Hierarchical Functional Category Learning for Efficient Value Function Approximation in Object-Based Environments

by

Yongjia Wang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2011

This work is dedicated to my wife Yue (Joy), and daughter QiQi (Iris)

Acknowledgments

First of all, I would like to thank my advisor John Laird. John is instrumental in helping me turn abstract ideas into concrete research plans and always encourages me to explore different approaches. Meanwhile, his knowledge and advice has helped me avoid many potentially time consuming distractions. No matter how frustrated I was, I always feel reassured and enlightened after every meeting with John. He has also guided me to realize the importance of turning research work into well written technical documents. Writing paper with John is a pleasant learning experience, as he always provides thorough feedbacks, which is especially valuable to improve my academic writing skill.

I would like to thank my dissertation committee members. Their influence is beyond steering my dissertation into the right direction. When I first entered the CSE program, Satinder Singh taught the introductory and advanced AI classes that grabbed my initial interests and led me to the AI track. Thad Polk is an excellent teacher. His computational cognitive modeling class (Psych 644) provided the inspirations that shaped my research path. Rick Lewis advised me, after my thesis proposal, to focus on the most interesting direction of utility driven category learning, which turned out to be a critical decision. The conversations with Ben Kuipers' have reminded me to always think about my research in the bigger context, and to increase the impact of my research on a broader community.

I would like to thank Fan Meng, who was my advisor in the Bioinformatics program. The experience in his team provided solid foundations for my transition to the computational science. I would also like to thank H. V. Jagadish, who is the key person that led me to the CSE program. In addition, being the GSI for Jag's database class is one of the most valuable experiences in my graduate life.

I would like to thank the CSE faculty, staff and fellow graduate students, who altogether cultivated this rewarding academic environment. I am especially thankful to the students in the Soar group – Andy, Bob, Joseph, Nate, Nick, Sam, Scott, Shelly, and everyone else who has shared the same office with me.

Finally, and most importantly, I would like to thank my family. My wife, Yue, has always supported me with tremendous love. I could not have gone this far without her accompany. Her contagiously joyful personality brightens even the dullest days, which fills me with the energy to get through difficult times. Our energetic daughter, Iris, inherited her mother’s joyful nature, constantly brings laughter to the family. Spending time with Iris, I rarely have the chance to feel stressful, even during the most stressful time. I would also like to thank my parents and parent-in-laws for repeatedly reminding me “are you done yet?”, and “when are you going to find a job?” directly or indirectly. My father, who is an engineer, cultivated my early interest towards science and technology. I am especially thankful to my mother, my mother-in-law and father-in-law for helping us take care of Iris - I swear this is no easy job as it may sound.

Table of Contents

List of Figures

List of Tables

Abstract

Creating autonomous long-lived agent that can robustly function in a complex object-based environment has been a persistent goal in the field of artificial intelligence. Learning the appropriate functional categories of objects is one of the keys to achieve this goal, and is the theme of this thesis.

We formulate the research problem as finding efficient value function approximation algorithms, where the input to the function is an object-based state representation, and output of the function is the utility value of that input state. The challenges arise from the requirements of efficient learning, and incremental learning of complex nonlinear value functions, whose input consists of diverse objects in high dimensional feature space. Our solutions are based on three key principles. The first is that the value function representation can usefully exploit the compositional structure of object-based environments, where the state representations consist of independent objects with their own perceptual features and functional properties. The second is that hierarchical symbolic category representations, inspired by human cognitive models, can help achieve efficient learning. The third is that the object categorization criteria must be consistent and coherent with the target utility value function. We provide two implementations based on these key principles, with evaluations both based on functionality and on cognitive plausibility.

Traditionally, category learning and value function approximation are studied as separate problems. The thesis presents a unique synthesis of the two. On one hand, it provides efficient value function approximation algorithms that can take advantage of compact representational basis adaptively generated by hierarchical category learning. On the other hand, it provides a utility based category learning model that offers new computational insights to human category learning behaviors.

Chapter 1

Introduction

Manipulating objects is a significant part of our everyday life, and object category learning is believed to be an important cognitive capability for general human intelligence. In cognitive psychology, the study of category learning has been a prominent area. Many theories and cognitive models of category learning have been developed, including: rule-based theories (Rouder & Ratcliff, 2006), exemplar theories (Medin & Schaffer, 1978), and the prototype theory (Rosch, 1978).

Creating autonomous and robust agents with the capability of adapting to novel complex environments has been a persistent goal for the artificial intelligence (AI) community. Object category learning is one of the keys to achieve this goal. As pointed out by Russell & Norvig (2009), "*although interactions with the world takes place at the level of individual objects, much reasoning takes part at the level of categories*".

In the machine learning field, various forms of category learning problems have been formulated, and a variety of learning paradigms, techniques and algorithms have been developed. One of the most straight-forward formulations is the direct supervised classification problem, where the desired outputs are symbolic labels explicitly provided to the learner. In many real-world applications, the class labels are not directly available. In such situations, latent class models (Clog, 1995) or unsupervised clustering algorithms (Jolion *et al*., 1991; Frigui & Krishnapuram, 1997) are applicable.

However, object category learning in the context of a long-lived, online learning agent has been a relatively unexplored area. In such environments, the agent must learn to deal with diverse types of objects, incrementally form new categories without externally provided category labels, and must response quickly to the external environment. Such problems involve multiple challenges that have traditionally been tackled separately from different perspectives, with different assumptions, techniques, methodologies and evaluation frameworks.

In the scenario of reactive online learning agent, the tasks are often studied under the reinforcement learning (RL) framework (Sutton & Barto, 1998). Traditional reinforcement learning techniques have focused on control problems and relied on general propositional state representations. Relational reinforcement learning (RRL) (Džeroski *et al.*, 2001) uses more expressive object-based relational representations, which are shown to have advantages over traditional propositional representations for object-based environments (Duik et al., 2008; Walsh, 2010). Figure 1.1 listed several examples of traditional control problems and object-based environments, which have been studied within the RL framework. However, the work on RRL has emphasized

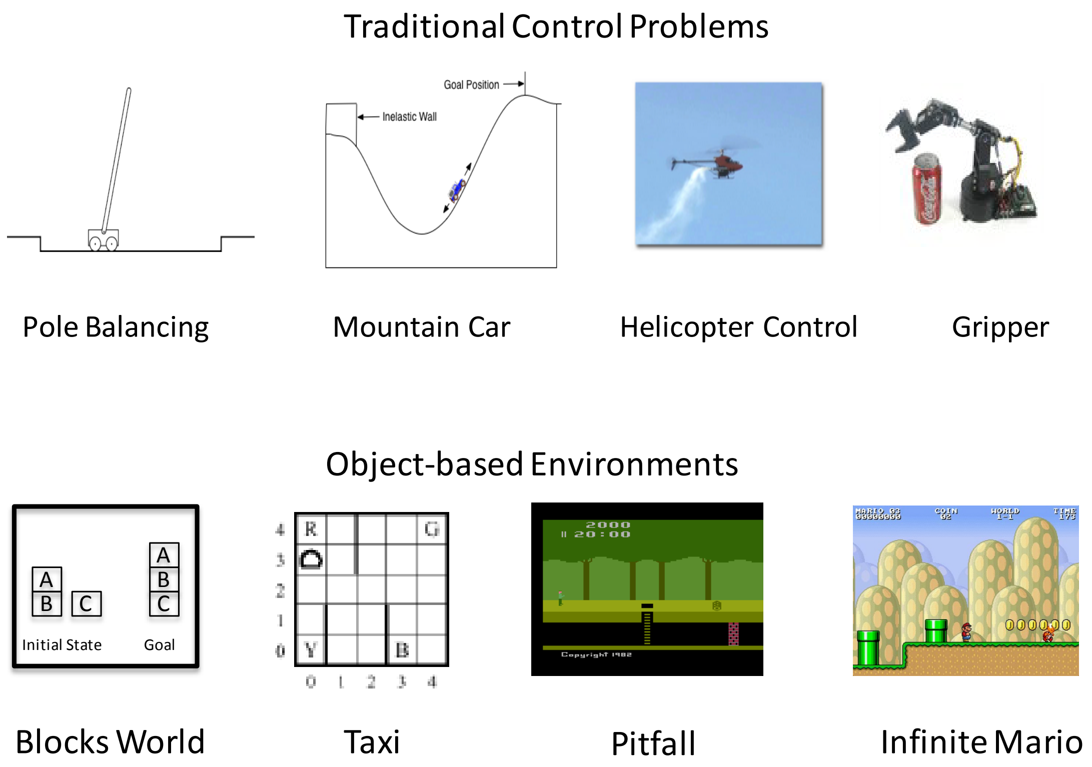


Figure 1.1: Examples of RL domains.

The top 4 are traditional control problems, and the bottom 4 are in object-based environments.

learning action models, such as the effect of colliding into a wall (in Taxi domain), or jumping over a monster (in Pitfall and Infinite Mario). These evaluation tasks often make simplifying assumptions about the diversity of objects and the origination of object categories. The requirement of forming new object categories is not addressed in those approaches.

In the field of robotics, it is sometimes required that symbolic representation of objects is built up from grounded primitive sensory inputs. For example, when acting in an environment populated with objects, it is desirable that the agent can develop perceptually grounded ontology of objects (Modayil & Kuipers, 2007). However, the current immediate challenge in this area is about the spatial aspect, not the object aspect. For example, in typical robotics tasks, the main challenges include: construct the maps (Durrant-Whyte & Bailey, 2006; Kuipers *et al.*, 2004), perform path planning and motion planning to avoid and maneuver around obstacles, identify a target object and manipulate it with the right motor controls. The diversity and complexity of objects are usually limited.

1.1 Challenge of Object Diversity

The practical reason for the attention to spatial problems is that space is always the immediate challenge while object diversity is secondary. For example, in the taxi domain shown in Figure 1.1, the interacting objects include the walls, passengers and possibly other taxis. Navigation is a major component of the domain. Both Pitfall and Infinite Mario have a larger collection of objects such as various kinds of obstacles, enemies and treasure. However, the primary challenge is still about how to spatially navigate through the maze, get around obstacles and enemies while collecting as much treasure points as possible.

