2.Adaptive Dynamic Programming

- If the transition probabilities and the rewards of the states are known (which will usually happen after a reasonably small set of training examples), then the actual utilities can be computed directly as
  - $U(i) = R(i) + SUM_j M_{ij}U(j)$

  where *U(i)* is the utility of state *i*, *R* is its reward, and M_{ij} is the probability of transition from state *i* to state *j*

- This is identical to a single *value determination* in the policy iteration algorithm for Markov decision processes.
- *Adaptive dynamic programming* is any kind of reinforcement learning method that works by solving the utility equations using a dynamic programming algorithm.
- It is exact, but of course highly inefficient in large state spaces

3. Temporal Difference Learning

- uses the difference in utility values between successive states to adjust them from one epoch to another
- key idea is to use the observed transitions to adjust the values of the observed states so that they agree with the ADP constraint equations
- Practically, this means updating the utility of state *i*

- This is done with the *temporal-difference* (TD) equation:
  - U(i) <- U(i) + a(R(i) + U(j) - U(i))
  - where *a* is a *learning rate* parameter

  Temporal difference learning is a way of approximating the ADP constraint equations without solving them for all possible states

- The idea generally is to define conditions that hold over local transitions when the utility estimates are correct, and then create update rules that nudge the estimates toward this equation.
- This approach will cause *U(i)* to converge to the correct value if the learning rate parameter decreases with the number of times a state has been visited [Dayan, 1992].
- In general, as the number of training sequences tends to infinity, TD will converge on the same utilities as

Passive Learning in an Unknown Environment

- neither temporal difference learning nor LMS actually use the model *M* of state transition probabilities
- they will operate unchanged in an unknown environment
- The ADP approach, however, updates its estimated model of an unknown environment

- Any method for learning stochastic functions can be used to learn the environment model;
- in particular, in a simple environment the transition probability M_{ij} is just the percentage of times state *i* has transitioned to *j*

Basic difference between TD and ADP:

- TD adjusts a state to agree with the observed successor, while ADP makes a state agree with all successors that might occur, weighted by their probabilities
- ADP's adjustments may need to be propagated across all of the utility equations, while TD's affect only the current equation.

- A middle-ground can be found by bounding or ordering the number of adjustments made in ADP, beyond the simple one made in TD
- The *prioritized-sweeping* heuristic prefers only to make adjustments to states whose *likely* successors have just undergone *large* adjustments in their utility estimates
- Such approximate ADP systems can be very nearly as efficient as ADP in terms of

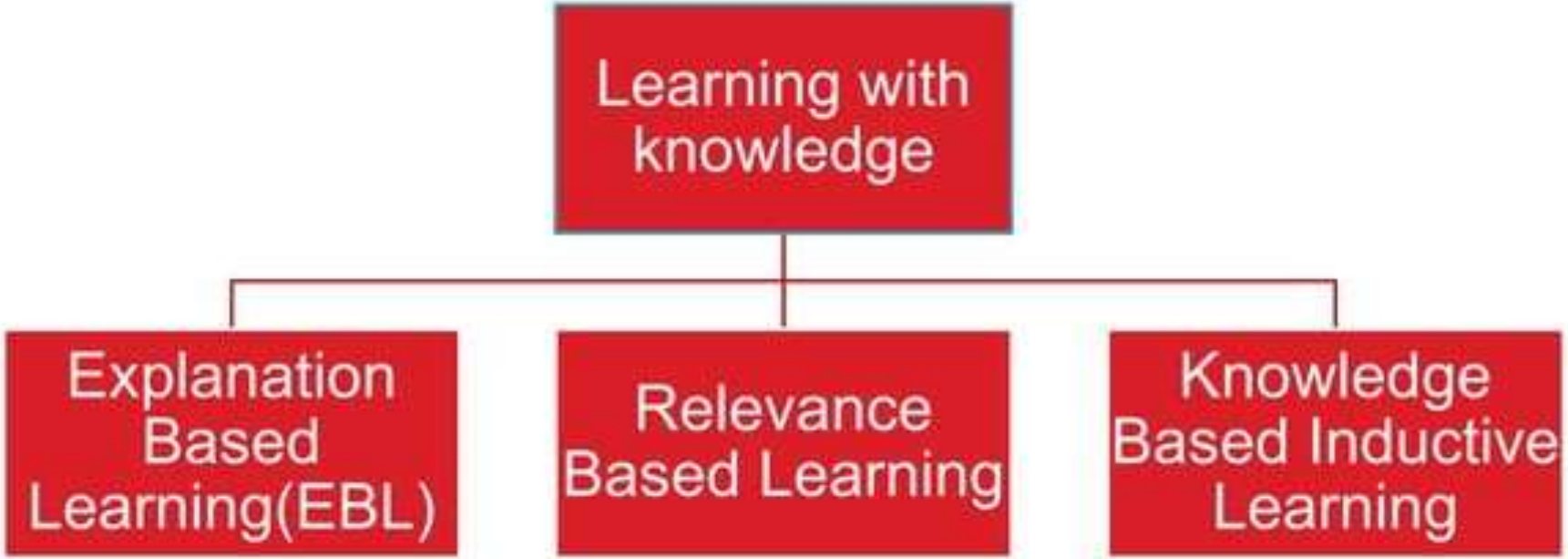
Active Learning in an Unknown Environment

