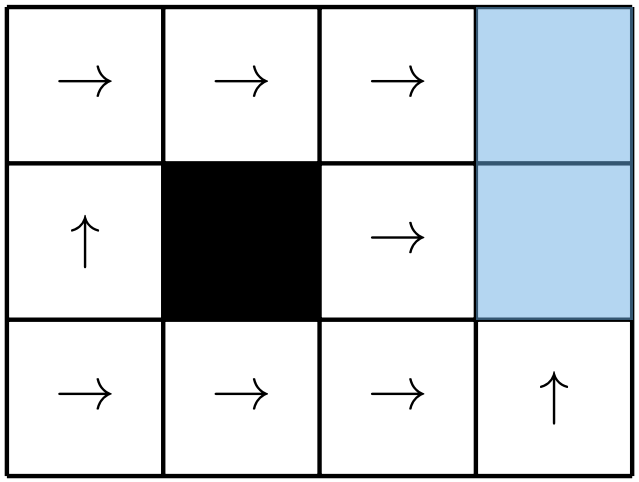
$$P((1,1)) \cdot P^{\pi}((1,2) \mid (1,1)) \cdot \ldots = 1 \cdot 0.1 \cdot 0.8 \cdot 0.8 \cdot 0.8 \cdot 0.8$$

$$P((1,1)) \cdot P^{\pi}((2,1) \mid (1,1)) \cdot \ldots = 1 \cdot 0.8 \cdot 0.8 \cdot 0.1 \cdot 0.8$$

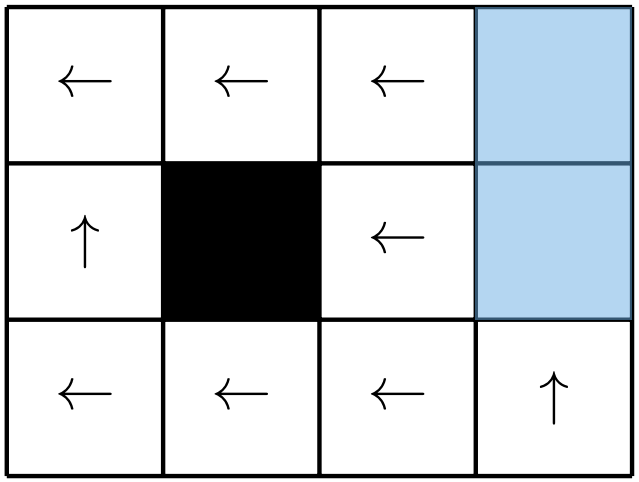
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Proper Policy

A **proper policy** is a policy such that from each state $x \in X$ there is a sequence of state transitions that has non-zero probability and that ends a terminal state.



proper



not proper

Rewards

Rewards are instant, i.e., as in (12) but since x_k (85) depends on the same conditions x_{k-1}, y_k, we can assume instead a functional dependence:

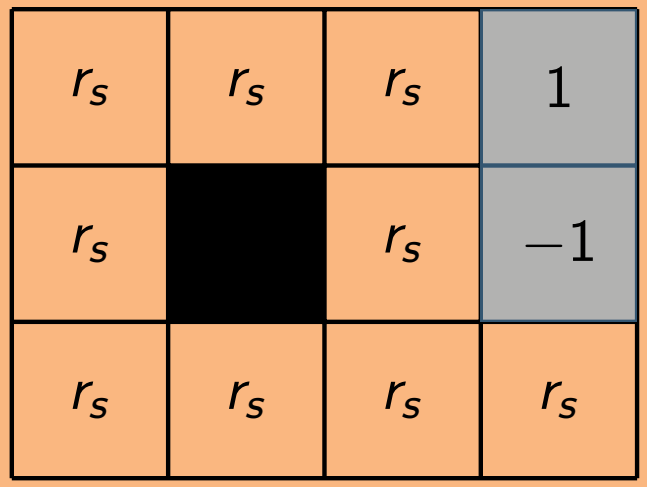
$$r_k = r(x_k) \tag{89}$$

(This is because r_k can be formally made a part of x_k and 'extracted' by the function above.)

Rewards are thus a *function* of states.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Grid World - Rewards



Terminal states : 'win' (1) and 'lose' (-1) rewards

Non-terminal states : a small punishment r_s (e.g., $r_s = -0.05$) for every step taken

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

State Utility under a Proper Policy

For a $\gamma \in [0; 1]$, define the **utility $U^\pi(x)$ of state $x \in X$ under a proper policy π** as

$$U^\pi(x) = \begin{cases} r(x) \text{ if } x \text{ is terminal} \\ r(x) + \gamma \sum_{x' \in X} P^\pi(x' \mid x) U^\pi(x') \text{ otherwise} \end{cases} \tag{90}$$

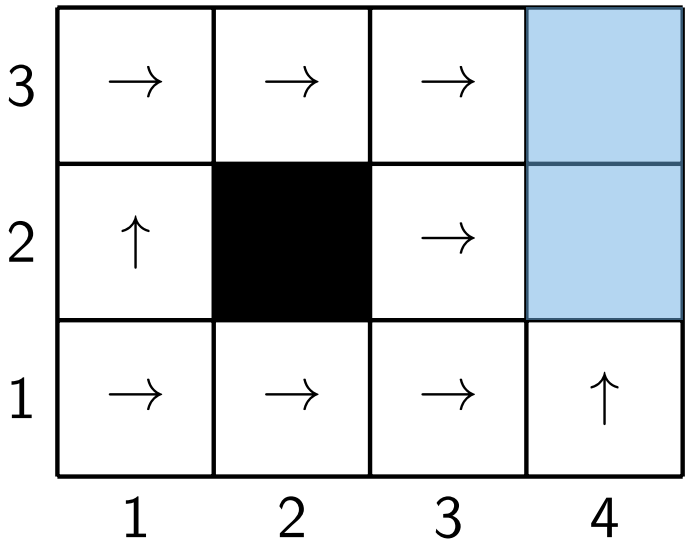
$U^\pi(x)$ is thus recursively defined as the expected sum of γ-discounted rewards in an episode starting in state x. Since an episode has a finite length, $U^\pi(x)$ converges even with $\gamma = 1$ (i.e., if γ dropped).

This is similar to U^π (11) but U^π is the expected sum of γ-discounted rewards in the entire interaction consisting of an infinite succession of episodes.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

State Utility under a Proper Policy (cont'd)

Given π, $U^\pi(x)$ for all $x \in X$ can be calculated by solving $|X|$ linear equations. Example, abbreviating $U^\pi((i,j))$ as u_{ij}:



$$u_{43} = 1 \text{ (terminal)}$$

$$u_{42} = -1 \text{ (terminal)}$$

$$u_{41} = r_s + \gamma(0.8 \cdot u_{42} + 0.1 \cdot u_{31} + 0.1 \cdot u_{41})$$

etc.