In real-world complex environments, object diversity is much higher than normally presented in the aforementioned RL and robotics tasks. The challenge from object diversity will inevitably arise when we have more capable and long-lived agents that can interact with the complex environment in many ways. Imagine an agent trying to survive in the jungle, it must deal with hundreds or thousands of species that can be potentially encountered, including things like trees, fruits, flowers, mushrooms and animals. Fruits and mushrooms are either poisonous or not. They may have very different appearances, such as their color, odor, shape and texture. Furthermore, two objects that look similar may be functionally different, such as a venomous snake and a harmless snake. On the other hand, some objects may have very different perceptual features, while they are functionally the same from the agent's perspective, such as a whitefish and a blackfish.

The challenge can be decomposed at two levels. At the first level, an agent can interact with the environment in many ways, which translate to different dimensions of functionality and semantics. For example, the agent can choose to eat an apple, play with the apple, or throw the apple at its enemy. The agent must be able to organize category knowledge with respect to these different dimensions. The second level is that within the same functional purpose, different objects can be associated with very different outcomes upon the agent's actions. Therefore, the agent must also be able to organize a discriminative categorization system among diverse objects with regard to each functional dimension. In such complex environments, it is impractical, if not impossible, to predefine all the functional categories of objects and provide them to the agent. It is inevitable that a long-lived agent is required to incrementally form categorical representations of objects based on their functional properties which are obtained through experiences interacting with the objects.

The research presented in this thesis focuses on exploring techniques of category formation in object-based environments with two defining characteristics. One is the emphasis on the challenge of *object diversity*. The other is the learning of *functionally meaningful* categories that are relevant to the agent's decision making, as opposed to category learning purely based on perceptual features. The following section will first set up a concrete example of object diversity.

1.2 Example Task of Object Diversity

A concrete example of a hunting scenario is presented in the following paragraphs, which not only intuitively grounds the problem, but also captures the essential characteristics of object-based environments. This hunting scenario is used to develop more detailed

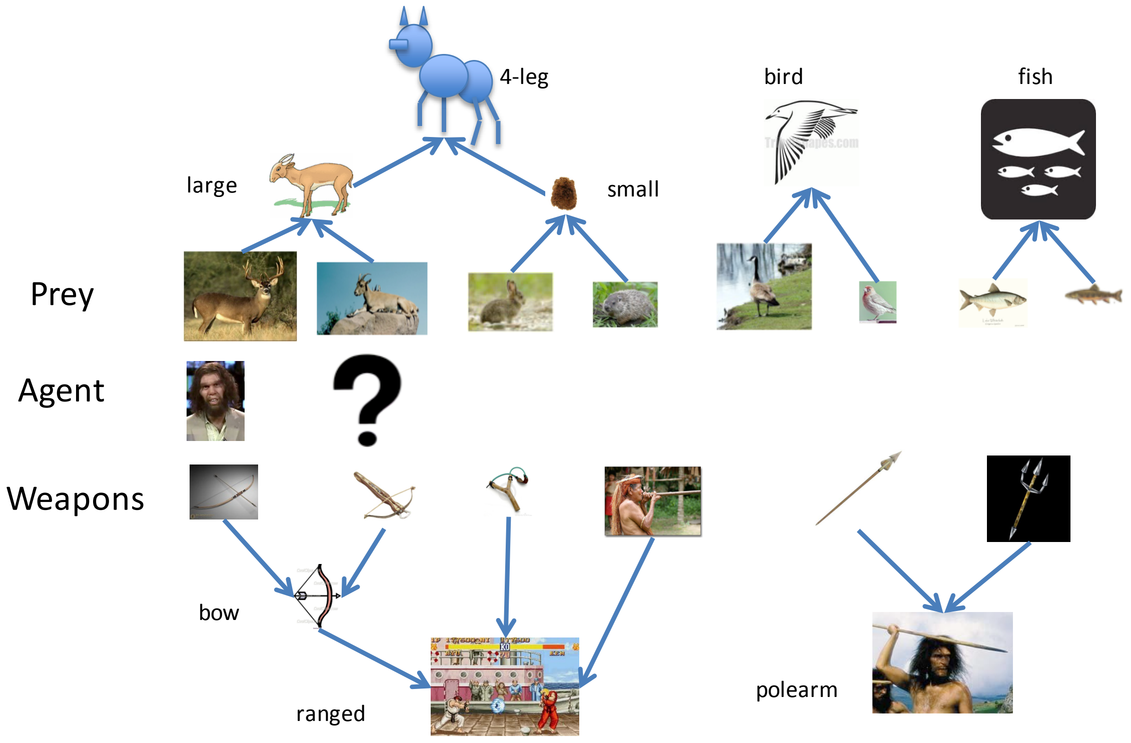
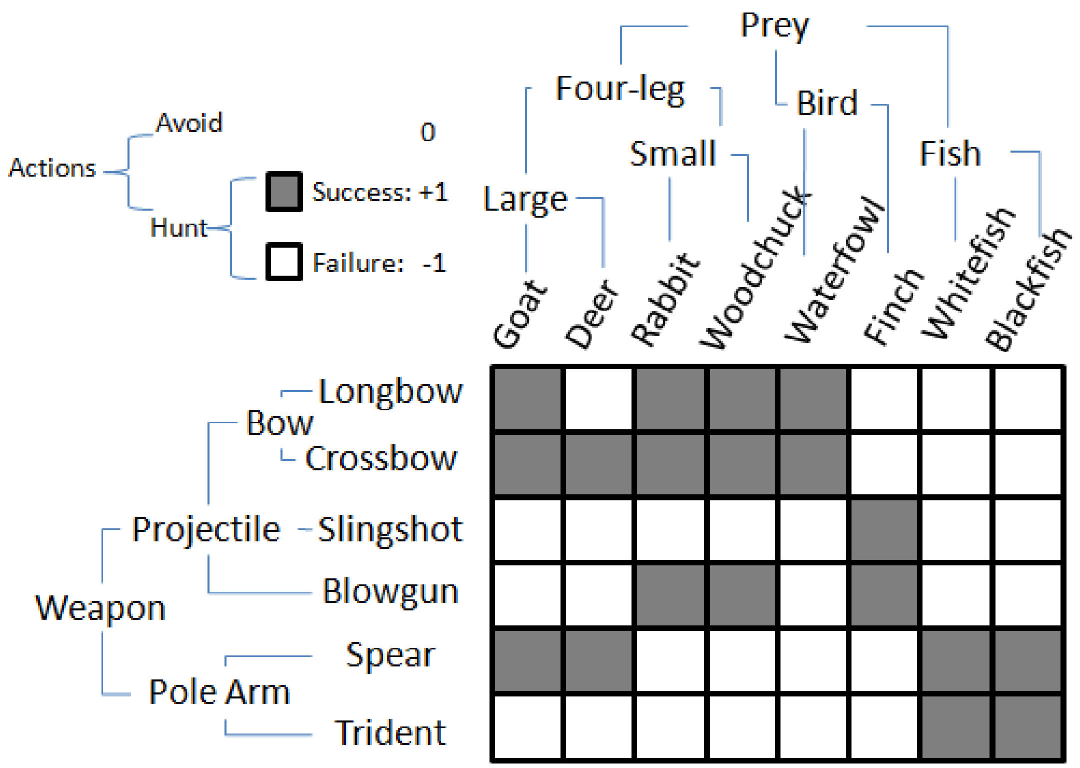


Figure 1.2: **An example of object diversity in the hunting task. The agent must make decisions regarding diverse types of objects.**

understanding of requirements, challenges and assumptions about the problem. Variations of this scenario serve as the evaluation tasks throughout the thesis.

In Figure 1.2, the agent, whose icon is a caveman, faces the decision of choosing the right weapon for the right prey. There is a certain degree of diversity in both weapons and prey. There are 8 prey and 6 weapons - in reality the numbers could be much larger. The perceptual features of each objects result in even higher diversity, because functionally equivalent objects can have perceptually distinctive variations, such as a black rabbit versus a white rabbit. We assume that the agent has no prior knowledge about the interactions between weapons and prey, and must learn to predict the outcome based on its experiences. The outcome of a weapon-prey pair depends on the functional properties of both objects, such as the range and power of a weapon or the size and robustness of a prey. In this example, memorizing the results for each of the 48 combinations is inefficient. An observation is that human can create hierarchical category abstractions, which can help achieve efficient generalizations.



	Goat	Deer	Rabbit	Woodchuck	Waterfowl	Finch	Whitefish	Blackfish
Longbow	■	□	■	■	■	□	□	□
Crossbow	■	■	■	■	■	□	□	□
Slingshot	□	□	□	□	□	■	□	□
Blowgun	□	□	■	■	□	■	□	□
Spear	■	■	□	□	□	□	■	■
Trident	□	□	□	□	□	□	■	■

Figure 1.3: **Instantiation of the interaction function in the hunting domain.**

Figure 1.3 shows an instantiation of the interaction function and the related decision making problem. The interaction outcome function in this domain is represented in the two-dimensional table. A dark cell means the corresponding weapon is good for hunting the prey and the agent will receive a reward of +1 if it chooses the action 'hunt'. A white cell means the corresponding weapon is bad for hunting the prey, and the agent will receive a reward of -1 if it chooses the action 'hunt'. The agent can always choose the default action 'avoid', which will give a 0 reward. The figure illustrates the functional categories of the objects (represented by the hierarchies for the rows and columns) and the outcomes. The mapping from perceptual features to the hierarchical categories is not shown. We assume the agent has a fixed set of feature detectors, and more detailed assumptions about the perceptual features are discussed in later chapters.

The above example, although simple, demonstrates the essential characteristics of the problem: the input to the function lies in a high dimensional space (many perceptual features for prey and weapon), and the underlying interaction function cannot be represented by simple forms (non-linear function). On the other hand, the goal is to achieve efficient learning. Learning of a complex non-linear function in high dimensional space is a challenging problem in general. However, there are certain forms of regularities in such problems which make efficient learning possible. Our hypothesis is that hierarchical categorization, as shown in Figure 1.2 and Figure 1.3, is the key to achieve efficient learning.

