Symbolic Machine Learning

Filip Železný and Jiří Kléma

Contents

1 A General Framework

1.1 Percepts and Actions

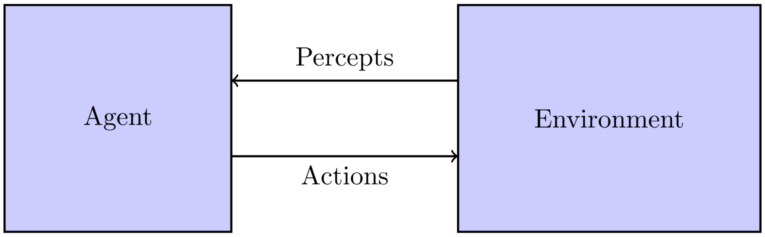


Figure 1: The basic situation under study.

- Discrete *time*
$$k = 1, 2, \ldots$$
- *Percepts*
$$\forall k : x_k \in X$$
- *Actions*
$$\forall k : y_k \in Y$$

X and Y are finite.

A *history* is a sequence of alternating percepts and actions, i.e,

$$x_1, y_1, x_2, y_2, \ldots, x_k, y_k$$

and is denoted as $xy_{\leq k}$. Similarly, $xy_{<k} = x_1, y_1, x_2, y_2, \ldots, x_{k-1}, y_{k-1}$. There is a probability distribution μ on histories

$$\mu(xy_{\leq k}) = \mu(x_1)\mu(y_1|x_1)\mu(x_2|x_1, y_1)\ldots\mu(x_k|xy_{<k})\mu(y_k|x_k, xy_{<k}) \tag{1}$$

After the initial 'kick-off' x_1 from the environment distributed according to $\mu(x_1)$, any percept x_k generated by the environment at time k depends on the entire preceding history $xy_{<k}$ according to

$$\mu(x_k|xy_{<k}) \tag{2}$$

Actions y_k are determined by agent's decision *policy* which also depends on the history as well as the current percept and are distributed according to

$\mu(y_k|x_k, xy_{<k})$. We will assume that the policy is *deterministic*. Thus we identify the policy with function $\pi : (X \times Y)^* \times X \to Y$, so

$$y_k = \pi(xy_{<k}, x_k) \tag{3}$$

This means that $\mu(y_k|xy_{<k}, x_k) = 1$ for $y_k = \pi(xy_{<k}, x_k)$ and 0 otherwise.

The following diagram illustrates the influences between the introduced variables.

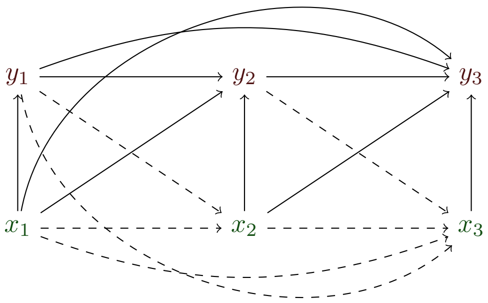


Figure 2: Influence diagram for actions y_k and percepts x_k for $1 \leq k \leq 3$ with full lines indicating deterministic influences (via π) and dashed lines showing probabilistic influences (via μ).

While we have yet to define what goals the agent should achieve through interaction with the environment, obviously some histories will be “better” than others in terms of the goal achievement. To maximize the probability (1) of good histories, the agent cannot influence the conditional probability (2), which is inherent to the environment, but it can follow a good policy (3). However, the effect of actions proposed by the policy depends on (2) which is generally not known to the agent. So the agent needs to recognize the environment by experimenting with it. This is formally reflected by (3) where action y_k depends not only on the current percept x_k but also on the history $xy_{<k}$. So the agent will generally make different decisions $y_k \neq y_{k'}$ for $k > k'$ even if $x_k = x_{k'}$ because the experience $xy_{<k}$ at time k is larger than experience $xy_{<k'}$ at time k'. This is our first reflection of *learning*.

How does the agent know how well it is doing? This information comes from the environment through a specially distinguished part of the percepts, called *rewards*. The remaining part of each percept contains *observations*. Formally, $X = O \times R$, $o_k \in O$, $r_k \in R \subset \Re$, so

$$x_k = (o_k, r_k) \tag{4}$$

Since X is assumed finite, it follows that rewards have their finite minimum and maximum.

The probability of x_k in (2) can be written in terms of the marginals μ_O and μ_R

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

which also makes it clear that o_k and r_k are in general not mutually independent, even if conditioned on $xy_{<k}$.

1.2 Nonsequential Cases

Scenarios where current percepts depend on the history of previous percepts and actions are called *sequential*. The framework described so far is maximally general in that dependence is assumed on the entire history from $k = 1$ on. On the other extreme are *nonsequential* scenarios. Here, observations are independent of the history as well as the current reward, i.e.

$$\mu_O(o_k|r_k, xy_{<k}) = \mu_O(o_k) \tag{5}$$

and thus $o_1, o_2, \ldots$ are mutually independent random variables sampled from the same distribution μ_O (they are "i.i.d.").

Rewards in the nonsequential case are assumed to depend only the immediately preceding observation and the action taken on it, i.e.

$$\mu_R(r_k|o_k, xy_{<k}) = \mu_R(r_k|o_{k-1}, y_{k-1}) \tag{6}$$

however, since y_{k-1} is functionally determined by the history $xy_{<k-1}$ and percept $x_{k-1} = (o_{k-1}, r_{k-1})$ through (3), we may rewrite (6) as

$$\mu_R(r_k|o_{k-1}, r_{k-1}, xy_{<k-1}) \tag{7}$$

which makes it clear that reward r_k depends on previous rewards, and thus rewards $r_1, r_2, \ldots$ are not i.i.d.. This is natural since if they were, it would mean the agent never improves its performance.

1.3 Batch Learning

We will also consider a specific yet important nonsequential case called *batch learning* consisting of two phases switching right after time K

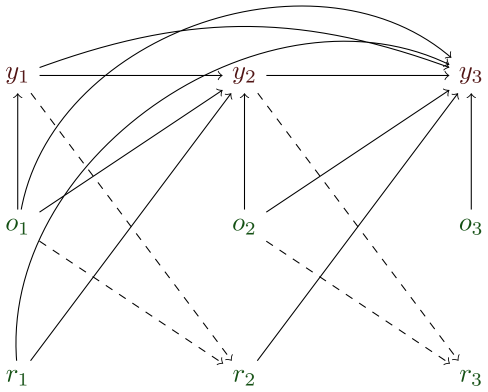


Figure 3: Influence diagram for actions y_k, observations o_k, and rewards r_k for $1 \leq k \leq 3$ with full lines indicating deterministic influences (via π) and dashed lines showing probabilistic influences (via μ) in the nonsequential case.