Optimal Policy and State Utility

Theorem 22

For any $x, x' \in X$, $U^\pi(x)$ and $U^\pi(x')$ are maximized by the same policies:

$$\overline{\pi} = \arg\max_\pi U^\pi(x) = \arg\max_\pi U^\pi(x')$$

and if $\gamma < 1$, these policies also maximize U^π:

$$\overline{\pi} = \arg\max_\pi U^\pi$$

Intuitive proof: the best action to take in any state x evidently does not depend on how x was reached - whether going from x or from x'. Thus $\arg\max_\pi U^\pi(x) = \arg\max_\pi U^\pi(x')$. Since $\overline{\pi}$ is optimal for each starting state, i.e., for each episode, the discounted sum U^π over all episodes is also maximized by $\overline{\pi}$. Hence the last equality.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

State Utility and Policy Iteration

$\overline{\pi}$ is called the **optimal policy** and the **utility of state** $x \in X$ is defined as the utility of x under the optimal policy, i.e.

$$U(x) = U^{\overline{\pi}}(x) \tag{91}$$

An optimal policy can be computed by **policy iteration**, which is similar in spirit to the EM algorithm. Start with a random initial policy π and iterate

1. $U(x) := U^{\pi}(x)$
2. compute a new policy π such that

$$\pi(x) := \arg\max_{y} \sum_{x' \in X} P(x' \mid x, y) U(x') \tag{92}$$

until the policy does not change.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Value Iteration

As an alternative to policy iteration, the following method can be used to calculate $U(x)$ and $\overline{\pi}$:

1. Calculate $U(x)$, $x \in X$ as a solution to the set of Bellman equations

$$U(x) = r(x) + \gamma \max_{y \in Y} \sum_{x' \in X} P(x' \mid x, y) U(x') \tag{93}$$

   one for each non-terminal $x \in X$ (for a terminal x, $U(x) = r(x)$). This is called **value iteration**.
2. Calculate $\overline{\pi}$ by (92).

Unlike in policy iteration, the two steps are not *iterated*. However, (93) is not linear so an iterative algorithm is needed to find its solution.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Grid World - Optimal Policy and Utilities

$-.04$	$-.04$	$-.04$	1
$-.04$		$-.04$	-1
$-.04$	$-.04$	$-.04$	$-.04$

rewards

$\rightarrow$	$\rightarrow$	$\rightarrow$	
$\uparrow$		$\uparrow$	
$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

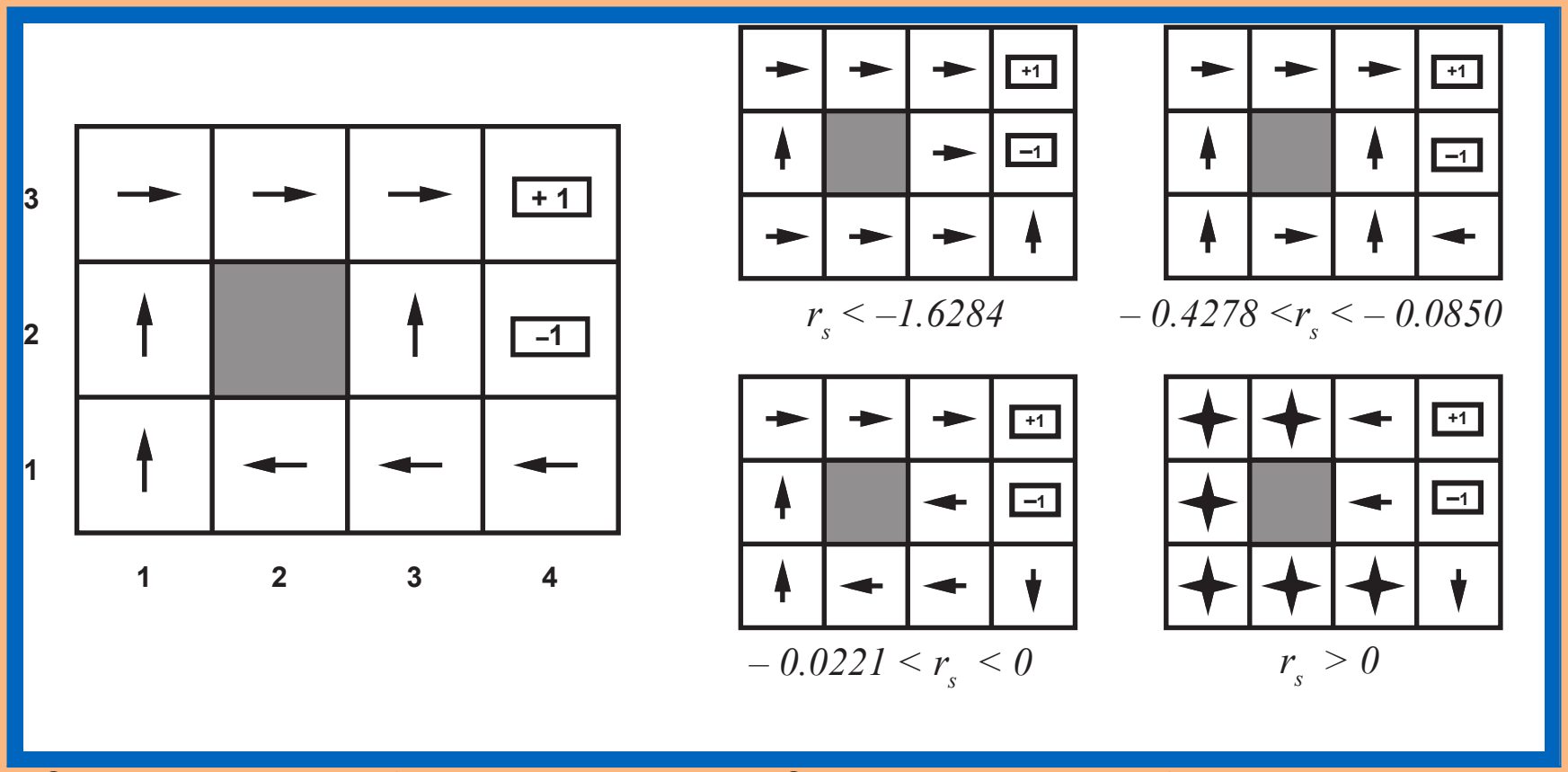
optimal policy

0.812	0.868	0.918	1
0.762		0.660	-1
0.705	0.655	0.611	0.388

state utilities

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Optimal Policies Under Different Rewards



Optimal policy for $\gamma = 1$ and $r_s = -0.04$ for non-terminal states.