1.3 Overview of Chapters

In Chapter 2, we define our research problem. We identify and analyze the challenges and assumptions in the problem, and introduce the design of our general approach, which leads to specific implementations in the following chapters. As an initial step, Chapter 3 presents a learning system that integrates a unsupervised hierarchical clustering algorithm with the sparse coarse coding based value function approximation algorithm in the Soar cognitive architecture (Nason & Laird, 2005; Laird, 2008). In Chapter 4, we show that the learning system presented in Chapter 3 provides computational insights to human category learning by qualitatively modeling several prominent human category learning phenomena. In Chapter 5, we introduce a new algorithm in a principled statistical optimization framework to address a major limitation in the previous learning system. We evaluate the new algorithm with systematically generated synthetic data. Chapter 6 summarizes the contributions and discusses future directions.

Chapter 2

Research Problem Definition and Design of General Approach

2.1 Research Problem Definition

We define our research problem as finding efficient value function approximation algorithms for decision making in object-based domains, where the state representation (input to the value function) consists of objects. Our algorithms can be used as value function approximators for a reinforcement learning (RL) agent (Sutton & Barto, 1998). Unlike existing work on value function approximation in RL that focus on the convergence issue (Maei *et al.*, 2009), we focus on the structural characteristics of object-based environments to design efficient generalization algorithms. We also assume the agent is long-lived, and therefore must scale with high degree of object diversity in a complex environment. Our approach utilizes hierarchical categorization inspired by the observation of naturally occurring hierarchies in human category learning (Mervis and Rosch, 1981).

In the following sections, we first elaborate on the requirements of efficient value function approximation. Then we discuss and analyze the specific challenges and assumptions in the research problem. Finally, we describe our general approach, which leads to specific implementations to be discussed in later chapters.

2.2 Requirements of Efficient Learning

In the hunting example, in order to better survive, the agent needs to generalize its experience to novel situations. For example, having learned how to hunt a whitefish with a spear, the agent should be able to successfully hunt a blackfish without going through the same trial-and-error learning procedures again, which would be costly to the agent's survival. On the other hand, incorporating a piece of new experience into its current knowledge base should not incur too much computational cost. For example, it is not acceptable to have a learning algorithm where the computational cost of processing a new training sample grows linearly with the total number of samples in the agent's knowledge base. The above two requirements address two aspects of efficiency in learning: sample efficiency and computational efficiency.

Sample efficiency concerns the relation between the number of training samples and accuracy of the algorithm. Traditionally, learning algorithms are evaluated by asymptotic performance – the degree of accuracy that can be achieved given enough training samples. In online learning problems, the intermediate learning curves are even more important than the asymptotic performance, because every decision is important to the agent's survival.

Computational efficiency concerns the computational cost of incorporating each new training sample into the agent's existing knowledge. For most optimization problems, the learning algorithm has to re-evaluate every training instance in order to find the true optimal solution whenever a new training sample is incorporated. Such algorithms, which do not consider incremental updates, are called batch algorithms. Batch algorithms are unacceptable for a long-lived learning agent, which can potentially collect large amount of experiences. Therefore, we need an online learning algorithm that incurs a fixed, or at least a very slowly growing, computational cost for incorporating a new training sample. Such an online algorithm can usually be constructed as an approximation algorithm that is based on the same optimization criteria, but is not guaranteed to find the global optimal solution. Although online algorithms exist for many optimization problems, it can be a technical challenge to find a good one.

2.3 Challenges

The challenges arise from the combination of the efficient learning requirements discussed above and the environmental complexity which will be discussed in the following paragraphs. In an object-based environment, there are two major components in the environment model. One is the distribution of different objects in the feature space, and the other is the interaction dynamics among objects.

2.3.1 Object Diversity

The jungle survival example and the hunting tasks in Chapter 1 are intuitively grounded examples about the challenge of object diversity. More formally, object diversity is characterized by the distribution of object instances in the feature space. In our target environments, we expect the feature space has rather high dimensions and the instances are randomly distributed across the entire feature space. We will give more precise characterization of object diversity when we evaluate our algorithms in later chapters. At this point, an intuitive understanding should suffice.

High degree of diversity undoubtedly corresponds to large amounts of information to be learned and poses a challenge to the agent. In natural environments, high degree of object diversity is a pervasive phenomenon: from living organisms to manmade objects, and natural objects. The law of increased entropy implies that disorganization, randomness and chaos in the universe tend to increase. As a consequence, objects in the physical world end up in very diverse states. For living organisms, in the light of evolution by means of natural selection, new species have evolved to fill the various environmental niches. The tendency for diversity and complexity to increase in evolutionary systems is recently proposed as the “Biology’s first law”, or “zero force evolutionary law” (ZFEL) (McShea & Brandon, 2010). It is argued that ZFEL complements the principle of natural selection – the force that helps make order out of chaos.

2.3.2 Nonlinear Interaction Function

One way to specify the dynamics of a complex environment is by defining condition-action like rules. Each rule corresponds to an event, such as a successful hunting. The rule can be viewed as a function. The input to the function is the condition for the event to happen. For example, using a crossbow to hunt a deer is the condition for a successful hunting. The output of the function is the effect of the event. For example, the effect of a successful hunting is receiving +1 reward. We refer to such event functions in object-based environment as the interaction function to reflect its object-based nature. In a complex object-based environment, the interaction function can exhibits high degree of nonlinearity as illustrated by the example in Figure 1.3. This is because events, such as the successfulness of hunting, are determined by complex underlying physics. In such situations, piecewise approximation is a natural solution. There are a variety of piecewise function approximation techniques used in similar situations, which involve partitioning the input space into smaller regions (pieces, slices), and using simple function forms to approximation the value within each region. One example is the Cerebellar Model Articulation Controller (CMAC), which has been used as a sparse coarse coded value function approximator in reinforcement learning problems (Sutton, 1996). Another example, used in very different domains, is the finite difference method and the related finite element method (Johnson, 1987). Unlike CMAC, they do not use overlapping cells, and are more suited to approximate smooth functions in lower dimensional space. However, all piecewise approximation methods, if applied in their plain forms, are subject to the "curse of dimensionality" since the number of pieces grows exponentially with the number of input dimensions. In complex object-based environments, high dimensionality is expected because each object is described by a large number of features. Meanwhile, the intrinsic manifold of the data, or the true dimensionality of the data is expected to be high as explained in the previous section about object diversity. Therefore, to deal with the challenge of value function approximation in complex object-based environment, we must resort to additional structural assumptions, which are discussed in the next section.

2.4 General Assumptions and Approach

Given the above challenges, it is unlikely that there exist general solutions without applying additional constraints to the problem. Our strategy is to identify the general constraints in object-based environments, and then design efficient function approximation algorithms based on those general assumptions.

2.4.1 Assumptions

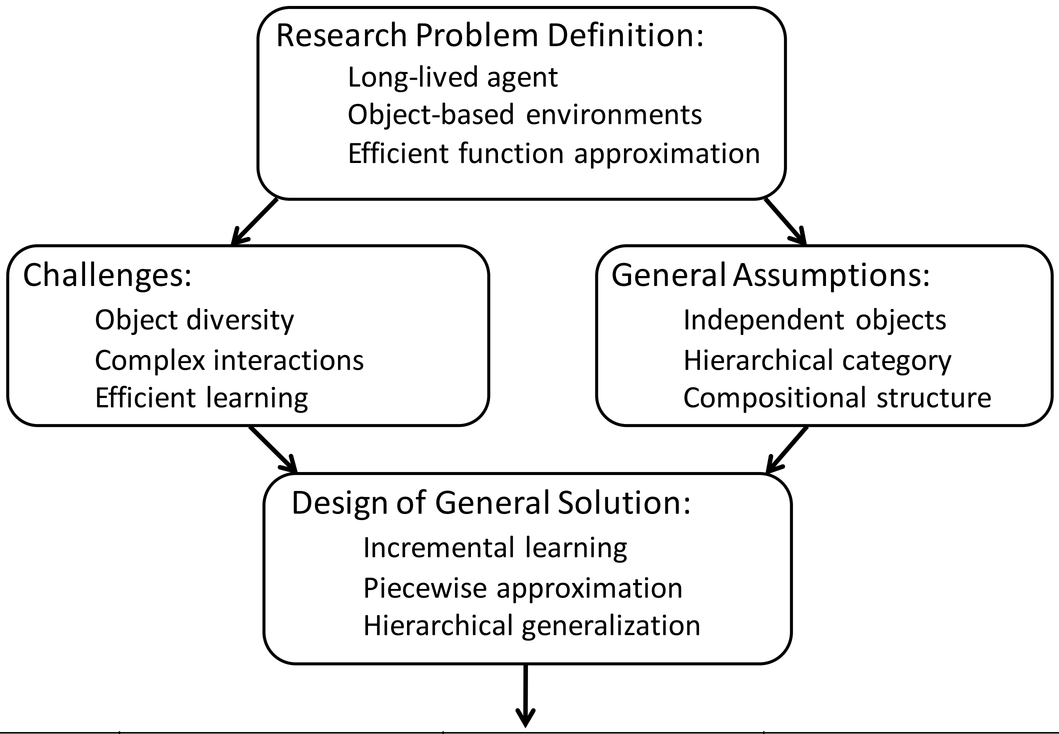
The first assumption is about the compositional structure of object-based environments. Objects are independent entities, each having its own perceptual features and functional properties. The utility value of a state is determined by the functional properties of the objects and the relationships among the objects. This suggests that learning categories for individual objects is more effective than directly categorizing the entire composed state, because categories of objects can be reused in different situations.

The second assumption is about the distribution characteristic of objects in natural environments. Instead of assuming that objects are uniformly distributed in the functional space, we assume there is a heterogeneous functional similarity structure among objects, such that object categorization for a specific purpose (such as prey or weapon) can be organized into a hierarchy as shown in Figure 1.2. Using hierarchical taxonomy to organize object categories has been a common practice for a long time, one of the most prominent examples being the Cyc project (Lenat & Guha, 1990). Tenenbaum *et al.* (2006) described a computational model that can discover structures like trees, rings, dominance hierarchies, and cliques, for a variety of physical, biological, and social domains. Among those structures, hierarchy is the optimal structure for organizing object categories. In addition to supporting efficient generalization, hierarchical data structures can also support efficient online learning algorithms. COBWEB (Fisher, 1987) is an incremental hierarchical clustering algorithm that has inspired this work.