- difference between active and passive agents is that **passive agents learn a fixed policy**, while the **active agent must decide what action to take and how it will affect its rewards**
- To represent an active agent, the environment model *M* is extended to give the probability of a transition from a state *i* to a state *j*, *given an action*

- Utility is modified to be the reward of the state plus the maximum utility expected depending upon the agent's action:
  - $U(i) = R(i) + \max_a \times SUM_j M^a_{ij} U(j)$
- An ADP agent is extended to learn transition probabilities given actions; this is simply another dimension in its transition table
- A TD agent must similarly be extended to have a model of the environment.

Learning with Knowledge

Learning with knowledge : Tree



Learning with knowledge

- considering the kinds of logical constraints placed upon different kinds of knowledge-based learning, we can classify them more clearly
- Examples are composed of *Descriptions* and *Classifications*, and we are trying to find a *Hypothesis* to explain the data

- Inductive learning can be characterized by the following *entailment constraint*:
  - Hypothesis ^ Descriptions |= Classifications
- given our hypothesis and descriptions of problem instances, we want to generate classifications
- This is inductive learning

Other kinds of learning that use prior knowledge are:

1) Explanation based learning (EBL)
2) Relevance based learning
3) Knowledge based inductive learning

1) Explanation based learning(EBL)

- this kind of learning occurs when the system finds an explanation of an instance it has seen, and generalizes the explanation
- The general rule follows logically from the background knowledge possessed by the system
- The entailment constraints for EBL are
  - Hypothesis ^ Descriptions |= Classification
  - Background |= Hypothesis

- agent does not actually learn anything factually new, since the hypothesis was entailed by background knowledge
- This kind of learning is regarded as a way to convert first principles into useful specialized knowledge (converting problem-solving search into pattern-matching search)

- basic idea is to construct an explanation of the observed result, and then generalize the explanation
- More specifically, while constructing a proof of the solution, a parallel proof is performed, in which each constant of the first is made into a variable
- Then a new rule is built in which the left-hand side is the leaves of the proof tree, and the right-hand side is the variabilized goal, up to any bindings

- Any conditions true regardless of the variables are dropped
- Note that by pruning the tree before the leaves, even more general rules may be learned
- However, the more general, the more computation may be required to apply the rule
- One approach is to require the *operationality* of the subgoals in the new rule -- that they be "easy" to solve

2) Relevance Based Learning

- This is a kind of learning in which background knowledge relates the *relevance* of a set of features in an instance to the general goal predicate
- For example, if I see men in the Forum in Rome speaking Latin, and I know that if seeing someone in a city speaking a language usually means all people in the city speak that language, I can

- In general, background knowledge, together with the observations, allows the agent to form a new, general rule to explain the observations
- The entailment constraint for RBL is
  - Hypothesis ^ Descriptions |= Classifications
  - Background ^ Descriptions ^ Classifications |= Hypothesis

- This is a deductive form of learning, because it cannot produce hypotheses that go beyond the background knowledge and observations
- We presume that our knowledge base has a set of *functional dependencies* or *determiners* that support the construction of hypotheses
- The learning algorithm then tries to find the minimal consistent determination (e.g., a sentence of the form "P determines Q," meaning that if the examples match on P they match on Q)

3) Knowledge based inductive learning

- This is a kind of learning in which our background knowledge, together with our observations, lead us to make a hypothesis that explains the examples we see
- If I see the Old Man from Scene 24 on the Bridge of Despair, and notice that he asks a simple question of every *other* knight that attempts to cross, I can hypothesize that only the odd-

- The entailment constraint in this case is
  - Background ^ Hypothesis ^ Descriptions |= Classifications
- Such knowledge-based inductive learning has been studied mainly in the field of *inductive logic programming*

- Such systems reduce learning complexity in two ways
  - First, by requiring all new hypotheses to be consistent with existing knowledge, they reduce the search space of hypotheses
  - Secondly, the more prior knowledge available, the less new knowledge required in the hypothesis to explain the observations

- Attribute-based learning algorithms are incapable of learning predicates
- One of the advantages of ILP algorithms is their much broader range of applicability

Instance Based Learning (IBL)

Background

- Storing and using specific instances improves the performance of several supervised learning algorithm
- Include algorithms that learn decision trees, classification rules, and distributed networks
- IBL algorithms are derived from the nearest neighbor pattern classifier

Instance based learning

- generates classification predictions using only specific instances
- do *not maintain a set of abstractions* derived from specific instances
- This approach extends the nearest neighbor algorithm, which has large storage requirements
- storage requirements can be significantly reduced with, at most, minor sacrifices in learning rate and classification accuracy