Optimal policies for $\gamma = 1$ and different ranges of r_s for non-terminal states.

Reinforcement Learning

The goal of RL is to find a good (high-utility) policy in an *unknown* environment, i.e., the agent does not know P (86) and r (89).

After each action y following a state x, the agent observes the new state x' and receives $r(x')$.

From these (x', x, y) samples, it can learn an estimate $p(x' \mid x, y)$ of $P(x' \mid x, y)$.

- for small $|X|, |Y|$, p can be just a 3-dimensional array of relative frequencies
- larger $|X|, |Y|$ may require a parametric approach (such as a Bayes Network if independence structure available)

Learning $r(x)$ is easier - just store the reward for each state x visited.

Exploration vs. Exploitation

What actions should the agent take while collecting the (x', x, y) and r samples? Consider two options:

1. *Random* actions. Reason: a diverse sample will be collected, agent will not get stuck looping over a subset of states.
2. *Greedy* actions, i.e. optimal w.r.t. the current estimates

$$\pi(x) = \arg\max_y \sum_{x' \in X} p(x' \mid x, y) \widehat{U}(x') \tag{94}$$

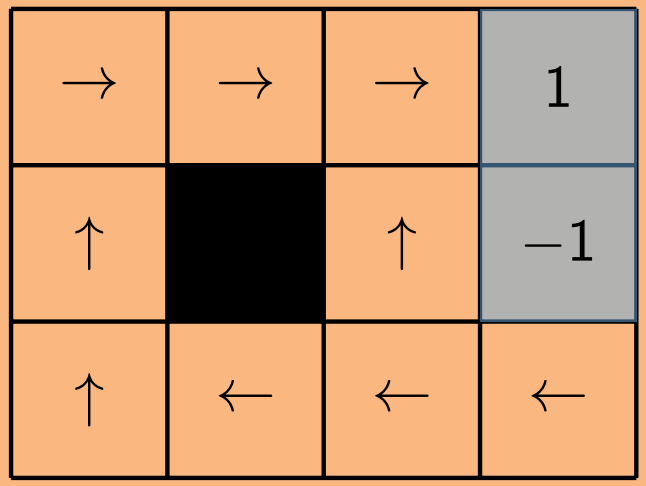
   Reason: the agent's goal is to maximize rewards.

This is the exploration-exploitation dilemma we have already investigated (although with a different estimated quantity).

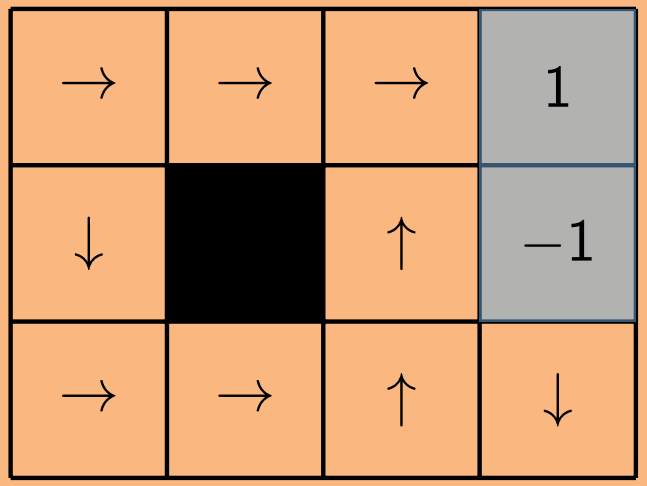
$\widehat{U}(x')$ is the estimate of $U(x')$ computed with p instead of P.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Greedy Approach



Optimal policy
($r_s = -0.04$)



Policy learned with greedy actions (94)

After each greedy action (94), the agent re-computes π by policy iteration using $p(x' \mid x, y)$ estimated from collected samples of x', x, y, r.

This process converges into a *non-optimal* policy.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

GLIE Approach

To balance btw. exploration and exploitation, we can adopt this strategy, i.e. instead of (94), take a random action with probability approaching 0.

This approach is called **GLIE** = *greedy in the limit of infinite exploration*.

A popular instance of GLIE is to take action y in state x with probability

$$\frac{e^{Q(x,y)/\tau}}{\sum_{y' \in Y} e^{Q(x,y')/\tau}} \tag{95}$$

where

$$Q(x,y) = \sum_{x' \in X} p(x' \mid x, y) \widehat{U}(x') \tag{96}$$

and the *temperature* $\tau \to 0$ with $k \to \infty$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Heuristic Approach: Exploration Function

An alternative to GLIE speeds up convergence by promoting underexplored x, y pairs by changing (93) into

$$\widehat{U}(x) = r(x) + \gamma \max_{y \in Y} f\left(\sum_{x' \in X} p(x' \mid x, y)\widehat{U}(x'), N(x, y)\right)$$

where f is an **exploration function** and $N(x, y)$ is the number of times action y has been taken in state x.

We want f to give the utility expectation (w.r.t. the estimated probability p) only for pairs x, y such that y has been already tried on x a sufficient number of times.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Heuristic Approach: Exploration Function (cont'd)

For other (under-explored) pairs x, y, it should yield an *optimistic* utility value attracting the agent. This can be done with

$$f(u, v) = \begin{cases} \max R \text{ if } v < m \\ u \text{ otherwise} \end{cases}$$

Recall that R is bounded so $\max R$ is finite.

Observe that the GLIE approach changes (randomizes) a policy that has been computed through policy iteration or value iteration.

On the other hand, the exploration function is used only with value iteration because it is used to modify (93).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Adaptive Dynamic Programming

An **adaptive dynamic programming** (ADP) agent is one that uses its estimated model p of P to calculate the state utilities and the policy by value or policy iteration.

(Both the iteration procedures are a form of dynamic programming.)

p is re-estimated on each new sample (x', x, y) and so the iteration procedures are also repeated on each time step.