2.4.2 General Design

The general solution, as suggested in Figure 1.2 and Figure 1.3, is to form hierarchical object categories based on functional similarities. The state representation consists of combinations of hierarchical categories from each composing object. Then piecewise approximation can be performed utilizing such combinatorial hierarchical partitions of the sample space. The detailed algorithms are presented in Chapter 3 and Chapter 5.

Figure 2.1 provides a high level outline of the thesis. The arrows represent logical dependencies. We first formulate and define our research problem: efficient value function approximation for long-lived agents in object-based environments. Then we identify specific challenges as well as general assumptions, which lead to the major design considerations in our general solutions. From Chapter 3 to Chapter 5, we present two different implementations, with evaluations both based on the algorithm's functionality (sample efficiency), and on cognitive plausibility (match with human behavior).



	Implementations	**Evaluation Tasks**	**Evaluation Criteria**
Chapter 3	Hierarchical clustering with sparse coarse coding	Artificial hunting task	Functionality (sample efficiency)
Chapter 4			Cognitive plausibility (match human behavior)
Chapter 5	Probabilistic latent hierarchical class model	Systematically generated synthetic data	Functionality (sample efficiency)

Figure 2.1: **Outline of the thesis.**

2.4.3 Implementations and Evaluations

Cognitive architectures such as Soar, have been serving as a framework for building robust, long-lived, online learning and reactive agents (Laird, 2008). Recent extensions to Soar enhance its capabilities in sub-symbolic realms such as reinforcement learning (Nason & Laird, 2005), episodic learning (Nuxoll & Laird, 2007), and spatial reasoning (Lathrop & Laird, 2009; Wintermute, 2010). One major motivation of the current work is to further extend the category learning capability of Soar to deal with the challenge of object diversity which is common for a long-lived agent in complex natural environments. Therefore, some of the initial implementations, which are discussed in Chapter 3 and Chapter 4, are integrated with the existing Soar system. However, the second algorithm presented in Chapter 5 is implemented independent of Soar, because the research does not need to be constrained by the details of Soar other than respecting the most important constraints of a general cognitive agent, namely incremental learning and scalability.

Due to the uniqueness of our approach, we cannot find proper evaluation tasks and data sets from existing sources. Therefore, we use new artificial domains and synthetic data to evaluate our algorithms. For the evaluation criteria, our major goal is to achieve better functionality. On the other hand, being able to match with human category learning behavior is also a desirable outcome. In Chapter 4, we present such an evaluation using the first algorithm (Chapter 3). The second algorithm (Chapter 5) is only evaluated by functionality. Matching with human behavior using the second algorithm is discussed as a future direction in Chapter 6.

Chapter 3

Combining Unsupervised Clustering and Sparse Coarse Coding

In the previous chapter, we formulate our research problem as finding efficient value function approximation algorithms in object-based domains. We discussed the challenges and proposed the general approach of combining hierarchical categorization with piecewise approximation. In this chapter, we provide an implementation of such a learning system, which consists of a category learning system for hierarchical categorization, and a sparse coarse coding system for piecewise function approximation.

3.1 Background

Coarse coding is a popular general approach to achieve value function approximation for reinforcement learning agents. Coarse coding approximators generally involve a linear combination of coarse grained basis functions. Some examples are CMAC (Sutton, 1996), adaptive tile coding (Whiteson *et al.*, 2007), and kd-tree Q Learning (Vollbrecht, 2000).

For object-based environments, relational representations are considered to be the most natural choice of state representation. The two components in a relational representation are objects, and the relationships among objects. In this thesis, we focus on the object aspect and assume a state representation that consists of an ordered set of component objects, with each object represented by its own feature vector. The set of objects are ordered because each of them is assumed to take a different role in the represented situation. For example, in the hunting task, weapon and prey are two distinctive roles. The state representation is composed of two object categories, one for prey, one for weapon. In an object-based environment, objects are independent entities with intrinsic functional properties, and the utility value of a state is directly determined by the objects and their roles. Therefore, our approach of representing the state as combinations of object categories not only reduces the dimension via categorization, but also respects the compositional structure of the environment.

3.2 Our Approach

Figure 3.1 shows the overall architecture of our implementation. The original prototype system was implemented in Soar and uses the existing architectural mechanisms, including Soar-RL (Nason & Laird, 2005) and production rule based knowledge encoding. The system first transforms each input object into symbolic hierarchical categories via a hierarchical clustering algorithm (the bottom half in Figure 3.1). The symbolic hierarchical categories are used as representational basis to form coarse coding basis functions (the top half). Finally, the combinatorial hierarchical basis functions are linearly combined to approximate the value as in other coarse coding algorithms. The coefficients of each basis function correspond to the weighted connections from the top layer to the output as shown in the top of the figure. These weights are adjusted based on the desired output value. Regarding the overall structure, from the input and output layers, there are two intermediate representational layers and three processing steps (outlined on the left of Figure 3.1). In the following section, we present the details of the process in two sections: the category learning system, which learns the mapping from input to the representational basis, and the sparse coarse coding system, which performs piecewise function approximation.

3.2.1 Category Learning System

The category learning system maps raw input features into symbolic categories for each individual object. It has two modes: learning mode and recognition mode. The learning mode corresponds to the process of category formation. When adding a new instance to the system, either a new symbolic category is created or existing categories are modified. In the recognition mode, an input object is mapped to an existing category in the hierarchy based on its features.

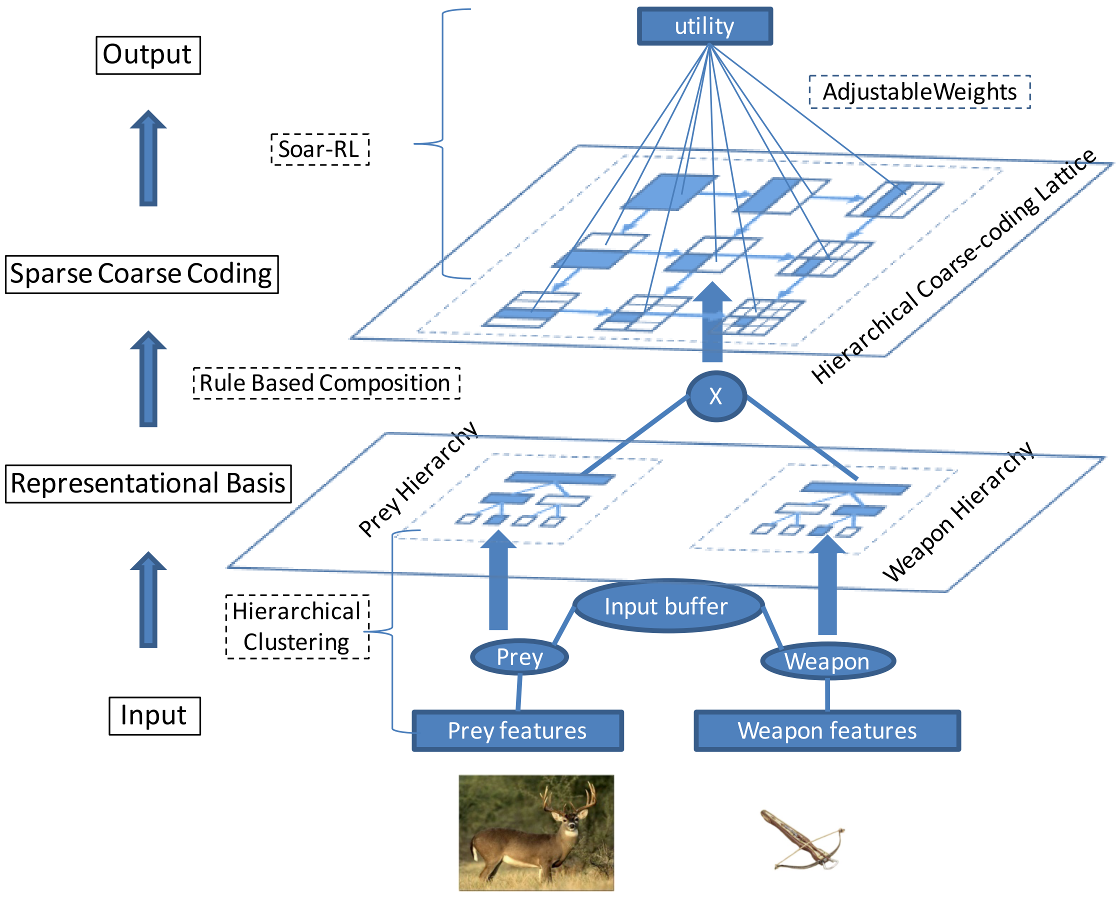


Figure 3.1: The overall learning architecture.

3.2.1.1 Category Formation

For category formation, we use an unsupervised hierarchical clustering algorithm. It is unsupervised because the categorizations are determined solely by the input features and there is no pre-specified category label. In this framework, prior knowledge can be injected to select a subset of features to bias the clustering. The hierarchical clustering algorithm is adapted from COBWEB (Fisher, 1987) and CLASSIT (Gennari *et al.*, 1989), which recursively partition the training samples to build up a hierarchy based on the optimization criteria listed in Equations 3.

Equation 3.1 (1) defines the *partition utility* optimization objective, which gives higher scores for categories with high within-group resemblance and prefers fewer number of categories. This is achieved by using the sum of *category utilities* penalized by the number of categories *n*. Category utility is defined in Equation 3.1 (2) which is the sum of feature utilities for that category. Feature utilities for nominal features and numeric features are defined in Equation 3.1 (3) and (4). For nominal feature, the utility is the sum of squared probabilities for each discrete value. For numeric features, in Equation 3.1 (4), σ_0 is the standard deviation of the feature's distribution across all training samples, and σ_i is the standard deviation within category *i*.

$$PartitionUtility = \frac{\sum_{i=1}^{n} P(C_i) \bullet CategoryUtility(C_i)}{PENALTY(n)} \tag{1}$$

$$CategoryUtility(C_i) = \sum_{j} FeatureUtility(j) \tag{2}$$

$$NominalFeatureUtility = \sum_{k} P(v_k)^2 \tag{3}$$

where v_k is the *kth* value of the feature

$$NumericFeatureUtility = \frac{1}{1 + {}^{\sigma_i}\!/_{\sigma_0}} \tag{4}$$

where

σ_i is the feature's standard deviation under category i

σ_0 is the feature's overall standard deviation

Equation 3.1: Definition of partition utility.