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

In the action phase, the agent no longer changes its decision making, i.e.

$$\text{if } k, k' > K \text{ and } x_k = x_{k'} \text{ then } y_k = y_{k'} \tag{8}$$

and ignores rewards. So the action proposed by the policy depends only on the current observation and the history only up to time K. So for $k > K$, (3) changes here into

$$y_k = \pi(xy_{\leq K}, o_k) \tag{9}$$

and (6, 7) change into

$$\mu_R(r_k|o_{k-1}, y_{k-1}) = \mu_R(r_k|o_{k-1}, xy_{\leq K}) \tag{10}$$

because due to (9), y_{k-1} is determined by o_{k-1} and $xy_{\leq K}$. The observation o_{k-1} does not depend on rewards due to (5). So reward r_k does not depend on previous rewards $r_{k'}$, $k > k' > K$. Another way to say this is that rewards in the action phase are conditionally independent of each other, given the learning phase history:

$$\mu_R(r_k, r_{k'}|xy_{<K}) = \mu_R(r_k|xy_{<K})\mu_R(r_{k'}|xy_{<K}) \tag{11}$$

The following figure illustrates the batch-learning situation.

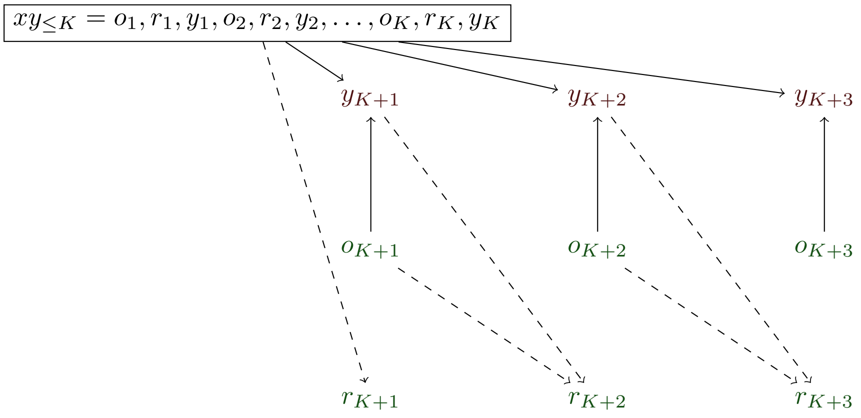


Figure 4: Influence diagram for actions y_k, observations o_k, and rewards r_k in the action phase ($k > K$) of batch learning with full lines indicating deterministic influences (via π) and dashed lines showing probabilistic influences (via μ). The top row indicates the influence of the learning phase on the agent's decisions in the action phase.

We can further express the distribution of r_k ($\forall k > K$) without conditioning on the observations, which are i.i.d. by (5)

$$\mu_R(r_k|xy_{\leq K}) = \sum_{o_{k-1} \in O} \mu_O(o_{k-1}) \mu_R(r_k|o_{k-1}, xy_{\leq K}) \tag{12}$$

So rewards in the action phase are i.i.d. according to the above distribution conditioned only on the history of the learning phase.

1.4 Rewards and Goals

It has been obvious that the agent's goal is to maximize rewards. Here we formalize this goal. Since rewards come at each point of the history, we want the agent to maximize their sum up to a finite time *horizon* $m \in N$

$$r_1 + r_2 + \ldots + r_m$$

or, more generally, maximize the *discounted* sum

$$\sum_{k=1}^{\infty} r_k \gamma_k$$

where $\forall k : \gamma_k \geq 0$ and $\sum_{i=1}^{\infty} \gamma_i < \infty$, so the above sum converges.

But since rewards are probabilistic, the agent should choose a sequence $y_{\leq m}$ of actions leading to a high *expected* cumulative reward

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

or, in the discounted case

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

where the first sum in both cases goes over all possible reward sequences $r_{\leq m}$ (since R and m are finite, there is a finite number of them).

However, for the specific case of *batch learning*, we establish a more appropriate learning goal. First, we do not care about maximizing rewards in the *learning phase* as the purpose of this phase is to probe the environment even at the price of possibly poor rewards. Second, in the *action phase* after time K, the rewards r_k, $k > K$ are sampled independently from the same distribution (12) so we can simply maximize their expectation with respect to this distribution

$$\sum_{r_k \in R} \mu_R(r_k|xy_{\leq K}) r_k \tag{13}$$

It is again obvious from the formula that the expected reward only depends on the learning phase history $xy_{\leq K}$, after which the agent no longer changes its action policy. Note also that the batch learning scenario allowed us to define an objective (13) without the need to choose the parameters m or γ_k ($k = 1, 2, \ldots$) needed in the sequential scenario.

1.5 Environment States

With the exception of the non-sequential scenario, our framework has been very general in that percepts x_k generally depend on entire histories $xy_{<k}$. In the real world, many histories may be equivalent, i.e. leading to the same probabilities of x_k conditioned on action y_{k-1}. This can be formalized through the notion of *environment state* $s_k \in S$ at time k.

For generality, let us first assume that the state is probabilistically established by the preceding state, the last percept, and the last action through the following state *update* distribution

$$\mathcal{S}(s_k|s_{k-1}, x_{k-1}, y_{k-1}) \tag{14}$$

and that this state generates the current percept

$$\mu(x_k|s_k) \tag{15}$$

This modification does not lessen the generality of the framework if we allow S to be infinite as then there could simply exist a distinct state for each possible history (there is an infinite number of possible histories for unbounded k). Indeed, if one instantiates the distribution (14) to the functional dependence

$$s_k = s_{k-1} \parallel (x_{k-1}, y_{k-1}) \tag{16}$$

where $\parallel$ denotes concatenation, s_k will simply collect the entire history and its occurrence in (15) would be just a different name for $xy_{<k}$ in (2). However, we will make the important assumption that the number of possible states is finite

$$|S| < \infty \tag{17}$$

which will significantly simplify the framework. In practical tasks, there will be far fewer states than possible histories.