This may be computationally costly.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Direct Utility Estimation

The costly procedures can be avoided by **direct utility estimation** (DUE)

- instead of estimating P from samples of x', x, y and then computing $U(x)$, $x \in X$ using the estimate
- estimate $U^\pi(x)$ directly from obtained rewards following some π

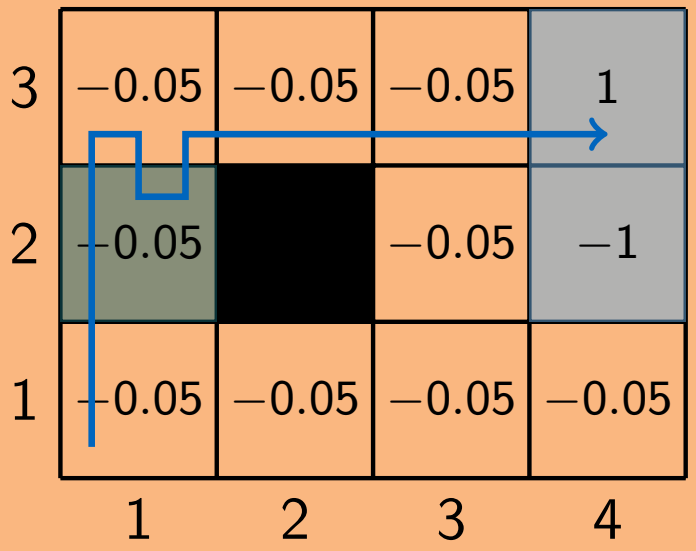
The estimate $\widehat{U}^\pi(x)$ of $U^\pi(x)$ is the average of **utility samples** for x. A utility sample (also called *reward-to-go*) for x is the sum of discounted rewards obtained on the path from x to a terminal state (end of episode).

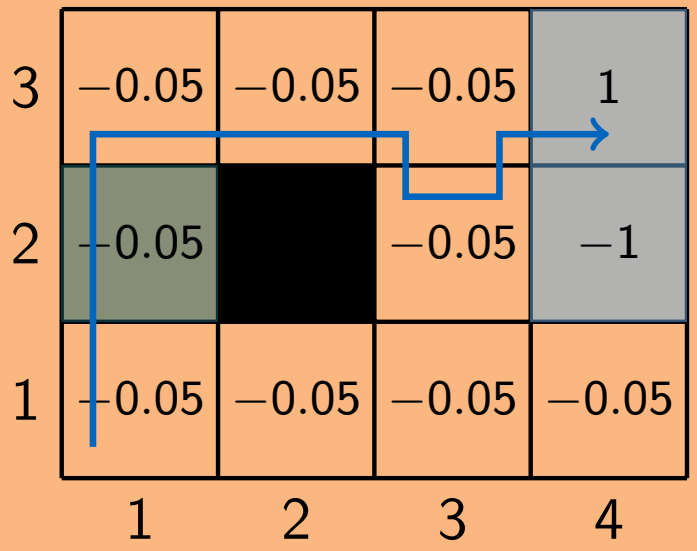
$\widehat{U}^\pi(x)$ converges to $U^\pi(x)$. To make it converge to $U(x)$, exploration is needed, e.g., using GLIE (with $\widehat{U}^\pi(x)$ plugged in for $\widehat{U}(x)$ in (96)).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Direct Utility Estimation

Estimate $\widehat{U}((1,2))$ with $\gamma = 1$ after two episodes





Episode 1: 2 samples
From 1st time in (2,1):
$6 \cdot (-0.05) + 1 = 0.7$
From 2nd time in (2,1):
$4 \cdot (-0.05) + 1 = 0.8$

Episode 2: 1 sample
$6 * (-0.05) + 1 = 0.7$

$$\widehat{U}((1,2)) = (0.7 + 0.8 + 0.7)/3 \approx 0.73$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Temporal Difference Learning

DUE ignores the strong correlation of utilities between adjacent states. The **temporal difference learning** method makes an explicit use of it.

First assume that x' always follows x under π ($x, x' \in X$, x non-terminal). That is, $P^\pi(x' \mid x) = 1$. Then the 2nd case of (90) changes to

$$U^\pi(x) = r(x) + \gamma U^\pi(x')$$

If this relation btw. $U^\pi(x)$ and $U^\pi(x')$ is not observed for the estimates $\widehat{U}^\pi(x)$ and $\widehat{U}^\pi(x')$ on a transition from x to x', i.e.,

$$\widehat{U}^\pi(x) \neq r + \gamma \widehat{U}^\pi(x')$$

(where r is the reward received in x) then the value of $\widehat{U}^\pi(x)$ must be 'corrected'.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Temporal Difference Learning (cont'd)

This is done by setting

$$\widehat{U}^{\pi}(x) := \widehat{U}^{\pi}(x) + \alpha_k \left(r + \gamma \widehat{U}^{\pi}(x') - \widehat{U}^{\pi}(x) \right) \tag{97}$$

moving $\widehat{U}^{\pi}(x)$ closer to the correct value $U^{\pi}(x)$ with speed given by the *learning rate* $\alpha_k \in \mathbb{R}$.

(97) is used as well when x can be followed by different states x' with non-zero probabilities. The higher $P^{\pi}(x' \mid x)$, the more times (97) is applied for x, x' during the interaction.

Lemma 8

Iterating (97), $\widehat{U}^{\pi}(x)$ *converges to* $U^{\pi}(x)$ *for all* $x \in X$ *if* $\sum_{k=1}^{\infty} \alpha_k = \infty$ *and* $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$.

DUE vs. TD vs. ADP

DUE tends to converge slower to the true state utilities than ADP in terms of time steps (k). But convergence may be faster in terms of runtime as the costly value or policy iteration is avoided.

TD is a middle ground between DUE and ADP.

Although DUE and TD do not need a model p of P for computing utilities, they still need it for evaluating the policy through (94) (greedy case) or (96) (GLIE case).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Q-Learning

Consider an alternative method **Q-Learning** which does not need p at all.

Instead of state utility, define the utility of a *state-action pair* called the **Q value** as

$$Q(x,y) = \begin{cases} r(x) \text{ if } x \text{ is terminal} \\ r(x) + \gamma \sum_{x' \in X} P(x' \mid x, y) \max_{y' \in Y} Q(x', y') \quad \text{otherwise} \end{cases} \tag{98}$$

Note: unlike $U^\pi(x)$, $Q(x,y)$ does not depend on a policy π.

Given an estimate $\widehat{Q}$ of Q, instead of (94), the greedy policy is given by

$$\pi(x) = \arg\max_{y \in Y} \widehat{Q}(x,y) \tag{99}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Q-Learning

Similarly to TD, the estimate $\widehat{Q}$ is iterated on each transition from x to x' where y is the action that taken on x and r is the reward in x:

$$\widehat{Q}(x,y) := \widehat{Q}(x,y) + \alpha_k \left(r + \gamma \max_{y' \in Y} \widehat{Q}(x',y') - \widehat{Q}(x,y) \right) \tag{100}$$

ultimately converging to $Q(x,y)$.

Exploration can be promoted by GLIE (95), using $\widehat{Q}(x,y)$ instead of (96) or by changing (99) into

$$\pi(x) = \arg\max_{y \in Y} f\left(\widehat{Q}(x,y), N(x,y) \right)$$

where f is an exploration function and $N(x,y)$ is the number of times x has occurred with y taken on it.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

SARSA Learning

In (100), the action maximizing $\widehat{Q}(x', y')$ is $\pi(x')$, i.e., the one chosen by the greedy policy (99) on state x'. So with a greedy policy, (100) simplifies into

$$\widehat{Q}(x, y) := \widehat{Q}(x, y) + \alpha_k \left(r + \gamma \widehat{Q}(x', y') - \widehat{Q}(x, y) \right) \tag{101}$$

where y' is the action taken on state x'.