Figure 3.2 illustrates an example of hierarchical categories for objects with three features: color (nominal), shape (nominal) and size (numeric). The statistics of each feature are listed with each category node. For a nominal feature, the frequencies of each discrete value are recorded. For a numeric feature, the mean and variance are recorded. Each category node is also visually depicted. The optimal hierarchy should maximize the partition utility from top down recursively.

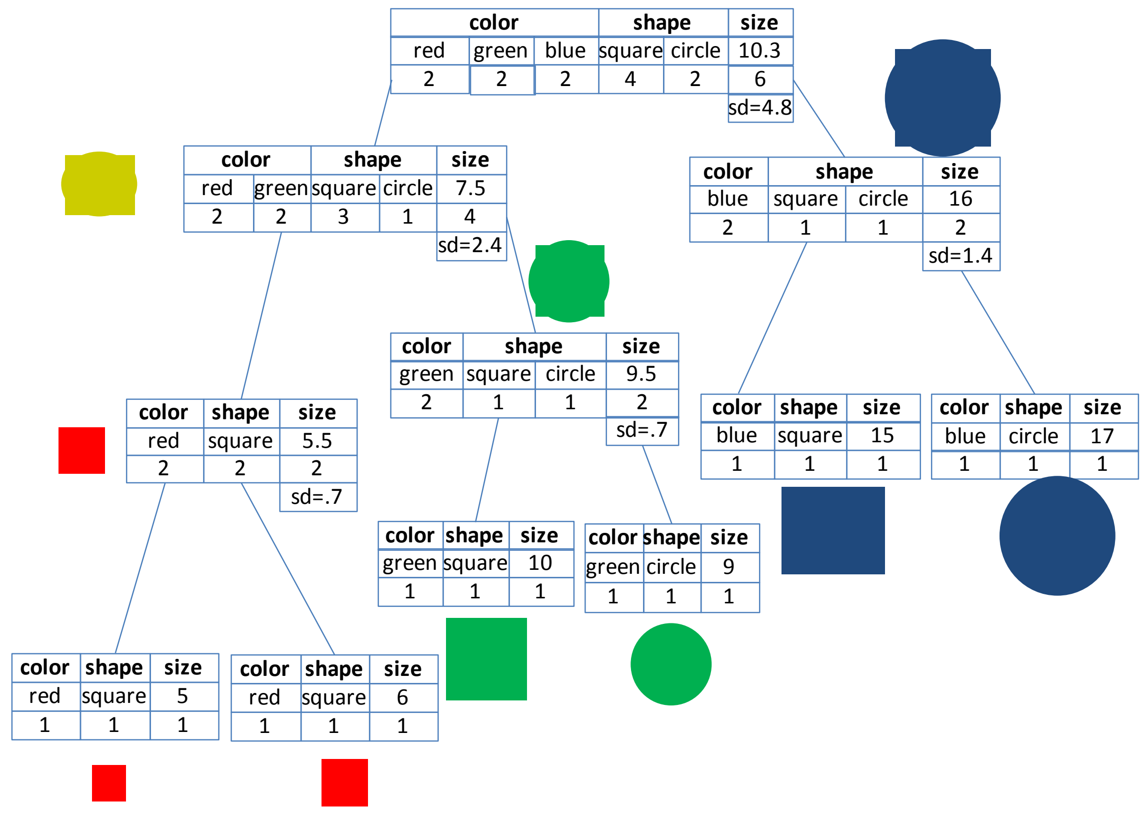


Figure 3.2: An example categorization hierarchy with detailed data structure.

The incremental algorithm that searches for such optimal structure is based on greedy search in the space of hierarchical structures. The two primitive operators are *split* and *merge*, as illustrated in Figure 3.3. The numbers in each hierarchy represent the unique identifiers of the categories. The entire search space can be viewed as a graph structure, with the vertices being distinctive hierarchical structures and the edges being split/merge operators. To evaluate the hierarchy after a merge or a split, the data structure shown in Figure 3.2 must be updated accordingly and the partition utility shown in Equation 3.1 (1) must be recalculated. A greedy algorithm can be used to search for the new optimal hierarchy in the neighborhood of the current structure whenever a new instance is integrated into the hierarchy. Details of the incremental learning algorithm and its complexity analysis can be found in the original COBWEB paper (Fisher, 1987).

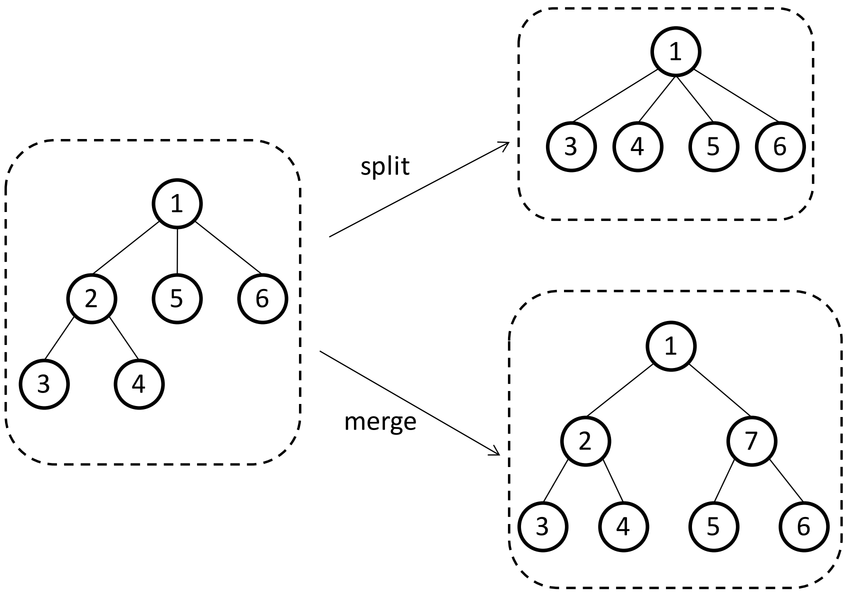


Figure 3.3: **Local restructuring operators in the space of hierarchies.**

3.2.1.2 Category Recognition

As illustrated in Figure 3.1, in order to predict the utility value of an input state consisting of multiple objects, each object is first mapped to an existing symbolic category. This is achieved via the category recognition process. In our system, the category recognition process selects the category with the highest posterior probability given the input features. For example, if the prey is furry and has long ears, then rabbit is the category with the highest posterior probability. In the hunting example, recognition based on perceptual features is straightforward if the system is presented with good discriminative features such as whether the prey has wings or scale skin, the shape of tail, ear and size, *etc*. In more general situations, there can be multiple categories with close posterior probabilities. Our system uses the winner-takes-all principle – only the category with the highest probability is selected as the output. Such a recognition process is recursively applied at each level of the hierarchy, and the final output is the path of categories at different levels of abstraction, as illustrated in Figure 3.1.

Equations 3.2 describe how the posterior probabilities are calculated. Equation 3.2 (1) states that *P*(*C*$_i$|*I*), the posterior probability that the instance belongs to category *i* given the observed feature vector *I*, is proportional to the product of *P*(*I*|*C*$_i$) and *P*(*C*$_i$), where *P*(*I*|*C*$_i$) is the conditional probability of the observed feature vector *I*, and *P*(*C*$_i$) is the prior probability of that category. Equation 3.2 (2) calculates *P*(*I*|*C*$_i$) by taking the product of *P*(*I*$_j$|*C*$_i$), the conditional probability of each individual feature *I*$_j$, assuming the distribution of individual features are independent. Equation 3.2 (3) is the conditional probability of an individual nominal (symbolic) feature under category *i*. Equation 3.2 (4) is the conditional probability for a numerical feature based on the assumption that the distribution being Gaussian. The recognition algorithm descends the category hierarchy and chooses the category with the highest posterior probability.

$$P(C_i \mid I) \propto P(I \mid C_i) \bullet P(C_i) \qquad (1)$$

where

I is the input feature vector

C_i is the *ith* category

$$P(I \mid C_i) = \prod_j P(I_j \mid C_i) \qquad (2)$$

where I_j is the value of the *jth* feature in the input

$$P(I_j \mid C_i) = \frac{frequency_i(I_j)}{n_{ij}} \qquad (3)$$

where

$frequency_i(I_j)$ is the count of the discrete value I_j within category i

n_{ij} is the total count of the nominal feature j within category i

$$P(I_j \mid C_i) = \frac{1}{\sqrt{2\pi\sigma_{ij}^{\,2}}} \exp\left(-\frac{(I_j - \mu_{ij})^2}{2\sigma_{ij}^{\,2}} \right) \qquad (4)$$

where

μ_{ij} is the mean of numeric feature j within cateogry i

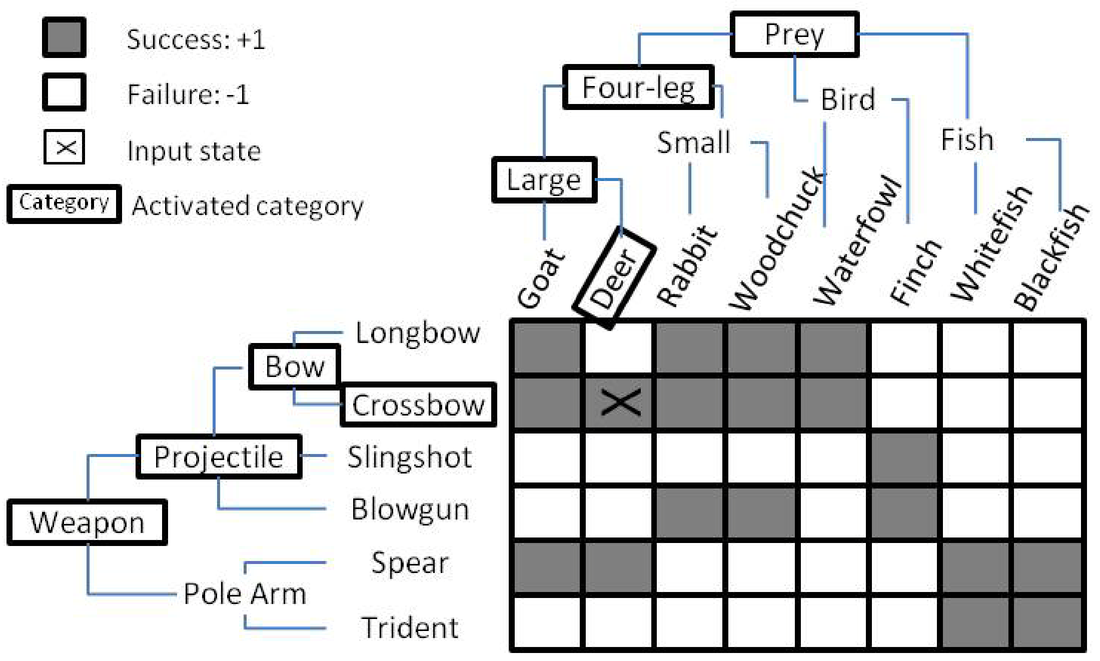
σ_{ij} is the standard deviation of numeric feature j within category i

Equation 3.2: Category recognition.