- While the storage-reducing algorithm performs well on several real world databases, its performance degrades rapidly with the level of attribute noise in training instances
- save and use only selected instances to generate classification predictions

Using specific instances in supervised learning algorithms

- decreases the costs incurred
- when updating concept descriptions, increases learning rates,
- allows for the representation of probabilistic concept descriptions,
- and focuses theory-based reasoning in real-world applications

Instance-based learning algorithms suffer from several problems

- they are computationally expensive classifiers since they save all training instances,
- they are intolerant of attribute noise,
- they are intolerant of irrelevant attributes,
- they are sensitive to the choice of the algorithm's similarity function,
- there is no natural way to work with nominal-valued attributes or missing attributes, and
- they provide little usable information regarding the

Overview of IBL

- Learning task : supervised learning or *learning from examples*
- Only input is a sequence of *instances*
- Each instance is assumed to be represented by a set of attribute-value pairs (?? Next slide)
- All instances are assumed to be described by the same set of n attributes, although this restriction is not required by the paradigm itself (Aha, 1989c) and missing attribute values are tolerated

What are attribute-value pairs?

- An action-value function assigns an expected utility to the result of performing a given action in a given state
- If *Q(a, i)* is the value of doing action *a* in state *i*, then
  - $U(i) = \max_a Q(a, i)$
- The equations for Q-learning are similar to those

- The difference is that Q-learning agents do not need models of the world. The equilibrium equation, which can be used directly (as with ADP agents) is
  - $Q(a, i) = R(i) + \text{SUM}_j M^a_{ij} \max_{a'} Q(a', j)$
- The temporal difference version does not require that a model be learned; its update equation is

About attributes

- set of attributes defines an n-dimensional *instance space*
- *Exactly* one of these attributes corresponds to the *category attribute;*
- *the other attributes are predictor attributes*
- *A category is the set of all instances in an instance space that have the same* value for their category attribute

IBL

- IBL algorithms can learn multiple, possibly overlapping concept descriptions simultaneously
- primary output of IBL algorithms is a *concept description (or concept)*
- This is a function that maps instances to categories: given an instance drawn from the instance space, it yields *a classification, which is the predicted value for this instance's category*

- An instance-based concept description includes a set of stored instances and, possibly, some information concerning their past performances during classification
- e.g., their number of correct and incorrect classification predictions
- This set of instances can change after each

- However, IBL algorithms do not construct extensional concept descriptions
- Instead, concept descriptions are determined by how the IBL algorithm's selected *similarity and classification functions use the current set of saved instances*

IBL framework components

- ***Similarity Function:***
  - *This computes the similarity between a training instance i and the* instances in the concept description
  - Similarities are numeric-valued

- ***Classification Function:***
  - *This receives the similarity function's results and the classification performance records of the instances in the concept description*
  - *It yields a classification for i*

- ***Concept Description Updater:***
  - *This maintains records on classification performance and* decides which instances to include in the concept description
  - Inputs include i, the similarity results, the classification results, and a current concept description

- The similarity and classification functions determine how the set of saved instances in the concept description are used to predict values for the category attribute
- Therefore, IBL concept descriptions not only contain a set of instances, but also include these two functions.

- IBL algorithms assume that *similar instances have similar classifications*
- *This leads* to their local bias for classifying novel instances according to their most similar neighbor's classification
- IBL algorithms also assume that, without prior knowledge, attributes will have equal relevance for classification decisions (i.e., by having equal weight in the similarity function)

Summary

- IBL algorithms differ from most other supervised learning methods:
- they don't construct explicit abstractions such as decision trees or rules
- Most learning algorithms derive generalizations from instances when they are presented and use simple matching procedures to classify subsequently presented instances

Performance Dimensions

- *1) Generality: This is the class of concepts which are describable by the representation* and learnable by the algorithm
  - We will show that IBL algorithms can pac-learn (Valiant, 1984) any concept whose boundary is a union of a finite number of closed hyper-curves of finite size
- *2) Accuracy: This is the concept descriptions'*

- *3) Learning Rate: This is the speed at which classification accuracy increases during training*
  - It is a more useful indicator of the performance of the learning algorithm than is accuracy for finite-sized training sets
- *4) Incorporation Costs: These are incurred while updating the concept descriptions with* a single training instance
  - They include classification costs

IBL algorithm

Table 1. The IB1 algorithm (*CD* = Concept Description).

$CD \leftarrow \emptyset$

for each $x \in$ Training Set **do**

1. **for each** $y \in CD$ **do**
   Sim[y] $\leftarrow$ Similarity(x, y)
2. $y_{max} \leftarrow$ some $y \in CD$ with maximal Sim[y]
3. **if** class(x) = class(y_{max})
   then classification $\leftarrow$ **correct**
   else classification $\leftarrow$ **incorrect**
4. $CD \leftarrow CD \cup \{x\}$

Figure 1. The extension of IB1's concept description, denoted by the solid lines, improves with training. Dashed lines delineate the four disjuncts of the target concept. Positive (+) and negative (−) instances are shown where possible.

THANK YOU