The iteration (101) uses the *s*tate - *a*ction - *r*eward - *s*tate - *a*ction quintuplet

$$x, y, r, x', y'$$

hence this modification of Q-learning is called **SARSA**. It is used even with non-greedy policies.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Q vs. SARSA

The iteration (100) in Q-Learning is *less* dependent on the policy employed than (101) in SARSA because y' in the latter is determined by the policy.

For this reason, Q-Learning is called an **off-policy** strategy whereas SARSA is called an **on-policy**.

In Q-Learning, $\widehat{Q}$ tends to converge faster to Q even with a policy far from optimal.

In SARSA, it tends to converge faster with a *partially enforced* policy, i.e. where $\pi(x)$, $x \in X'$ are fixed for some $X' \subset X$. (E.g., in state $(1, 1)$ the agent must go up independently of state utilities.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Representation of $\widehat{U}$ and $\widehat{Q}$

If $\widehat{U}, \widehat{Q}$ are represented as arrays (one cell for each $x \in X$, they require at least $\mathcal{O}(|X|)$ resp. $\mathcal{O}(|X| \cdot |Y|)$ memory and time.

This would not scale to large state spaces X in real-life problems, e.g.,

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

A more compact ('generalized') model for

$$\widehat{U} : X \to \mathbb{R} \text{ or } \widehat{Q} : X \times Y \to \mathbb{R}$$

is needed that allows learning (updating) from $[x, y, r, x']$ or $[x, y, r, x', y']$ samples.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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

Consider learning $\widehat{U}$ with the DUE agent.

A simple option is to define a set of features, i.e., mappings $\phi^i : X \to \mathbb{R}$ informative of the utility, and use a *regression model*.

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

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

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

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

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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

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

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

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

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

and the iterative update:

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

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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

1. Observe:

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

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

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

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

2. Features allow to deal with a kind of *partial state observability* (similar to one considered here). If a fixed component of the state is not observable, features can be designed that do not use that component.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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

A similar approach can be applied with the Q-Learning agent:

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

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

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

$$w^i := w^i + \alpha \left(r + \gamma \max_{y'} \widehat{Q}(\mathbf{w}, x', y') - \widehat{Q}(\mathbf{w}, x, y) \right) \phi^i(x, y)$$

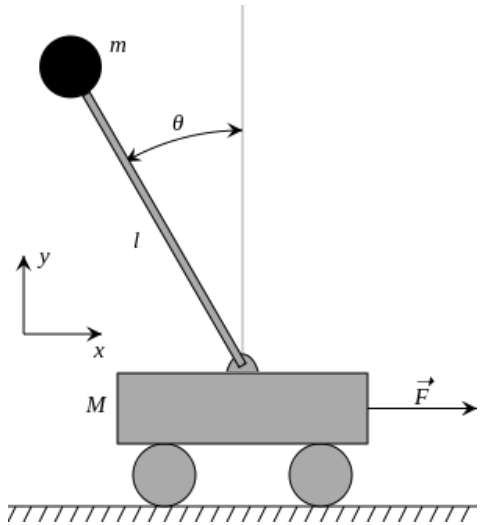
The principle is simple, the art is in designing good features ϕ^i.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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 (!).

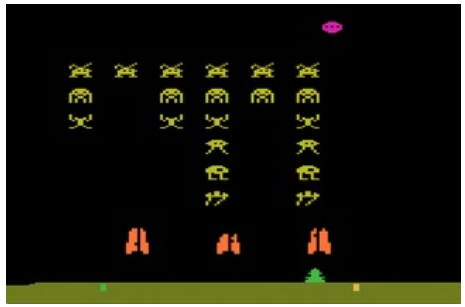
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

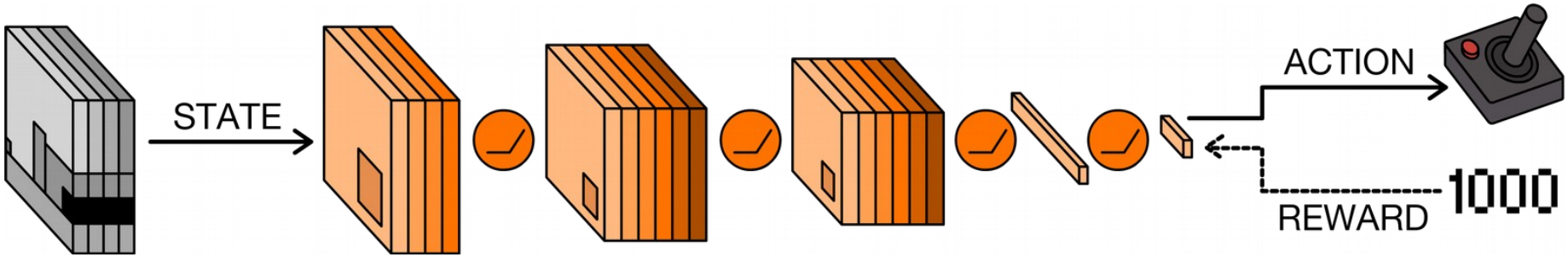
Deep Q-Learning: DQN

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

Deep feed-forward network approximating $Q(x, y)$

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



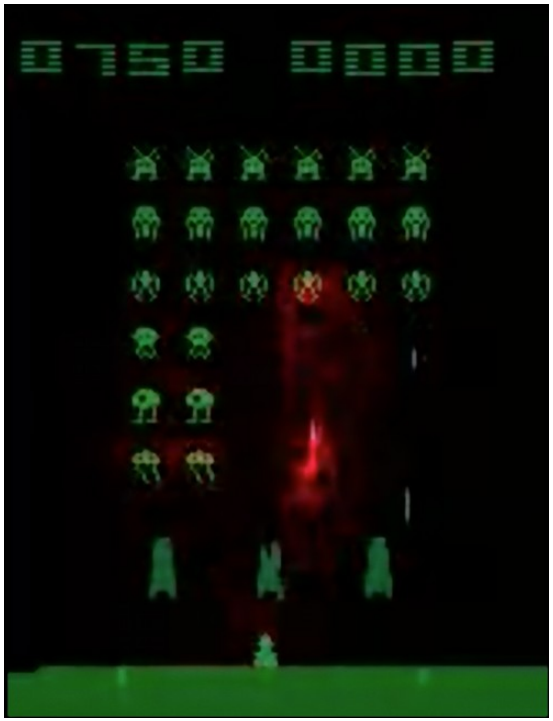


Demo

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 ϵ 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 $\vec{\phi}(x), y, r, \vec{\phi}(x')$ tuples recorded in the past.