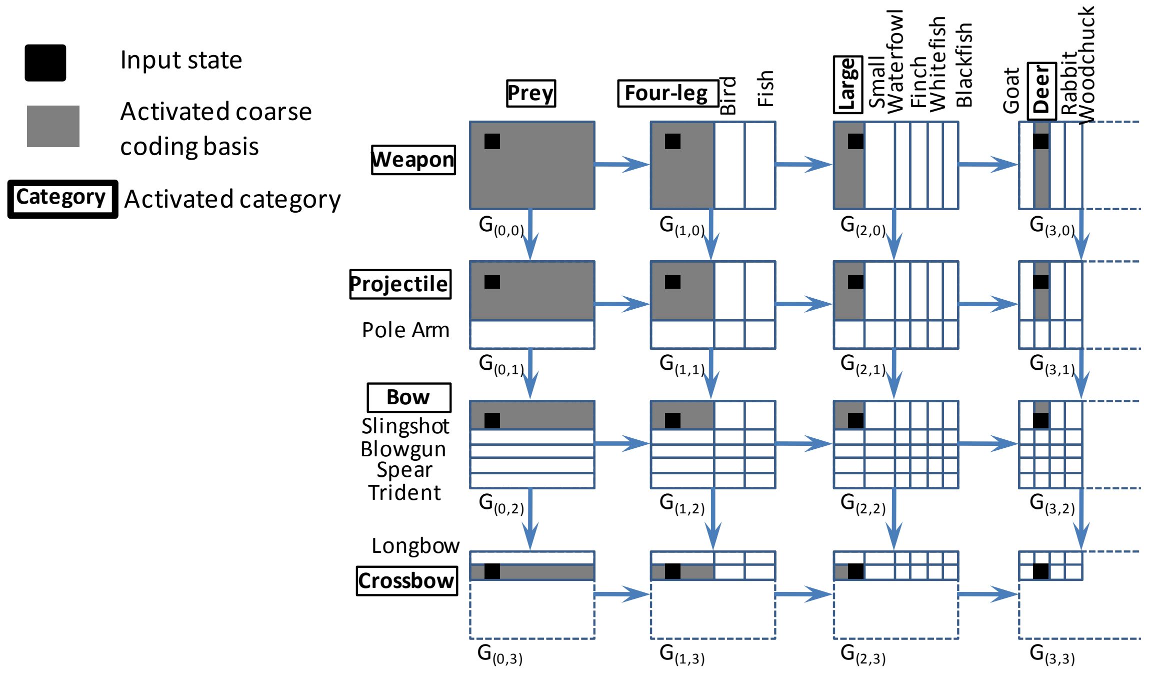
3.2.2 Sparse Coarse Coding System

As illustrated in Figure 3.1, after mapping each object to its symbolic categories, the state representation consists of an ordered set of hierarchical categories. The sparse coarse coding system composes the categories to form a set of coarse coding basis functions. Figure 3.4 shows a concrete situation for a specific input based on the hunting task example in Figure 3.1 from the previous chapter.

In Figure 3.4a, the state space is represented as a 2D grid after mapping objects with raw features into their symbolic hierarchical categories, which are output from the category learning system, and serve as the inputs to the coarse coding system. The specific example is hunting a deer with a crossbow, and the activated categories are highlighted. The coarse coding system approximates the value function in the now transformed and reduced state space. In the Figure 3.4b, the coarse coding system consists of 16 grids in a 4 by 4 lattice structure. Each grid represents a partition of the state space at a specific level of resolution according to the levels of the two composing hierarchies. The levels are indicated by the numbers associated with each grid. For example, $G_{(1,2)}$ means the partition grid consists of prey categories at level 1 and weapon categories at level 2. The black dots spatially represent the input instance, and the grey areas represent the more general regions covered by the activated coarse coding basis. The region with the dotted border in 7 of the grids on the lower and right sides of the lattice means there are no more specific rules generated for those regions because it has already reached the leaf level of the categorization hierarchies. The utility value for a state is approximated by the sum of multiple activated basis functions, which are simultaneously updated by the learning algorithm.



(a)



(b)

Figure 3.4: **Illustration of sparse coarse coding.**

The specific input is hunting deer with crossbow. (a) shows state space and activated categories. (b) shows the organization of coarse coding basis into a lattice.

We formally describe the overall algorithm below. To learn a target function, the system first maps the input objects into a vector of *functional roles R*, which represents the distinctive arguments in the input of the target function. The vector *O* represents the input objects binding with *R*, thus determining the ordered set of objects:

$$R = (r_1, r_2, \dots, r_n)$$

$$O = (o_1, o_2, \dots, o_n)$$

To give a more concrete example, assume the general target function is to predict the utility of hunting some prey with some weapon. The inputs are two objects: rabbit and bow. If the system has no prior knowledge about the detailed functional roles, then it may use a single functional role for all objects. According to our notation, input to the system will look like *R*=(generic-object, generic-object), *O*=(rabbit, bow). If we know that prey and weapon play distinctive functional roles, then the input will look like *R*=(prey, weapon), *O*=(rabbit, bow). If we know a more refined structure about the interaction model, for example, that there are two interaction components, one is about how to get close to the prey, the other is about how to choose the most effective weapon, then we may want to categorize the prey based on two different criteria. The input will look like *R*=(prey-sensing, prey-physical, weapon), and *O*=(rabbit, rabbit, bow). These different degrees of structural level prior knowledge can be conveniently encoded as rules in our system. After matching objects with functional roles, the category learning system incrementally builds a set of hierarchies *H* correspondingly:

$$H = (h_1, h_2, \dots, h_n)$$

Let *height*(h_i) denote the height of the hierarchy h_i, and k_i denote a cluster/node within the hierarchy. Let *level*(k_i) denote the level of cluster k_i in the hierarchy h_i, with the root node level being 0. Cells, grids and their relations, shown in Figure 3.4, are defined as following:

$$Cells = \{C_K, K = (k_1, k_2, \dots, k_n) | k_i \in clusters\ in\ h_i\}$$

$$Grids = \{G_L, L = (l_1, l_2, \dots, l_n) | 0 \le l_i \le height(h_i)\}$$

$$C_K \; belongs \; to \; G_L \equiv \; \forall i \in [1,n], level(k_i) = l_i$$

$$G_{L1} \prec G_{L2} \; \equiv \; \forall i \in [1,n], l1_i \leq l2_i$$

More intuitively, each cell represents a rule in our RL system. A set of cells are composed into a grid that partitions the state space at a specific level of resolution. There is a lattice structure among the grids with the transitive relation *coarser-than* ($\prec$) represented by the arrows in the figure. For example, $G_{(1,1)}$ is coarser than both $G_{(1,2)}$ and $G_{(1,2)}$. Both $G_{(1,2)}$ and $G_{(2,1)}$ are coarser than $G_{(2,2)}$. But there is no such relation between $G_{(1,2)}$ and $G_{(1,2)}$.

Then given the set of input objects, the *activation* of a cluster k_i is denoted as $a(k_i)$:

$$a(k_i) = \begin{cases} 1 \; if \; o_i \in \; k_i \\ 0 \; if \; o_i \notin \; k_i \end{cases}$$

In the above equation, the mapping from o_i to k_i is achieved via category recognition, and only a single path of clusters are activated for a particular input as shown in Figure 3.4. $a(k_i)$=1 means an object in the current state, bound to the corresponding functional role, is an instance of the category represented by that cluster. The activation of a cell, $a(C_K)$, is defined as:

$$a(C_K) = \prod_{i=1}^{n} a(k_i)$$

$a(C_K)$=1 means the rule (a cell corresponds to a rule) matches the current state and will be fired to participate in predicting and learning the target value. The weight, $w(C_K)$, from the cell to the output unit is represented as a numeric value associated with the rule in the RL system. The learning algorithm updates the weights according to the *delta rule* as in stochastic gradient descent methods. α is the learning rate, and *t* is the target value:

$$y = \sum_{C_K} w(C_K) a(C_K)$$

$$\Delta w(C_K) = \frac{\alpha}{\sum_{C_K} a(C_K)} (t - y) a(C_K)$$

The connection between the coarse-coding layer and the output unit is always sparse, since, for any input, only one cell from each grid in the lattice has non-zero activation. This is due to the competitive learning (winner-take-all principle) nature of the hierarchical clustering layer – only one cluster is activated at each level.

3.2.3 Algorithm Complexity Analysis

The time and space cost of our system are reasonably bounded under practical assumptions. As in COBWEB, processing in each hierarchy in our system takes O(logN) time, where N is the number of leaf nodes, for both predicting and assimilating a new instance, given bounded branching factor and fixed dimensions in input features. Rule matching in Soar is based on the Rete algorithm (Forgy, 1982), which has constant time cost given bounded changes in working memory. The remaining time cost is determined by the number of grids in the lattice, which is:

$$|Grids| = \prod_{i=1}^{n} height(h_i)$$

For example, in Figure 3.4, there are two hierarchies, each having a height of four, and therefore the number of grids is sixteen. Since there always exists some level beyond which functional differences are too small to be meaningful, we can assume an effectively small and fixed height by keeping a limited number of leaf nodes for each hierarchy. Furthermore, based on the observation that the cardinality of object interaction is usually small, we can also assume that the number of functional roles in a target function is bounded by a small constant, and therefore the time cost of each update and prediction is practically constant. The space cost for each hierarchy is O(N). The space cost for the coarse coding lattice is $O(N^{|R|})$ where |R| is the number of functional roles in the target function. Given the above assumption about a bounded number of leaf nodes and a fixed small number of functional roles, the space cost is also constant.

