

# Symbolic Machine Learning

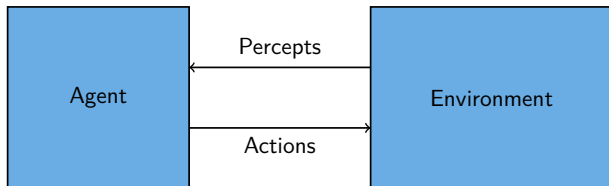
## Lecture Slides

Filip Železný

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



# Agent Interacts with Environment



- Discrete *time*

$$k = 1, 2, \dots$$

- *Percepts*

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

- *Actions*

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

## History

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

has probability

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

History influenced by agent's deterministic *policy*

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

so

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

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

$$x_k = (o_k, r_k)$$

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

Note:  $r_1$  is immaterial.

Define marginals on  $O$  and  $R$  so that

$$\mu(x_k | xy < k) = \mu(o_k, r_k | xy < k) = \mu_O(o_k | r_k, xy < k) \mu_R(r_k | xy < k)$$

# Nonsequential Interaction

*Nonsequential*: a special, simplified case of interaction.

- Observations are i.i.d.:

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

Note: very strong assumption!

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

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

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

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

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

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

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

A model for most “data-mining” scenarios.

# Rewards in Batch Learning

In the action phase ( $k > K$ )

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

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

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

where

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

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

# Agent's Goal

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

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

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

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

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

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



# Environment: State-Based Description

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

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

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

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

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

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

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

# Agent: State-Based Description

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

Generation of actions  $y_k$ :

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

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

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

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

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

# State-Based Interaction Model

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

Agent

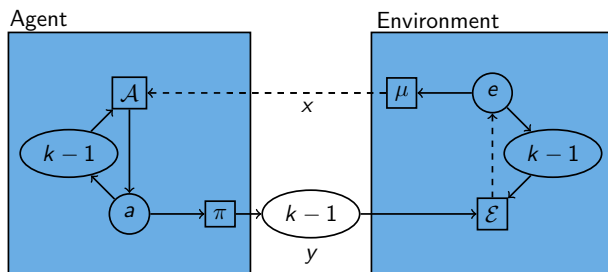
actions:  $y = \pi(a_k)$

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

Environment

percepts:  $x_k \sim \mu(x_k|e_k)$

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



# Non-sequential and Batch with States

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

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

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

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

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

Agent should reach state maximizing the expected reward

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

# On-line Concept Learning: Example

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

# Concept Learning Assumptions

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

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

Further simplifying assumptions (for convenience):

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

# Decisions and Rewards in Concept Learning

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

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

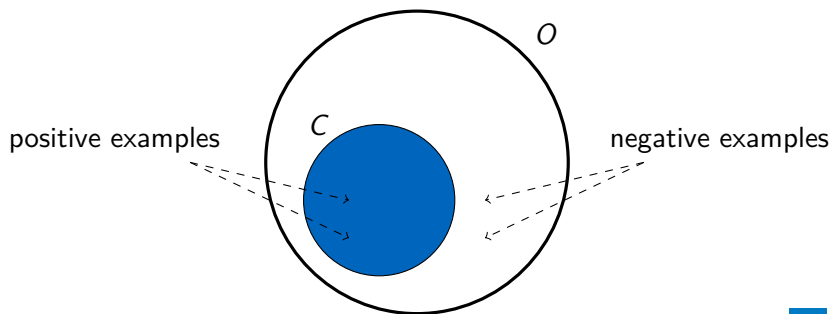
- Note: this is a special case of the *loss function*

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

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

- Environment (target) concept:

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





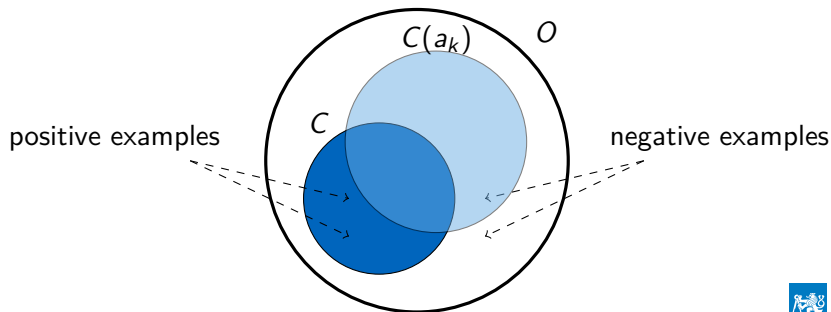
# Concept

- Environment (target) concept:

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

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

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



# Designing a Concept Learning Agent

Note on concept-learning terminology:

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

First thought:

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

# Agent with a Hypothesis

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

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

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

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

# Updating a Hypothesis

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

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

$$a_k = (o_k, h_k)$$

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

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

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

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

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

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

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

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

# Generalizing Agent: Update Step Example

For  $n = 4$ :

$$h_1 = \quad p_1 \wedge \neg p_1 \quad \wedge p_2 \wedge \neg p_2 \quad \wedge p_3 \wedge \neg p_3 \quad \wedge p_4 \wedge \neg p_4$$
$$o_1 = ( \quad \quad \quad 1, \quad \quad \quad 1, \quad \quad \quad 0, \quad \quad \quad 0 )$$
$$y_1 = 0$$

$$r_2 = -1$$

$$h_2 = \quad p_1 \quad \quad \quad \wedge p_2 \quad \quad \quad \wedge \neg p_3 \quad \quad \quad \wedge \neg p_4$$

# Generalizing Agent: Update Step

Formally:

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

where

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

That is, retains exactly the consistent literals.

# Generalizing Agent: How Well Does it Learn?

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

Observations (see textbook for more rigor):

- 1 All literals of  $h^*$  are consistent with any positive example.
- 2 If a hypothesis misclassifies a negative example, it has all literals consistent with the example.
- 3 From 1 and 2: no literal that is in  $h^*$  is removed from the agent's hypothesis. Since  $h_1$  contains *all* literals, we have  $h_k \supseteq h^*$ ,  $\forall k \in N$ .
- 4 From 3: mistakes are made only on positive examples.
- 5 From 3: on each mistake, at least one literal in the hypothesis is inconsistent and gets deleted.



# Generalizing Agent: Mistake Bound

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

So the cumulative reward is

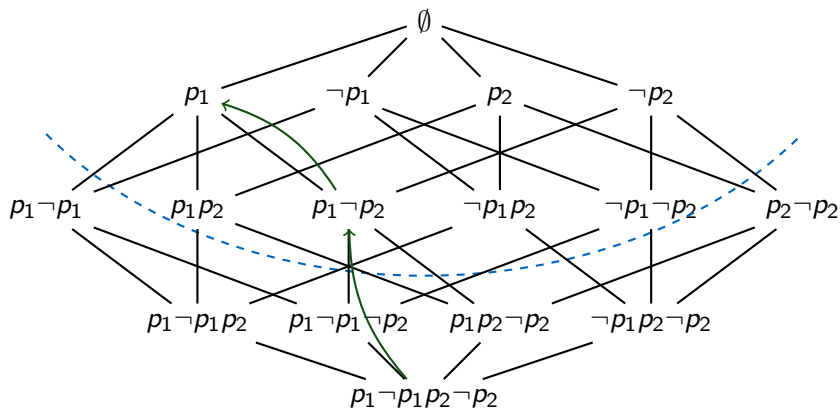
$$\sum_{k=1}^{k'} r_k \geq -2n$$

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

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

# Subsumption Lattice

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



# Subsumption and Entailment

For *conjunctions*, subsumption implies logical entailment:

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

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

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

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

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

# Generalizing Agent for Disjunctions

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

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

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

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

# Generalizing Agent for $s$ -CNF, $s$ -DNF

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

With  $n$  variables,  $n' = \binom{2n}{s}$  different  $s$ -clauses can be made. Assign a new propositional variable  $p'_i$  ( $1 \leq i \leq n'$ ) to each of them.