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

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Policy Search

Instead of searching $\widehat{Q}$ / $\widehat{U}$ / p, search directly a good policy $\pi : X \to Y$, i.e. pose the problem as a classification problem.

A feature-based approach applicable again:

$$\pi(x) = \pi'(\phi_1(x), \ldots \phi_n(x))$$

Quality of π' is estimated as mean total rewards over repeated episodes using π.

Gradient-based search for a π' maximizing the quality not directly applicable since Y is finite (discrete), and π' thus not differentiable in its argument.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Differentiable Policy Search

Gradient-based policy search is possible with a *stochastic policy* choosing action y in state x with softmax probability

$$\frac{e^{q(\mathbf{w},\boldsymbol{\phi}(x,y))}}{\sum_{y' \in Y} e^{q(\mathbf{w},\boldsymbol{\phi}(x,y'))}}$$

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 $q : \mathbb{R}^{n+n'} \to \mathbb{R}$. An example (in which $n = n'$)

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

If q differentiable in all ϕ^i as above then $\mathbf{w}$ can be learned through stochastic gradient descent.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learning a Feature-Based Environment Model

The ADP agent derives its policy from a learned model of p of the transition distribution P.

Such a model can also be feature based. So $p(x'|x,y)$ can be represeted e.g. by

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

where ϕ^i are features of the x', x, y triple and w^i real parameters. This allows to use the gradient method as in the previous examples.

f need not even be normalized to the $[0;1]$ probability range if only used in $\arg\max_y$ expressions such as (94).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning of an Environment Model

Consider the following *Bayesian* approach *(not to be confused with Bayesian networks)*, which involves

- a countable (possibly infinite) set $\mathcal{M}$ of probability distributions ("model class")
- at each $k \in \mathbb{N}$, a probability distribution B_k on $\mathcal{M}$ where $B_k(p)$ ($p \in \mathcal{M}$) quantifies the belief that $P \equiv p$

The probability model at k is then assumed as

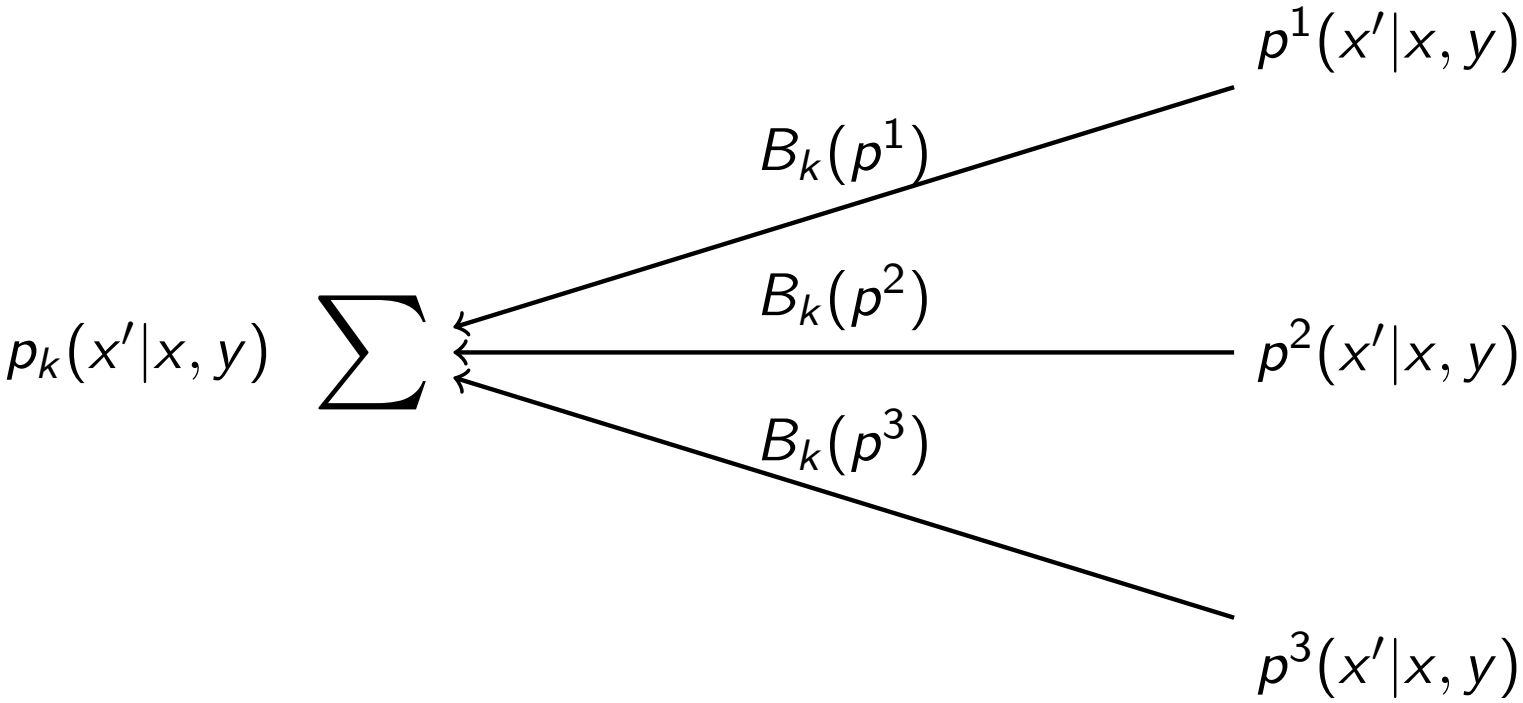
$$p_k(x'|x,y) = \sum_{p \in \mathcal{M}} p(x'|x,y) B_k(p) \tag{102}$$

i.e, a weighted sum where each distribution from $\mathcal{M}$ contributes the stronger the higher its belief.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning of an Environment Model (cont'd)

Current Model *Current Belief* *Possible Environments* $\mathcal{M}$



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning of an Environment Model (cont'd)

As soon as x_{k+1} has been observed, B_k is updated by the Bayes rule to the *posterior*

$$B_{k+1}(p) = \alpha p(x_{k+1}|x_k, y_k)B_k(p) \tag{103}$$

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

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

Theorem 23

If $p = P$ for some $p \in \mathcal{M}$ and $B_1(p) > 0$, then $p_k \to_{k\to\infty} P$.

So also the greedy policy (94) will converge to the optimal policy. *The Bayesian approach thus avoids the exploration-exploitation dilemma* but has a new element to supply: the initial belief B_1.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning - Continuous case

The Bayesian learning approach can be generalized to the continuous case where the model class $\mathcal{M}$ is uncountable.