3.3 Evaluation

There are no existing benchmark tasks that reflect the challenge of object diversity that we are pursuing. Therefore, we created a new artificial domain based on the hunting scenario to evaluate our system. In addition to hierarchical categories of objects, we added obstacles that interact with the weapons and animals, and introduced uncertainties in the execution of actions to make the task more complex.

3.3.1 Evaluation Task

The domain is a simulated environment with discrete time and a discrete location grid. The agent is equipped with different types of ranged weapons. To attack a prey, the agent must choose an appropriate weapon and distance from which to attack. The efficacy of a weapon depends on the functional properties of the weapon and the prey, as well as the distance to the prey. Moreover, the prey may detect the agent and become alerted before the attack action, which significantly reduces the success rate of hunting. In order to get close enough to the prey without alerting it, the agent can approach the prey from behind static obstacles. The probability of successfully moving towards the prey depends on the sensing capabilities of the prey and the types of obstacles between the prey and agent. This domain captures the characteristics and challenges that we intend to address. First, there are multiple interacting objects and multiple types of interaction: the interaction between prey and weapons, and the interaction between prey and obstacles. Second, there is diversity for each object type: prey, obstacles, and weapons all have instances with diverse functional properties.

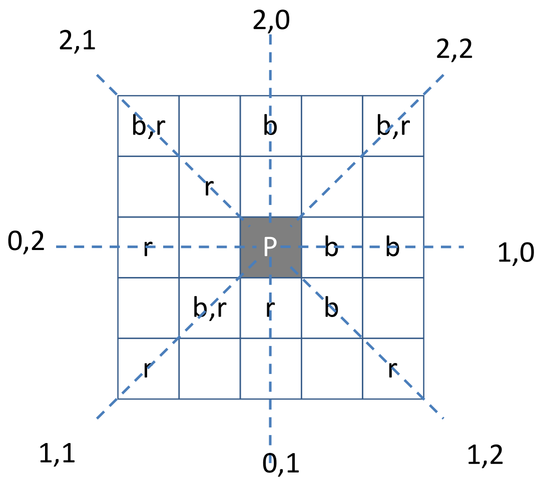


Figure 3.5: **Schematic representation of a hunting scenario.**

Figure 3.5 shows a scenario, where the prey (P) is in the middle. There are two types of static obstacles: bush (b) and rock (r), which can coexist in the same cell (b, r). For simplicity, the agent can only approach the prey from eight different directions, as represented by the dashed lines. The first number associated with a path indicates bush distance to the prey and the second number indicates rock distance. 0 indicates that the object is absent or out of effective range. There is no additive effect from multiple occurrences of the same obstacle type, so if there are two bushes at both distance 1 and 2, the effect is the same as if there is one bush at distance 1.

Figure 3.6 shows the model of interaction between objects in the domain. Grey boxes represent physical objects, white boxes with solid lines represent abstract quantities, white boxes with dotted lines represent stochastic events with probabilistic outcomes. Large dashed boxes highlight local interactions among objects. Prey have different sensing properties which affect how likely it can sense the agent when blocked by a bush or rock, as highlighted by the "Prey Obstacle Interaction" box. The prey also has physical properties that affect how easily it can be shot and fatally wounded, as highlighted by the "Prey Weapon Interaction" box. Finally, distance to the prey affects both types of interactions. In the task, the agent must choose a weapon, an attacking path and a

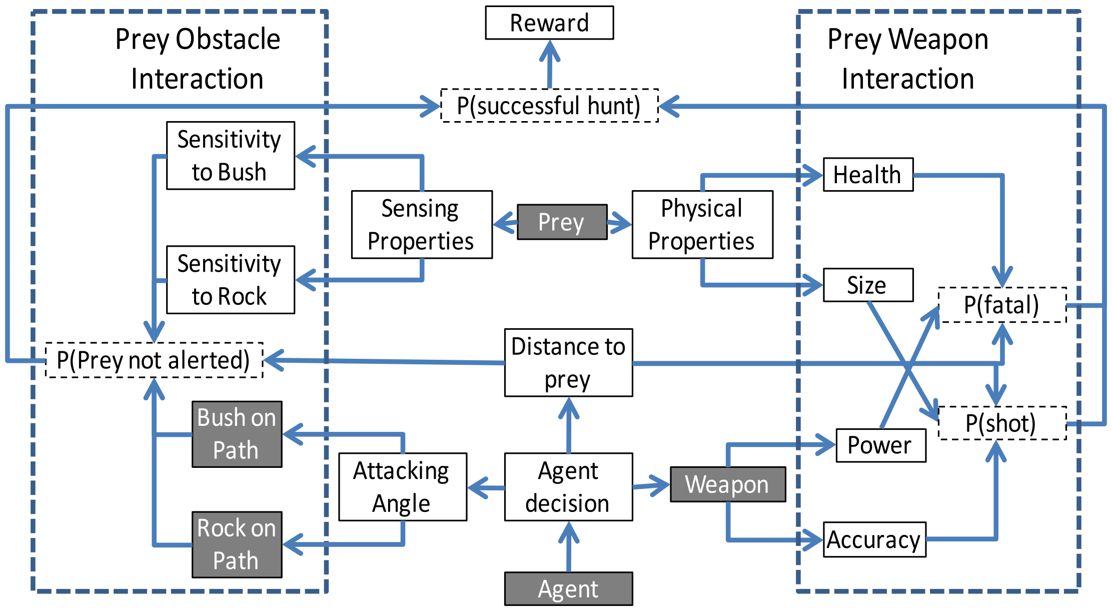


Figure 3.6: **Complex object interaction model of the hunting domain.**

shooting distance. If the hunting is successful, the agent receives a positive reward.

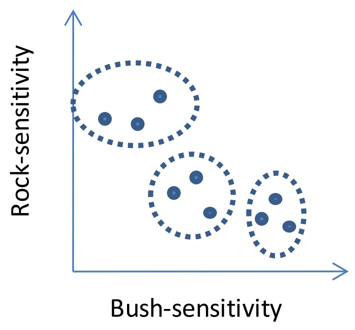
The functional properties of a prey and weapon are represented as continuous numbers in our environment model. We assume the agent can measure and internally represent these quantities as numeric features. Prey sensing properties consist of *sensitivity to bush* and *sensitivity to rock*. Prey physical properties consist of *health* and *size*. Weapon properties consist of *power* and *accuracy*. The probabilistic outcomes are determined by the numeric values of related features. For example, the probability of successfully approaching behind a rock is higher if the prey has higher *sensitivity to bush*. The probability of fatally wounding a prey is higher if the weapon has higher power or the prey has lower health. More details will be presented in the next section.

We call the above numeric features *functional features,* as they represent an object's perceivable functional properties that have intrinsic meanings to the agent. For many functional features, the values are "expensive" to obtain in nature because they have to be tested out by actual interactions with the object. There are other non-functional *perceptual features* (not shown in the figure) of prey that are more easily perceivable, such as visual, smell, and sound features, and they can be correlated with certain functional features. These perceptual features are useful for the agent to predict the functional features when they are not directly available. For example, the agent cannot directly observe a prey's sensing properties before choosing the action, and has to make predictions based on correlated perceptual features such as the size and shape of eye, nose, and ear.

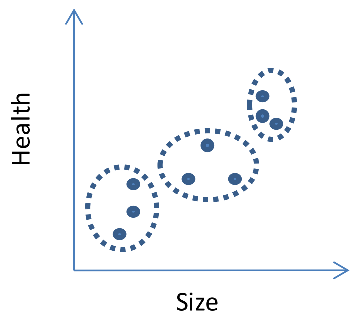
3.3.2 Empirical Results

As discussed in Chapter 2, the general requirements are computational efficiency and sample efficiency. Our empirical evaluation will focus on sample efficiency, i.e. how fast the performance improves with regard to the number of training samples. The computational complexity has been analyzed in section 3.2.3.

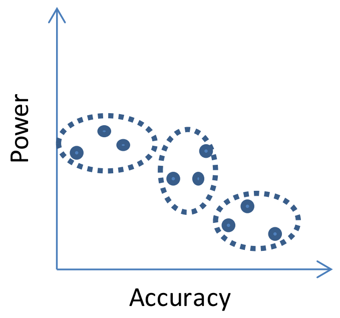
3.3.2.1 Evaluation Data

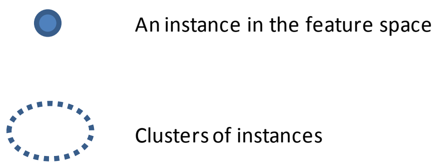


(a) Distribution of prey sensing properties



(b) Distribution of prey physical properties



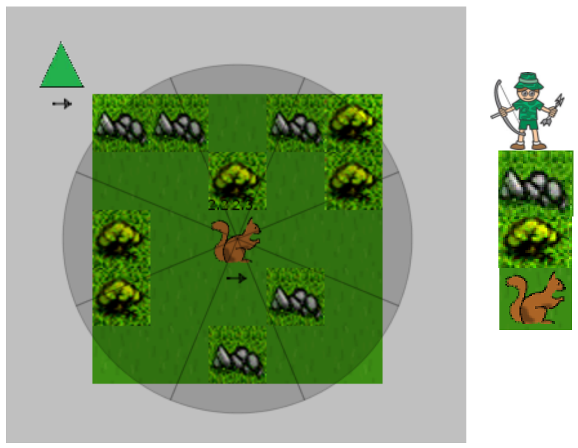


(c) Distribution of weapon properties

Figure 3.7: Distributions of the evaluation data.