We can afford further simplifying assumptions under which the state-based framework will still encompass the learning scenarios we are going to elaborate. First, we will assume that the influence between environment states and the emitted percepts are single-directional. In particular, the percepts depend on states by (15) but not vice versa, so we remove x_{k-1} from (14)

$$\mathcal{S}(s_k|s_{k-1}, x_{k-1}, y_{k-1}) = \mathcal{S}(s_k|s_{k-1}, y_{k-1}) \tag{18}$$

As a consequence, the state cannot collect the history of percepts as in (16) but it can still collect the history of actions

$$s_k = s_{k-1} \parallel y_{k-1} \tag{19}$$

If the state evolves according to (19) then the percept in (15) depends on all historical states $s_{k-1}, s_{k-2}, \ldots, s_1$ as well as all historical actions $y_{k-1}, y_{k-2}, \ldots y_1$ embedded in them, and not on any other factors. So instead of assuming the specific update rule (19), we may equivalently assume that the state evolves in any other way but the state-percept dependencies are preserved, so that percepts are sampled according to

$$\mu(x_k|s_k, s_{k-1}, s_{k-2}, \ldots, s_1, y_{k-1}, y_{k-2}, \ldots, y_1) \tag{20}$$

Our simplification plan is to remove some of the dependencies above. We will do it differently for the two components of the percept, i.e. the observations

$$\mu_o(o_k|r_k, s_k, s_{k-1}, s_{k-2}, \ldots, s_1, y_{k-1}, y_{k-2}, \ldots, y_1) \tag{21}$$

and the rewards

$$\mu_r(r_k|o_k, s_k, s_{k-1}, s_{k-2}, \ldots, s_1, y_{k-1}, y_{k-2}, \ldots, y_1) \tag{22}$$

In particular, the observation will depend only on the current state and the last agent's action

$$\mu_o(o_k|s_k, y_{k-1}) \tag{23}$$

and the reward will depend on the last state and the action taken immediately on it

$$\mu_r(r_k|s_{k-1}, y_{k-1}) \tag{24}$$

1.6 Agent Hypotheses

A reasoning similar to the previous section applies to the agent, whose actions generally depend on the entire history as in (3). Again, many histories can lead to the same mapping from percepts to actions, for example because the agent has built the same hypothesis about the environment throughout the different histories. So analogically to the environmental states, we introduce the notion of agent's *hypothesis* $h_k \in H$. Since we work with deterministic agents, we will assume that the hypothesis is updated given the current percept through a functional prescription

$$h_k = \mathcal{H}(h_{k-1}, x_k) \tag{25}$$

and instead of (3), we will assume that actions depend on the (updated) state rather than the history, and the current observation

$$y_k = \pi(h_k, o_k) \tag{26}$$

Unlike in (3), explicit dependence on x_k is no longer needed in (26) as the latter can always be stored as part of h_k in (25). However, we do keep the o_k component of x_k as an argument of π because this will allow us to describe conveniently cases where the agent's hypothesis is kept constant and the actions depends only on their immediately preceding observation. This will in particular include the batch-learning case discussed below in the present context of state-based descriptions.

Again, we will postulate that

$$|H| < \infty \tag{27}$$

The formalization using enviroment states and agent hypotheses results in the agent and environment structures depicted in Fig. 5. The diagram of variable influences is shown in Fig. 6.

The agent hypothesis h_k has a very natural interpretation as it corresponds to the agent's model of the environment at time k, whereas π is the the interpreter of the model.[1] For example, h_k may encode a set of logical rules, and π may be a logical prover deriving actions as logical consequences of the rules. Since the hypothesis description has to fit in a finitely bounded memory, there can be only a finite number of different hypotheses. Therefore, the assumption in (27) is well justified.

The history of percepts and actions (in combination with the current percept) is obviously informative for updating the hypothesis so it seems the hypothesis update in (25) should also include previous percepts $x_{k-1}, x_{k-2}, \ldots$ and actions

[1] We might as well call h_k a *model* rather than a *hypothesis* but that would cause terminology clash in cases where the h_k is expressed in the formalism of logic, where the word *model* is already established and has a different meaning.

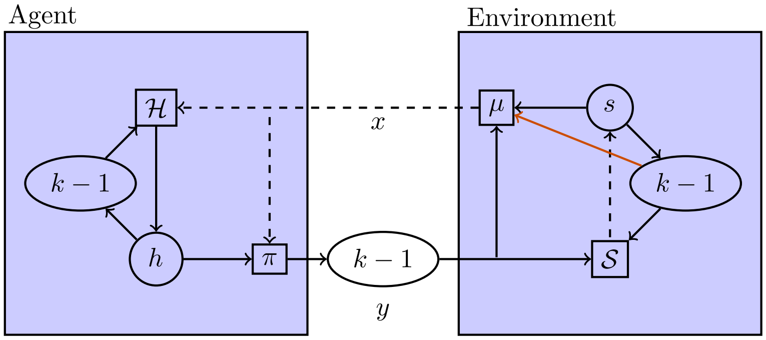


Figure 5: The state-based scheme of agent-environment interaction. Full and dashed lines denote functional and probabilistic influences, respectively. The $k-1$ nodes denote a one-step time lag. The highlighted dependence is only relevant for the reward part r of the percept x generated by μ; if the diagram only captured observations o and actions y, it would not contain this dependence and thus would be symmetric.

$y_{k-1}, y_{k-2}, \ldots$ as arguments. However, this is not necessary as the update function $\mathcal{H}$ in (25) can always be made to store any finite number of percepts and previous hypotheses in the memory, i.e. as part h_k, because they are inputs to the update step (25). But also any historical action $y_{k'}$, $k' < k$ can be retrieved by first retrieving $h_{k'}$ from the memory and then using (25). This is possible because π is deterministic and can be simulated by $\mathcal{H}$.

1.7 Nonsequential and Batch Cases with States and Hypotheses

Just like in the framework using entire histories, also with the formulation based on states and hypotheses the situation simplifies a lot in the *nonsequential case*. Here, the environment has no memory at all so the conditioning factors in (18) and states are updated by i.i.d. sampling from the marginal distribution

$$\mathcal{S}(s_k) \tag{28}$$

Furthermore, observations o_k no longer depend on agent's last action as in (23) so they are sampled from

$$\mu_o(o_k|s_k) \tag{29}$$

Since s_k's are i.i.d., the o_k's are also i.i.d.