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

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

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

## Sidenote: Non-Binary Observations

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

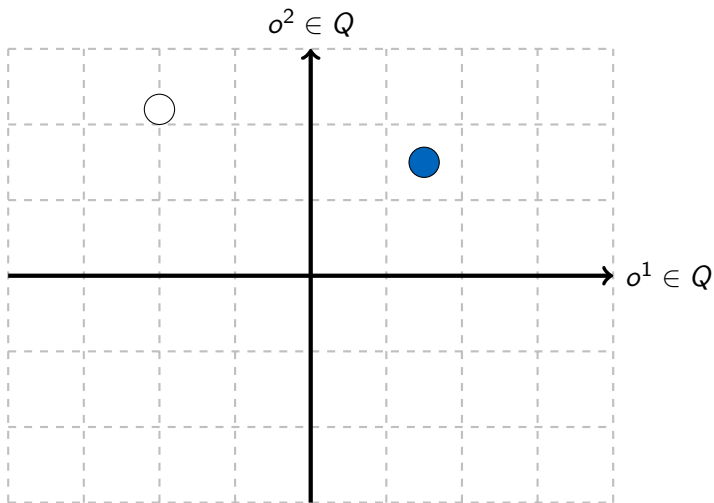
One way is to discretize  $o_k^i$  along each axis  $i$

$$\begin{array}{lll} o_k^1 > \theta_1^1 & o_k^1 > \theta_2^1 & \dots \\ o_k^2 > \theta_1^2 & o_k^2 > \theta_2^2 & \dots \end{array}$$

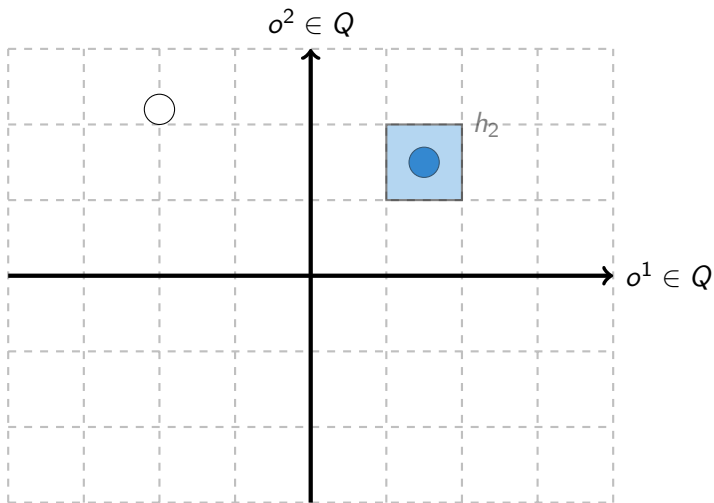
obtaining a (bigger) set of binary observations tuples we are 'used to'.