For example, $\mathcal{M}$ may be the class of (conditional) normal distributions with all means and co-variances forming the real parameter vector $\mathbf{w}$.

Then instead of (102), we have

$$p_k(x_{k+1}|x_k, y_k) = \int_{\mathbf{w}} p(x_{k+1}|x_k, y_k, \mathbf{w}) B(\mathbf{w})$$

and the update (103) changes to

$$B_{k+1}(\mathbf{w}) = \alpha P(x_{k+1}|x_k, y_k, \mathbf{w}) B_k(\mathbf{w})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Universal Learning

Assumptions on Environments So Far

Recall the special assumptions we have imposed so far on the probabilistic desription (7) of the environment, which is the product of (8) and (9).

Mistake-bound learning: no assumption on observations, rewards by (18)

PAC learning: i.i.d. observations (14), rewards by (18) (or (73) for inconsistent learning)

Distribution learning (Bayes nets): i.i.d. observations (14), rewards case-specific, e.g. 0 for correct prediction of missing values, -1 otherwise

Reinforcement Learning: observations by (85) (Markov property), rewards by (89)

Let us now investigate if an agent can maximize the utility *without such assumptions*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Universal Sequence Prediction

For the stated goal, the agent necessarily must be able to predict the percept xr_k from the history $y_{\leq k}, xr_{<k}$ so that xr_k maximizes (7).

For a start, we will disregard actions and rewards, and focus on the central problem:

Universal Sequence Prediction Problem

Given a set X and a sequence

$$x_1, x_2, \ldots x_k \in X$$

predict x_{k+1} maximizing $P(x_{k+1} \mid x_{\leq k})$ without knowing P.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Sequence Prediction

Some sequences seem obvious to extend. E.g.

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

because of the pattern $x_{k+1} = x_k + 1$.

But e.g.

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

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

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 x_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?

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing a Sequence

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

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

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

Here we assume the prefix Turing machine *which as an* input tape, output tape *(both unidirectional) and any finite number of bidirectional* working tapes. *This is just for convenience, its expressive power is same as that of the vanilla Turing machine.*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing a Sequence (cont'd)

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.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity

The **Kolmogorov complexity** $K_T(q)$ of $q \in X^*$ 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.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity (cont'd)

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 $\langle T \rangle \in X^*$ of T as a distinguished part of its input.

$\langle T \rangle$ can be distinguished from p on the input tape e.g. by putting the string $0^{|\langle T \rangle|}1\,\langle T \rangle\, p$ on it so that the number of leading zeros indicates the length of $\langle T \rangle$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity (cont'd)

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

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

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 $q \in X^*$:

$$K(q) \leq |q| + c \tag{104}$$

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.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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, 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

Theorem 24

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

Proof: Assume a deciding program p exists. Consider the first (in the lexicographic order) string $q \in X^*$ such that $K(q) \geq n$.

Then q can be generated as follows: for each $q \in X^*$ 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 $K(q) < n$, yielding a contradiction. So p does not exist.

Kolmogorov Complexity - Computability (cont'd)

Proof idea intuitively: define q as the

> shortest string than cannot be specified with fewer than twelve words.

But q has just been specified with eleven words!

This paradox implies that the property *can be specified with n words* is not decidable.

The property is analogical to $K(q) < n$ where $n = 12$ and 'specified with n words' means being output by a universal T.M. with a program of length n.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity - Enumerability

A function $f(x) : \mathbb{N} \to \mathbb{R}$ is **enumerable** if there is a Turing machine finitely computing a function $f(x, k)$ *(i.e., halts for each x,k written in the binary form on the input tape and writes $f(x, k)$ in the binary form on the output tape)* such that

- $\lim_{k\to\infty} f(x, k) = f(x)$
- $f(x, k) \leq f(x, k + 1)$

for $\forall k$. A function is **co-enumerable** under the same conditions except $\leq$ is replaced by $\geq$.

Theorem 25

K is co-enumerable.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity - Enumerability (cont'd)

Proof of Theorem 25: Use the following algorithm

1. $k := 1$, $K(x,k) := |q| + c$ (104). Start *all programs shorter than* $|q| + c$ in parallel.
2. $k := k + 1$. Make one step in all running programs.
   - If any program halts and has produced q, select the shortest among them and set $K(x,k)$ to its length; stop all programs of length $K(x,k)$ or greater;
   - otherwise $K(x,k) := K(x,k-1)$

   Go to 2.

Some of the programs started in 1 will never halt so this procedure will neither, and we will never know how close $K(q,k)$ is to $K(q)$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Unknown Environment

With zero knowledge about the environment, consider assigning *higher probabilities to simpler* x_k, i.e., sequences with with lower $K(x_k)$.

So e.g.

$$P(1|11111111) > P(0|11111111)$$

because $K(111111111) < K(111111110)$.

Note: $P(x_k|x_{<k}) = c \cdot P(x_{\leq k})$ *where* $c = 1/P(x_{<k})$ *is a constant after observing* $x_{<k}$.

The problem is that $K(x_{\leq k})$ depends only on the single shortest program generating $x_{\leq k}$. Intuitively, the probability of $x_{\leq k}$ should be higher if there are multiple simple programs each generating $x_{\leq k}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Universal Prior M

Solomonoff (1964) proposed the *universal prior* which is closely related to K but accounts for multiple generating programs:

$$M(x_{\leq k}) = \sum_{p:U(p)=x_{\leq k}*} 2^{-|p|} \tag{105}$$

where the sum is over all *minimal* programs for which the universal T.M. U outputs a string starting with $x_{\leq k}$, not necessarily halting.

(Minimal program = all the has been read from the input tape when all of $x_{\leq k}$ has appeared on the output tape. Ignores any 'useless' continuation of the input.)

So all minimal programs p generating $x_{\leq k}$ contribute to $x_{\leq k}$'s probability but short programs contribute exponentially more than long programs.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Universal Prior M: Properties

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

M is enumerable (proof omitted).

A function P is called a **semi-measure** if it satisfies all probability axioms but in the countable-union axiom $\sum_{i=1}^{\infty} P(e_i) = P(\cup_{i=1}^{\infty} e_i)$, $=$ is replaced by $\leq$.