Rewards, given by (24), are however still generally non-i.i.d. as they depend on the agent's actions, which in turn depend on the evolving agent's hypothesis.

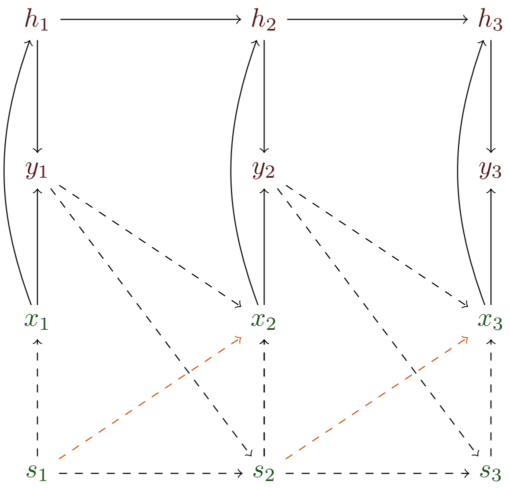


Figure 6: Influence diagram for states h_k, actions y_k and percepts x_k for $1 \leq k \leq 3$ with full lines indicating deterministic influences (via π and $\mathcal{H}$) and dashed lines showing probabilistic influences (via μ and $\mathcal{S}$). The highlighted dependencies are only needed for generating the reward part r_k of the percepts x_k.

Fig. 7 shows the complete set of influences in the nonsequential case.

A further simplification comes in the special *batch-learning* scenario of the nonsequential case. While in the learning phase of the latter, the agent uses the update rule (25), in the action phase it no longer updates the hypothesis, so

$$h_k = h_K, \forall k \geq K \tag{30}$$

This is illustrated in Fig. 8. Special attention is needed regarding the variables at time K. Reward r_K (part of percept x_K) is the last training reward, according to which the last update is conducted towards the final h_K. Observation o_K (another part of percept x_K) is, however, the first *testing* observation.

For $k > K$, y_{k-1} is fully determined by o_{k-1} and h_K through (26) in which $h_{k-1} = h_K$. So we can rewrite (24) into

$$\mu_r(r_k | s_{k-1}, o_{k-1}, h_K) \tag{31}$$

and further express

$$\mu_r(r_k|h_K) = \sum_{s_{k-1} \in O} \sum_{o_{k-1} \in O} \mu_r(r_k|h_K, s_{k-1}, o_{k-1}) \mu_o(o_{k-1}|s_{k-1}) \mathcal{S}(s_{k-1}) \tag{32}$$

where μ_o and $\mathcal{S}$, i.e. (29) and (28), are independent of k. So in the testing phase, rewards r_k are i.i.d. according to the distribution $\mu_r(r_k|h_K)$ depending only on

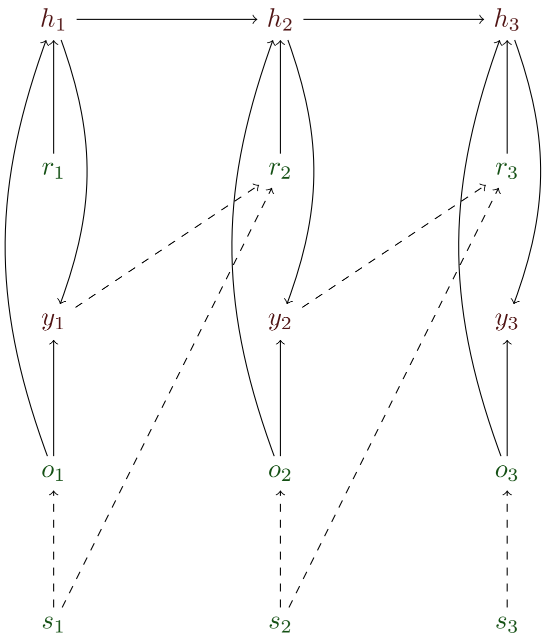


Figure 7: Influence diagram for hypothesis h_k, actions y_k, observations o_k, and rewards r_k for $1 \leq k \leq 3$ with full lines corresponding to deterministic influences (via π and $\mathcal{H}$) and dashed lines showing probabilistic influences (via μ and $\mathcal{S}$) in the nonsequential case.

the learned hypothesis h_K. This is analogical to the state-free formulation 12. Similarly to 13, an agent operating in the batch-learning scenario with states will be assessed by the expected reward in the testing phase

$$\sum_{r_k \in R} \mu_R(r_k|h_K) r_k \tag{33}$$

and should find a hypothesis h_K maximizing this quantity.

1.8 Prior Knowledge

- *Implicit*: the setting of H ("hard bias") and $\mathcal{H}$ ("soft bias")
- *Explicit*: the setting of h_1 ("background knowledge")