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

# Visualizing Generalization (with Rational Features)

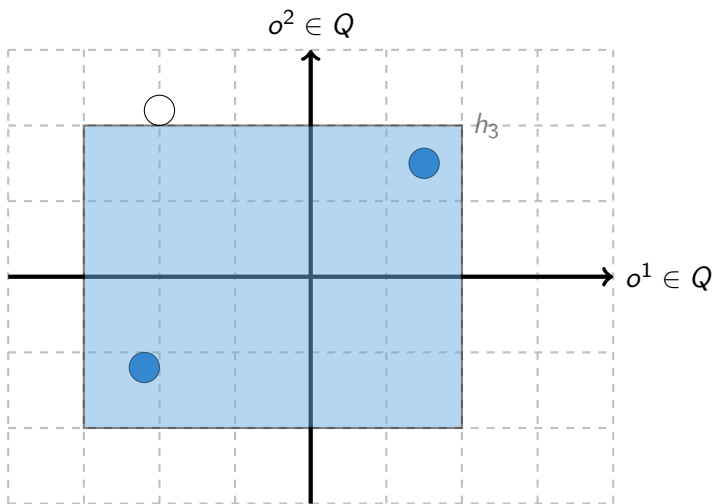


# Visualizing Generalization (with Rational Features)





# Visualizing Generalization (with Rational Features)



# Separating agent

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

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

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

Initial hypothesis:

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

Threshold decisions:

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

## Separating agent: Update Step

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

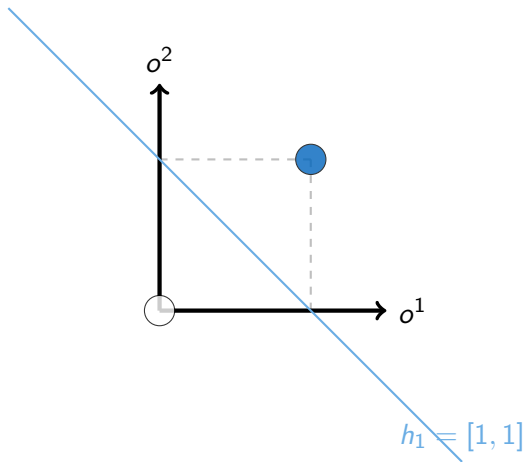
such that for  $h_k = \text{update}(\alpha, h_{k-1}, o_{k-1})$

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

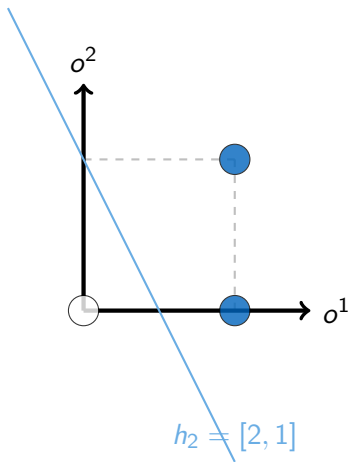
This means

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

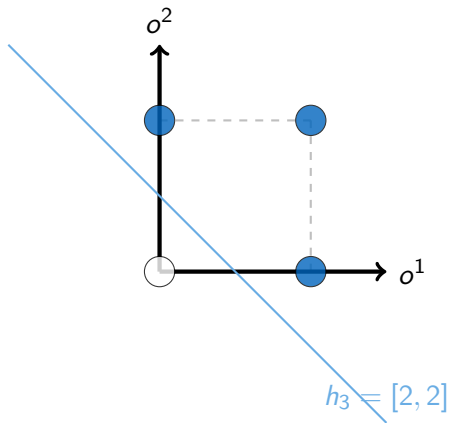
# Visualizing Separation



# Visualizing Separation



# Visualizing Separation



# Separating agent: How Well Does It Learn?

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

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

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

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

Better than the generalizing agent for sufficiently small  $s$ .

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

# Hypothesis and Concept Classes

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

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

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

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

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



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

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

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

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

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

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

Decides by a majority vote among all retained hypotheses:

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

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

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

for any horizon  $k' \in \mathbb{N}$ .

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

## On-Line (Mistake-Bound) Learning

Agent *learns class  $\mathcal{H}$  on-line (in the mistake-bound model)* if it makes at most  $poly(n)$  of mistakes in the on-line scenario provided that  $\mathcal{C}(\mathcal{H})$  contains the target concept. We call  $\mathcal{H}$  *learnable* on-line if there is an algorithm (agent) that learns it on-line.

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

So both of the agents so far learn conjunctions, disjunctions,  $s$ -CNF,  $s$ -CNF on-line.

## Efficient On-Line Learning

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