M is a semi-measure. Indeed e.g.

$$M(000) + M(001) < M(00)$$

since there are programs outputting 00 and halting or looping forever afterwards without writing 0 or 1.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Universal Prior M: Properties (cont'd)

Theorem 26

For any computable $x_{\leq k} \in X^*$,

$$\lim_{k \to \infty} M(x_k | x_{<k}) = 1 \tag{106}$$

This means that after "seeing" the beginning $x_{<k}$ 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 recall the catch: M itself is not computable, only enumerable.*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Proof of Theorem 26

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

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

Swap the sum with the logarithm:

$$= -\frac{1}{2}\ln M(x_1)\cdot M(x_2|x_1)\cdot M(x_3|x_{<3})\cdot\ldots$$

Apply the chain-rule:

$$= -\frac{1}{2}\ln M(x_{1:\infty})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Proof of Theorem 26 (cont'd)

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

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

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

$$\sum_{k=1}^{\infty} (1 - M(x_k|x_{<k}))^2 < \infty \tag{107}$$

Finally, (106) must hold, otherwise 107 would not hold.

The AIXI Agent

Let us now see how M can be used in the agent-environment setting involving not only observations but also rewards and actions.

Assume for simplicity that rewards and actions are binary $Y = R = \{0, 1\}$.
(No loss of generality; any finite description can be expressed as a binary string.)

The percept history $xr_{\leq k}$ is then a binary string so $M(xr_{\leq k})$ is defined by (105).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

The AIXI Agent (cont'd)

To quantify the probability of percept history $xr_{\leq k}$ *given action history* $y_{\leq k}$, Hutter (2005) proposed to adapt (105) into

$$M(xr_{\leq m} \mid y_{\leq m}) = \sum_{p:U(p,y_{\leq m})=xr_{\leq k}*} 2^{-|p|} \tag{108}$$

where $m \in \mathbb{N}$ and the sum is over all programs for the universal T.M. which outputs $xr_{\leq m}$ (followed by any suffix) *given* $y_{\leq m}$ *on the input tape*.

The program p can be distinguished from its input $y_{\leq m}$ *on the input tape e.g. by the coding described here.*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

The AIXI Agent (cont'd)

The agent which executes the sequence of actions

$$y_{\leq m} = \arg\max_{y_{\leq m}} \sum_{xr_{\leq m}} \left(M(xr_{\leq m} \mid y_{\leq m}) \sum_{k=1}^{m} r_k \right) \tag{109}$$

is the **AIXI agent**.

Theorem 27

The AIXI agent is Pareto-optimal with respect to maximizing utility (10), i.e., there is no other agent that performs at least as well as AIXI in all environments while performing strictly better in at least one environment.

Our presentation of AIXI is simplified. The resources here provide a proof for the theorem and also explain how the optimal action y_k is computed from the history $xr_{\leq k}, y_{\leq k}$ and how AIXI is defined for an infinite horizon ($m \to \infty$).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

M as a Bayesian Mixture

Define $K(f)$ for a function $f : \mathbb{N} \to \mathbb{R}$ as the length of the shortest program computing *(a binary representation of)* $f(q)$ given *(a binary representation of)* input q.

Theorem 28

Let $\mathcal{M}_U$ be the set of all enumerable semi-measures. $M(q)$ (106) *is equivalent to*

$$\xi(q) = \sum_{p \in \mathcal{M}_U} p(q) \cdot 2^{-K(p)} \tag{110}$$

in the sense that

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

Note that $M \in \mathcal{M}_U$ because it is an enumerable semi-measure.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

M as a Bayesian Mixture (cont'd)

Online sequence prediction with ξ can be done with iterative updates: the model at k is

$$\xi_k(x_{k+1} \mid x_{\leq k}) = \sum_{p \in \mathcal{M}_U} p(x_{k+1} \mid x_{\leq k}) B_k(p) \tag{111}$$

the initial beliefs in models $p \in \mathcal{M}_U$ are

$$B_1(p) = 2^{-K(p)} \tag{112}$$

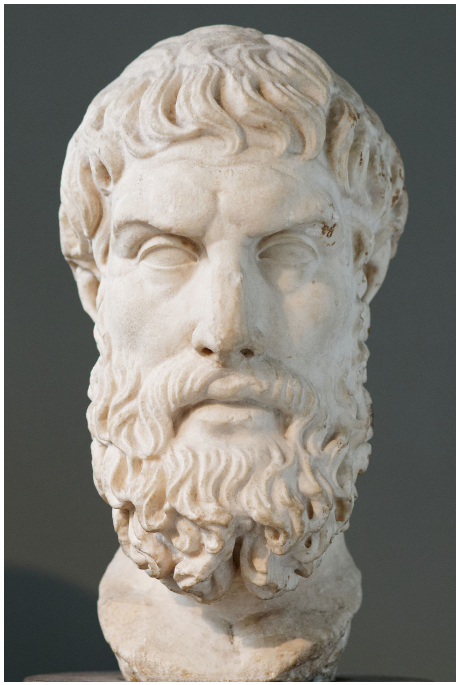
and they are updated (with the normalizer α) by

$$B_{k+1}(p) = \alpha p(x_{k+1} \mid x_{\leq k}) B_k(p) \tag{113}$$

(111) and (113) are analogical to (102) and (103), but here the model class $\mathcal{M}_U$ is the 'largest possible' and the initial beliefs are determined by (112).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Principle of Multiple Explanations



Epicurus

Greek philosopher

cca 342-270 B.C.

Keep all explanations consistent with the observations.

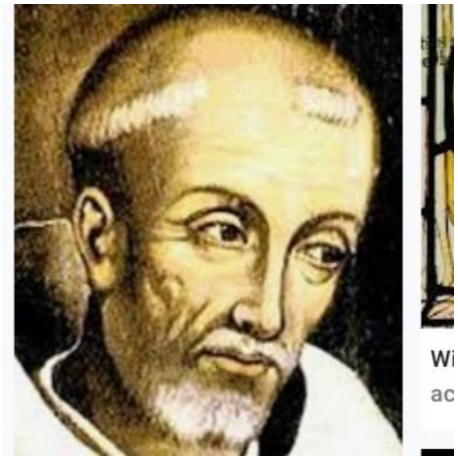
We have applied this principle in the version space algorithm.

But how to make a prediction using multiple explanations?

- In version space, we took the majority vote among all hypotheses. They all had the same voting weight.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Principle of Simplest Explanation



William of Ockham

English philosopher

cca 1290-1349 B.C.

Entities should not by multiplied beyond necessity

or

Keep the simplest explanation consistent with the observations.

Famously known as the Occam's razor

Kolmogorov complexity provides a mathematical way to determine 'simplest'.

But is it reasonable to discard all other explanations?

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Inference

Thomas Bayes

English mathematician

cca 1701-1761 B.C.

ξ combines the two principles using Bayesian inference.

$$\xi(q) = \overbrace{\underbrace{\sum_{p \in \mathcal{M}_U}}_{\text{All models contribute. (Epicurus)}} p(q) \cdot \underbrace{2^{-K(p)}}_{\text{Simpler models get more weight. (Ockham)}}}^{\text{weighted mixture (Bayes)}}$$