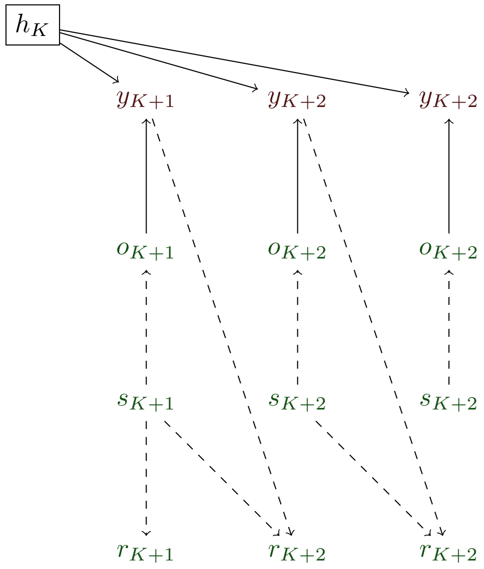


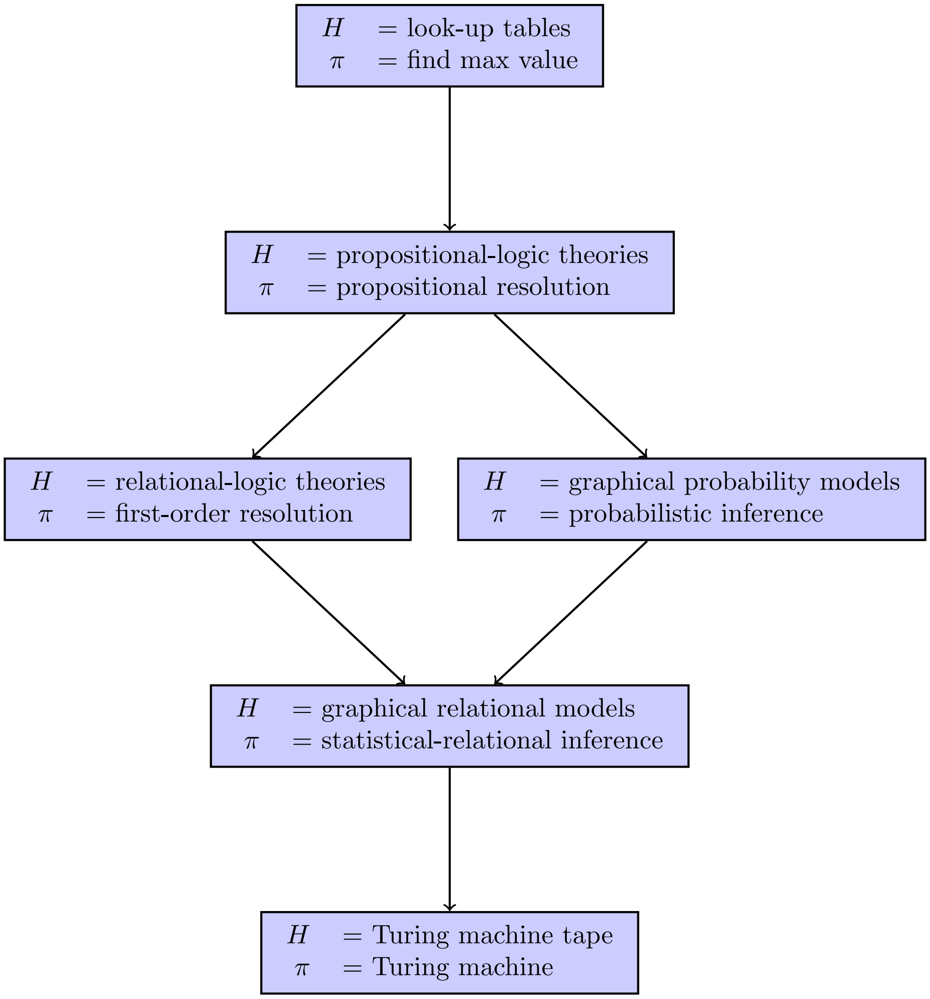
Figure 8: Influence diagram for actions y_k, observations o_k, states s_k, and rewards r_k in the action phase ($k > K$) of batch learning with full lines indicating deterministic influences (via π) and dashed lines showing probabilistic influences (via μ). The top row indicates the influence of the agent's last hypothesis learned in the learning phase on the action phase. The dependence of r_{K+1} on s_K and y_K is not shown.

1.9 Hypothesis Representations

See Fig. 9.

1.10 Learning Scenarios

1. on-line concept learning
2. batch concept learning
3. query-based and active learning
4. reinforcement learning
5. universal learning



H = look-up tables
π = find max value
H = propositional-logic theories
π = propositional resolution
H = relational-logic theories
π = first-order resolution
H = graphical probability models
π = probabilistic inference
H = graphical relational models
π = statistical-relational inference
H = Turing machine tape
π = Turing machine

Figure 9: Hypothesis representations and their corresponding policy classes (interpreters) considered in this course. Arrow directions indicate increasing expressiveness.

2 On-line Concept Learning

We implement the on-line concept learning scenario as a specific case of the general sequential learnig framework. The central assumption of concept learning is that the current observation uniquely determines the current state. Formally, this assumption restricts the state distribution obtained by inverting (18) according to the Bayes' rule.

$$\mu(s_k|o_k, y_{k-1}) = \frac{\mu_o(o_k|s_k, y_{k-1})\mu(s_k|y_{k-1})}{\mu_o(o_k|y_{k-1})} \tag{34}$$

Assume that function $c : O \to S$ exists such that (34) takes the specific form

$$\mu(s_k|o_k, y_{k-1}) = \begin{cases} 1 \text{ if } s_k = c(o_k) \\ 0 \text{ otherwise} \end{cases} \tag{35}$$

so function c, which is unknown to the agent, fully determines the current state according to the current observation and independently of the agent's actions. In other words, the observations are partitioned into *classes* co-inciding with states, and function c *classifies* the observations into these classes. The independence of y_{k-1} is an assumption similar to (29) made for the non-sequntial scenario. However, the above assumption is weaker than those of non-sequential learning. In particular, environment states can still depend on previous states as well as agent's actions through the observations o_k.

In the concept learning scenario we will work with two classes only, i.e.

$$S = \{ 0, 1 \} \tag{36}$$

Then function c can be conveniently identified with the subset of observations

$$\underline{\mathbf{c}} = \{ o \in O \ \mid \ c(o) = 1 \}$$

and write $o \in c$ or $c(o) = 1$ interchangeably. This subset view earns c the name *concept*.

Assume further that another function $L : S \times Y \to R$ exists such that (24) takes the specific form (incrementing the time index inconsequentially for shorter notation)

$$\mu_r(r_{k+1}|s_k, y_k) = \begin{cases} 1 \text{ if } r_{k+1} = -L(s_k, y_k) \\ 0 \text{ otherwise} \end{cases} \tag{37}$$

for $k > 1$, so the current reward is fully determined from the preceding state and action according to function L, which is called *loss*. The first reward r_1 is immaterial and is still sampled from the marginal $\mu_R(r_1)$.

In the concept learning scenario, we want the agent to learn the unknown concept by guessing the state. Thus we set

$$Y = S \tag{38}$$

and let the environment punish the agent with a negative reward for each incorrect guess y_k of the class s_k by by setting the loss as

$$L(s_k, y_k) = \begin{cases} 0 \text{ if } s_k = y_k \\ 1 \text{ otherwise} \end{cases} \tag{39}$$

since by (35), $s_k = c(o_k)$, and by (26), $y_k = \pi(h_k, o_k)$, the non-negative reward is awarded if an only if

$$c(o_k) = \pi(h_k, o_k) \tag{40}$$

that is, the agent hypothesis h_k in conjunction with the policy π that interprets it simulates the unknown concept c. Given (36), and assuming a fixed policy π, also hypothesis h_k can be formally identified with the set

$$\underline{\mathbf{h}}_k = \{\, o \in O \;\mid\; \pi(h_k, o) = 1 \,\}$$

so that

$$\underline{\mathbf{H}} = \{\, \underline{\mathbf{h}} \mid h \in H \,\} \tag{41}$$

and the fact that (40) holds for any observation o_k can be simply expressed as

$$\underline{\mathbf{c}} = \underline{\mathbf{h}}_k \tag{42}$$

which represents the basic goal of concept learning: by interaction with the environment, the agent should arrive at a hypothesis h_k exactly co-inciding with the target concept hidden in the environment.

The interaction is simple. At each step k, the environment samples a state s_k from (28). The state represents a *class*, which is either the concept c (if $s_k = 1$) or its complement $O \setminus c$ (if $s_k = 0$). Then it samples an observation o_k from the state according to (29) and provides it as an *example* of the class to the agent within percept $x_k = (o_k, r_k)$. The agent guesses the state (class) according to its current hypothesis by (26). If the guess was a mistake, reward $r_{k+1} = -1$ (39) is received in the next percept $x_{k+1} = (o_{k+1}, r_{k+1})$. Then, knowing the correct class of o_k due to (36), the agent updates its hypothesis by (25, replace k with $k+1$) and with this updated hypothesis, it guesses the class y_{k+1} of the next observation o_{k+1}. This cycle goes on indefinitely.

Whether or not the agent at some time k learns a hypothesis h_k satisfying (42) depends on the agent's update rule (25), and also on whether its hypothesis class[2] $\underline{\mathbf{H}}$ (27) contains such a $\underline{\mathbf{h}}_k$ at all. To formalize this latter condition, we will assume that the environmental concepts $\underline{\mathbf{c}}$ cannot be arbitrary but rather belong to a *concept class* $\underline{\mathbf{C}}$. An important property of the particular concept learning scenarios will be whether or not

$$\underline{\mathbf{C}} \subseteq \underline{\mathbf{H}} \tag{43}$$

[2] We take the liberty to call *hypothesis class* both H, i.e. the set of hypothesis representations, and $\underline{\mathbf{H}}$, i.e. the family of sets generated by the representations together with the fixed policy. The word *class* in the terms *hypothesis class* and *concept class* should not be confused with the classes of observations, which are states.

2.1 Conjunctive Concepts

Here we design an agent that learns an unknown conjunction by starting with the most specific hypothesis (a conjunction of all literals, i.e. all propositional variables as well as their negations) and then deleting all literals inconsistent with the received observations. So the initial hypothesis is gradually *generalized* towards the correct one. The main thing we will need to prove is that such deletions indeed lead to the correct hypothesis.

Observations are n-tuples of binary (truth) values

$$O = \{\, 0, 1 \,\}^n \tag{44}$$

The agent has the hypothesis class

$$H = \Phi \times O \tag{45}$$

where

$$\Phi = \left\{ \bigwedge_{i \in I} p_i \bigwedge_{j \in J} \neg p_j \;\middle|\; I, J \subseteq [1:n] \right\} \tag{46}$$

and $n \in N$. So

$$h_k = (\phi_k, o'_k) \tag{47}$$

consists of a conjunctive formula ϕ_k containing at most $2n$ literals, and $o'_k \in O$. The latter has the purpose of memorizing the last observation (example) provided by the environment and will be used only for updating hypotheses.

The formula ϕ_k is used to determine decisions through the agent's decision policy (26) $y_k = \pi(h_k, o_k) = \pi((\phi_k, o'_k), o_k)$. Whenever the policy does not depend directly on the memorized example o'_k, which will be the typical case, we will afford the shorter notation $\pi(\phi_k, o_k)$. The policy is set to

$$y_k = \pi(\phi_k, o_k) = \begin{cases} 1 \text{ if } o_k \models \phi_k \\ 0 \text{ otherwise} \end{cases} \tag{48}$$

where $o_k \models \phi_k$ means ϕ_k is true given the truth-value assignments o_i to variables p_i, $1 \leq i \leq n$. More precisely, we say that positive (negative, respectively) literal p_i ($\neg p_j$) is *consistent* with observation o_k if $o_k^i = 1$ ($o_k^i = 0$). Finally, $o_k \models \phi_k$ holds if and only if all literals of conjunction ϕ_k are consistent with o_k.

The update rule (25), which we expand by (4) and (47) to

$$(\phi_k, o'_k) = \mathcal{H}\left(\left(\phi_{k-1}, o'_{k-1}\right), (o_k, r_k)\right) \tag{49}$$

is set to

$$o'_k = o_k \tag{50}$$

$$\phi_k = \begin{cases} \phi_{k-1} \text{ if } r_k = 0 \\ \texttt{delete}(\phi_{k-1}, o'_{k-1}) \text{ otherwise} \end{cases} \tag{51}$$

where

$$\texttt{delete}\left(\bigwedge_{i \in I} p_i \bigwedge_{j \in J} \neg p_j, (o^1, o^2, \ldots, o^n) \right) = \tag{52}$$

$$\bigwedge_{\substack{i \in I \\ o^i = 1}} p_i \bigwedge_{\substack{i \in I \\ o^j = 0}} \neg p_j \tag{53}$$

So the `delete` function keeps exactly those literals from ϕ_{k-1} which are consistent with o'_{k-1}.

We assume that (43) holds. In particular, there exists a target conjunction $\phi^* \in \Phi$ such that $h^* = (\phi^*, o)$ exactly simulating the unknown concept c, i.e.

$$s_k = c(o_k) = \pi(\phi^*_k, o_k) \tag{54}$$

Lemma 2.1 $s_k = 1$ *if and only if all literals of* ϕ^* *are consistent with* o_k*.*

The above lemma follows directly from (48) and (54).

Lemma 2.2 *Whenever* `delete`(ϕ_{k-1}, o'_{k-1}) *is called,* $s_{k-1} \neq y_{k-1}$*, and if* $s_{k-1} = 0$*, then all literals of* ϕ_{k-1} *are consistent with* o'_{k-1}*.*

To see why Lemma 2.2 is true, note that according to (51), $r_k \neq 0$ when `delete` is called. Due to (37) and (39), this means that $s_{k-1} \neq y_{k-1}$. So if $s_{k-1} = 0$ then $y_{k-1} = 1$, but then due to (48), $o'_{k-1} \models \phi_{k-1}$ and so all literals of ϕ_{k-1} are indeed consistent with o'_{k-1}.

Lemma 2.3 `delete`(ϕ_{k-1}, o'_{k-1}) *never removes a literal* $l \in \phi_{k-1}$ *which is also in* ϕ^**.*

Assume for contradiction that it removes a literal $l \in \phi^*$. First assume $s_{k-1} = 0$. By Lemma 2.2, all literals of ϕ_{k-1} are consistent with o'_{k-1}. But because `delete`(ϕ_{k-1}, o'_{k-1}) keeps all literals of ϕ_{k-1} consistent with o'_{k-1}, it does not delete l, which is a contradiction. Now consider $s_{k-1} = 1$. Then by Lemma 2.1 all literals of ϕ^* including l must be consistent with o'_{k-1}. Again, since `delete` keeps all consistent literals, it does not delete l, which is a contradiction.

The starting hypothesis of the designed agent is set to contain all possible literals

$$\phi_1 = p_1 \wedge \neg p_1 \wedge p_2 \wedge \neg p_2 \wedge \ldots p_n \wedge \neg p_n \tag{55}$$

Thus $\phi_1 \supseteq \phi^*$, where the inclusion is with respect to the sets of literals in ϕ_1 and ϕ^*. Furthermore, due to Lemma 2.3, we have

$$\phi_k \supseteq \phi^*, \ k \in N \tag{56}$$

Given the above, the agent makes mistakes only on 'positive examples', and the mistakes are corrected by removing at least one inconsistent literal, as the following lemma formalizes.

Lemma 2.4 *Assuming (55), whenever* `delete`(ϕ_{k-1}, o'_{k-1}) *is called,* $s_{k-1} = 1$, *and the function deletes at least one literal from* ϕ_{k-1}.

Due to Lemma 2.2, $s_{k-1} \neq y_{k-1}$. If $s_{k-1} = 0$ and $y_{k-1} = 1$ then by the same lemma, all literals of ϕ_{k-1} are consistent with o'_{k-1}. According to Lemma 2.1, there would then be a literal in ϕ^* inconsistent with o'_{k-1}. But due to (56), this inconsistent literal would also be contained in ϕ_{k-1}, which is a contradiction. So we know that $s_{k-1} = 1$ and $y_{k-1} = 0$. According to (48), this means that ϕ_{k-1} contains a literal inconsistent with o'_{k-1}. Since `delete`, by definition, keeps exactly all consistent literals, the inconsistent literal is removed.

Theorem 2.5 *The agent makes at most* $2n$ *mistakes, i.e. the cumulative reward is*

$$\sum_{k=1}^{m} r_k \geq -2n \tag{57}$$

for an arbitrary horizon $m \in N$.

Since the first agent's conjunction has $2n$ literals by (55) and upon each mistake, at least one literal is removed from from the conjunction by Lemma 2.4, the maximum number of mistakes is $2n$.

2.2 Disjunctive Concepts

Here we will build an agent for the logical 'counterpart' of the former agent. As we will see later, disjunctions can be learned by the conjunctive agent explained previously with just a small modification, so in principle we have disjuctions already covered. However, the point of this section is to design a completely different strategy which will have different properties compared to the generalization strategy of the conjunctive agent.

Observations are as in (44).

Concept class

$$C = \{\, c_\phi \mid \phi \in \Phi \,\} \tag{58}$$

where for $s \leq n$

$$\Phi = \{\, p_{i_1} \vee p_{i_2} \vee \ldots \vee p_{i_s} \mid 1 \leq i_1, i_2, \ldots i_s \leq n \,\} \tag{59}$$

and

$$c_\phi(o) = \begin{cases} 1 \text{ if } o \models \phi \\ 0 \text{ otherwise} \end{cases} \tag{60}$$

Although (59) considers only *monotone* disjuctions, i.e. without negated literals, it can be easily generalized to general disjunctions by introducing $2s$ (instead of s) propositional variables $p'_i = p_i$, $p'_{2i} = \neg p_i$.

$$H = [0, 1, 2, \ldots, q]^n \times O \tag{61}$$

where $q \in N$, O is again memory for the last observation, and

$$h_k = (w_k, o'_k) \tag{62}$$

where $w_k = (w_k^1, w_k^2, \ldots w_k^n)$

Decision policy

$$y_k = \pi(w_k, o_k) = \begin{cases} 1 \text{ if } w_k \cdot o_k > n/2 \\ 0 \text{ otherwise} \end{cases} \tag{63}$$

Assume again that $\underline{\mathbf{C}} \subseteq \underline{\mathbf{H}}$. This can be achieved with a sufficiently large q as disjunctions are linearly separable.

$$w_1 = (1, 1, \ldots 1) \tag{64}$$

Hypothesis update

$$(w_k, o'_k) = \mathcal{H}\left(\left(w_{k-1}, o'_{k-1}\right), (o_k, r_k)\right) \tag{65}$$

$$o'_k = o_k \tag{66}$$

$$w_k = \begin{cases} w_{k-1} \text{ if } r_k = 0 \\ \texttt{update}(2, w_{k-1}, o'_{k-1}) \text{ if } w_{k-1} \cdot o'_{k-1} \leq n/2 \\ \texttt{update}(0, w_{k-1}, o'_{k-1}) \text{ if } w_{k-1} \cdot o'_{k-1} > n/2 \end{cases} \tag{67}$$

where the function `update` is defined such that for $w_k = \texttt{update}(\theta, w, o)$

$$w_k^i = \begin{cases} \theta \cdot w^i \text{ if } o^i = 1 \\ w^i \text{ otherwise} \end{cases} \tag{68}$$

Theorem 2.6 *The agent makes at most* $2 + 2s \lg n$ *mistakes, , i.e. the cumulative reward is*

$$\sum_{k=1}^{m} r_k \geq -2 - 2s \lg n \tag{69}$$

for any horizon $m \in N$*.*

(proof omitted)

ref to perceptrons

2.3 Further Concept Classes

The agent that learns conjunctions as explained in Section 2.1 can be also made to learn disjuctions due to

$$\neg\,(p_1 \vee p_2 \vee \ldots \vee p_n) = \neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \tag{70}$$

So the only required change is that the agent replaces observations o_k with $\overline{o_k} = (1 - o_k^1, 1 - o_k^2, \ldots, 1 - o_k^n)$ and its actions y_k with $1 - y_k$.

By the same principle, the disjuction-learning agent from Section 2.2 can learn conjunctions.

The difference is in the mistake bound. The latter agent performs better when the number of variables n is larger than the number of relevant variables s.

Other logical classes can also be reduced to conjunction and disjunction learning. Consider e.g. s-CNF ($s < \infty$). These are conjunctions of s-clauses. An s-clause is a disjunction of at most s-literals. There is a finite number of s-clauses so the agent can simply establish one new propositional variable for each possible s-clause a learn a conjunction with these new variables. This reduction would even be efficient if s is a small constant. Indeed, if n is the number of original variables, then the number of possible clauses is $\binom{n}{s}$ which grows polynomially with n. A similar reduction can be used to learn s-DNF.

2.4 General Concept Classes

How well can we do with *arbitrary* concept classes? Immediate mistake bound for any concept class C

$$|C| - 1 \tag{71}$$

Can be improved to $\lg |C|$ using the *version space* strategy.

Assume a set Φ of *versions*. These may be conjunctions, disjuctions, or other representations. The only assumption is that each version $\phi \in \Phi$ provides a decision $\phi(o)$ for any observation $o \in O$. So this function works similarly to a decision policy π, however, the plan for the version-space agent is to construct π that uses multiple versions for a single decision.

The hypothesis class is

$$H = 2^{\Phi} \times O \tag{72}$$

so

$$h_k = (V, o) \tag{73}$$

where V is a set ('space') of versions, and o again stores the last observation. The plan is that V maintains all versions from Φ consistent with the observations and rewards received so far.

Decisions are determined by voting of all versions in the current version space

$$y_k = \pi(V_k, o_k) = \begin{cases} 1 \text{ if } |\{\, \phi \in V_k \mid \phi(o_k) = 1 \,\}| > |V_k|/2 \\ 0 \text{ otherwise} \end{cases} \tag{74}$$

The initial version space contains all versions from Φ

$$V_1 = \Phi \tag{75}$$

Update step

$$o'_k = o_k \tag{76}$$

$$V_k = \{\, \phi \in V_{k-1} \mid \phi(o_{k-1}) = s_{k-1} \,\} \tag{77}$$

where s_{k-1} is determined as $s_{k-1} = |y_{k-1} - r_{k-1}|$ (check that this is true) and $y_{k-1} = \pi(V_{k-1}, o'_{k-1})$.

Assume that Φ is rich enough so that it contains $\phi \in \Phi$ so that $\phi(o) = c(o)$ for all $o \in O$ (check that this implies 43). Then the following holds.

Theorem 2.7 *The agent makes at most* $\lg|\Phi|$ *mistakes, i.e. the cumulative reward is*

$$\sum_{k=1}^{m} r_k \geq -\lg|\Phi| \tag{78}$$

for any horizon $m \in N$.

To see why the theorem holds note that the agent decides by the majority of current versions. So if a mistake is made, at least half of the versions are deleted. In the worst case, the last remaining version is correct.

The logaritmic bound is good but the computational demands for storing the version space are prohibitive.

What about a *lower bound* on mistakes? We say that a set of observations $O' \subseteq O$ is *shattered* by hypothesis class $\underline{\mathbf{H}}$ if

$$\{\, O' \cap \underline{\mathbf{h}} \mid \underline{\mathbf{h}} \in \underline{\mathbf{H}} \,\} = 2^{O'} \tag{79}$$

which means that the set of observations can be partitioned in all possible ways into two classes by the hypotheses from $\underline{\mathbf{H}}$.

The *Vapnik-Chervonenkis Dimension* (or VC-dimension) of $\underline{\mathbf{H}}$, written $\mathsf{VC}(\underline{\mathbf{H}})$, is the cardinality of the largest set $O' \subseteq O$ that ca be shattered by $\underline{\mathbf{H}}$. The definition extends formally also to H corresponding to $\underline{\mathbf{H}}$ by (41), so we will also write $\mathsf{VC}(H)$.

Theorem 2.8 *No upper bound on the number of mistakes made by an agent in the concept-learning scenario using hypothesis space* H *is smaller than* $\mathsf{VC}(H)$.

This is because for any sequence of agent's decisions $y_1, y_2, \ldots, y_{\mathsf{VC}(H)}$ there exists a $h \in H$ according to which all these decisions are wrong.

3 Batch Concept Learning

The batch concept learning situation is defined by the assumptions of batch learning (Section 1.7) combined with the concept-learning requisities which are the same as in on-line concept learning (Section 2). In particular, the latter include the assumption of a target concept determining states from observations (35), the binary range of states (36), and the unit loss function (39) determining rewards (37).

Since rewards are negative losses by (37), the expected reward (33) to be maximized is in $[-1; 0]$. Its negative value, for a given hypothesis and $k > K$, is called the *error* of the hypothesis

$$\mathrm{err}(h_K) = -\sum_{r_k \in R} \mu_R(r_k|h_K) r_k \tag{80}$$

and corresponds to the proportion of misclassified observations in the testing phase, i.e. those observations o_k $(k \geq K)$ for which $y_k \neq s_k$. Given (39) and (80), the error can be expressed as the probability of making a mistake, i.e. receiving a -1 reward

$$\mathrm{err}(h_K) = \mu_R(-1|h_K) \tag{81}$$

A natural question of interest is how the algorithms we designed for on-line concept learning would perform in the batch concept learning framework. Evidently, the bounds on the number of mistakes we established in Theorems 2.5, 2.6, and 2.7 do not translate to any bound on $\mathrm{err}(h_K)$ as there is no guarantee that the mistakes will happen in the learning phase $(k \leq K)$ where the agent still can fix its hypothesis.

Unlike in the on-line learning case, the batch case inherits the non-sequential assumptions (28) and (29), meaning that states and observations are sampled i.i.d. according to distributions that do not change with k. They prevent the environment from ‘adversarial’ behavior, for example, one where the training phase would only contain easy examples and the hard ones would be kept for the testing phase.

3.1 Conjunctive Concepts

3.2 Disjunctive Concepts

3.3 Further Concept Classes

3.4 General Concept Classes