The evaluation task used here is one specific configuration of the hunting domain based on some realistic considerations on the distribution of objects in the feature space, as well as the interaction model among objects. For example, prey with larger *sizes* tend to have higher *health*. Prey with lower *sensitivity to bush* tend to have higher *sensitivity to rock*. Weapons with higher *power* tend to have lower *accuracy*. The distribution of data has a 2-level hierarchical clustering structure as shown in Figure 3.7 for each categorization criterion.

The value of the state is determined by the probability of successfully hunting the prey. The general interaction model has been described in Figure 3.6. A specific scenario is visualized in Figure 3.8. The outcome in the simulation is computed by a complex function involving the interactions among several objects: weapon, prey, and obstacles. For example, to determine the probability of "not alerting the prey" (P1 in Figure 3.8), the numeric factor is the sum of *bush-sensitivity* and *rock-sensitivity*. If there is no bush or rock obstacles between the agent and the prey, then the corresponding factor is 0. The combined numeric value is transformed by a logistic function (represented as *f* in Figure 3.8) into the range of [0, 1] to represent the probability. Intuitively, when there are obstacles, and the prey is sensitive to the obstacles, the agent is more likely to successfully approach the prey. The distance factor negatively affects this probability since, for every step, there is some probability that the prey is alerted, so that the probabilities of not altering the prey form a geometric series indexed by the number of steps the agent has made within the detection range of the prey. To determine the probability of "shot on target" (P2 in Figure 3.8), the combined numeric factor is weapon



Distance: 3

Obstacles: bush at 1, rock at 2

Prey:

size	health	bush-sensitivity	rock-sensitivity
1	2	2	4

Weapon:

accuracy	power
5	2

Stochastic Environment:

1. Not alert the prey: **P1=*f*(distance, obstacles, sensitivities(prey))**
2. Shot on target: **P2=*f*(distance, accuracy(weapon), size(prey))**
3. Fatally wound the prey : **P3=*f*(distance, power(weapon), health(prey))**
4. Get reward: **P=P1 × P2 × P3**

Figure 3.8: A visualized example of a hunting scenario.

It shows the details about how the outcome is generated in the simulation. The function *f* is a logistic function.

accuracy + *prey size – distance*, with the sign reflecting the positive or negative influence by the corresponding factor. Similarly, for "fatally wound the prey" (P3 in Figure 3.8), the combined factor is weapon *power – prey health – distance*.

The coarse coding structure used for this task is similar to that shown in Figure 3.4, except that there are three category hierarchies, two for prey and one for weapon as shown in Figure 3.7. We use fixed hierarchical categories as input to the coarse coding system. Such structures were learned by the category formation process, and do not change during the value function learning. Learning rate of the algorithm is set at 0.3. Performance is evaluated by sample efficiency, measure by the improvement in performance with regard to the number of training episodes. Each trial involves incrementally training the agent and recording the average of 100 independent testing episodes at different points of training. The final results shown in the plots are the average of 100 such independent trials

3.3.2.2 Comparison with No Categorization

In Figure 3.9, we compare the learning performance of an agent using hierarchical categorization of objects with a baseline agent that uses the raw functional features without categorization. When there are only 9 unique instances, one from each prototype as shown in Figure 3.7, a 2-level 3-branch structure is sufficient, and the baseline agent is equivalent to only using the gird $G_{(2,2,2)}$, as explained in the section about sparse coarse coding. The results demonstrate that hierarchical categorization leads to faster learning because the coarser grids capture shared information and aid generalization. When there

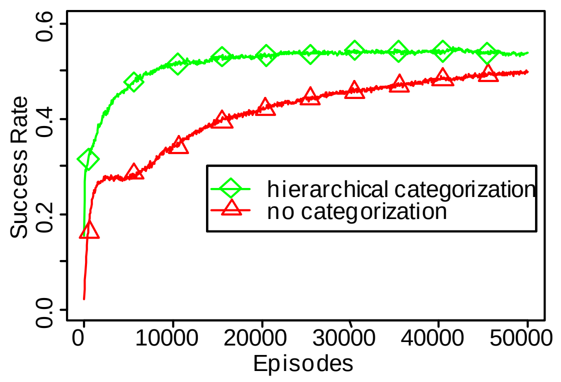


Figure 3.9: **Comparison with "No Categorization" by sample efficiency. Success rate is the ratio of successfully hunting the prey.**

are more subtypes under each of the 9 prototypes for each hierarchy, the performance of "hierarchical categorization" will not be affected because it utilizes all the general categories, while the baseline will be worse.

3.3.2.3 Comparison with Fixed-resolution Generalization

An advantage of using a hierarchy compared to flat categorization, such as the output from a k-means clustering algorithm, is the flexibility of representing categorical boundaries at different resolution levels, without forcing the system to make categorization decisions that are either over-general or over-specific. The next experiment tests whether such flexibility is naturally incorporated into our coding scheme for value function approximation. Figure 3.10 compares using hierarchical categorization to a baseline using single-level flat categorizations. In the baseline, the number of categories is set to 3 so that it is equivalent to only using grid $G_{(1,1,1)}$. Since such categorization does not capture the more subtle differences within each general category, it cannot improve performance beyond a certain point. We can choose to make a finer-grained flat categorization, which will lead to better asymptotic performance. However, that will inevitably lead to a worse improvement rate in the initial stage. In the extreme case, it will be the same as the performance generated by "no categorization" shown in Figure 3.9. In conclusion, using hierarchical categorization in our value function approximation scheme, reconciles the tradeoff between over-general and over-specific categorizations.

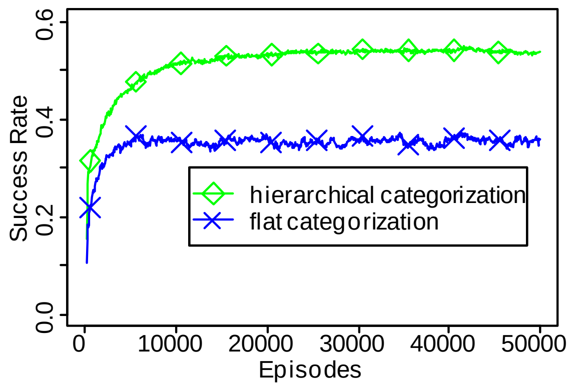


Figure 3.10: Comparison with "Flat Categorization" by sample efficiency. Success rate is the ratio of successfully hunting the prey.

3.3.2.4 Comparison with Suboptimal Hierarchy

To test if using prior knowledge in the form of functional roles and functional features can be beneficial, Figure 3.11 compares with a baseline using only a single functional role for prey with a monolithic category hierarchy based on all features of prey. The single hierarchy results in four levels with three branches at each level to represent all 81 types of prey – combinations of 9 sensing types and 9 physical types shown in Figure 3.7.

In Figure 3.11, the reason for poorer performance is because the single monolithic hierarchy does not reflect the true functional structure of the environment. The optimal structure requires two categorization hierarchies for prey, one based on sensing properties (*bush-sensitivity* and *rock-sensitivity*) and one based on physical strength (health and size). Our system supports encoding such useful structural domain knowledge via production rules.

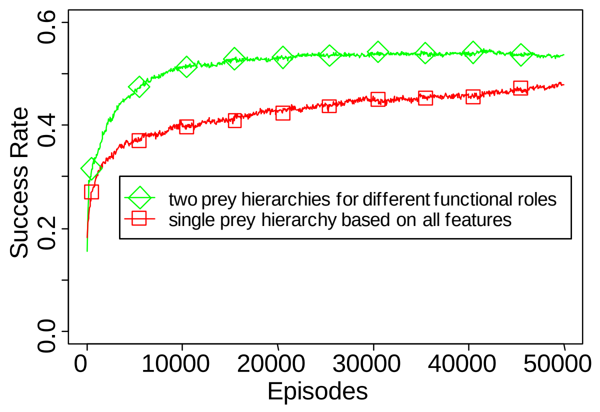


Figure 3.11: **Comparison with "Suboptimal Hierarchy" by sample efficiency. Success rate is the ratio of successfully hunting the prey.**

3.3.2.5 Summary of Results and Analysis

Comparing all three figures, even using the monolithic hierarchy (lower line in Figure 3.11) leads to faster initial learning than no categorization at all (lower line in Figure 3.9), as well as better asymptotic performance than flat categorization (lower line in Figure 3.10). This is because the monolithic hierarchy still captures certain amount of functional similarity. With the help of pre-specified functional roles and functional features, the learned category hierarchies more faithfully reflect structural constraints and leads to better performance.

3.4 Related Work

Our learning system combines sparse coarse coding and hierarchical structure for value function approximation with object-based state representations. Like coarse coding based value function approximation, our approach relies on learning piecewise constant local basis functions. The advantage of using local function approximators, compared to a global function approximator such as the multilayer perceptron (MLP) or linear regressors (Bradtke & Barto, 1996), is better stability and convergence when used in TD learning. This is because global function approximators are designed to fit relatively smooth functions, with linear models being the extreme case. Although MLP is a universal function approximator, its performance degrades when there are increasingly more intensive interference among nearby regions in the input feature space. Interference is the negative side-effect of generalization and can be catastrophic for learning arbitrarily non-smooth functions (McClelland *et al.*, 1995). Having sparser connections in MLP can reduce interference. However, French (1991) has noted that reducing overlap avoids catastrophic interference at the cost of a dramatic reduction in the exploitation of shared structure. Structured connections are required to balance the different needs.

Our approach can be viewed as one approach to reconciling interference and generalization by combining competitive learning (clustering), hierarchical representation and sparse-coding in a multi-layer network, which can be further regulated by rule-based symbolic domain knowledge. Competitive learning via hierarchical clustering generates symbolic categories, which serve as primitive structures to restrict interferences within local regions. On the other hand, sparse-coding with a hierarchical representation results in an emerging lattice structure, which regulates generalization by keeping the necessary connectivity, and at the same time minimizing interferences among unrelated regions. Such faster and more stable learning inevitably shifts the cost to somewhere else: compared to a fully connected MLP, more units (cells) are required in our system to achieve higher resolutions in value function approximation, although we have shown it is not a practical concern.

There are other approaches to combining hierarchical structure with local approximators, such as in kd-Q-Learning (Vollbrecht, 2000) and adaptive tile-coding (Whiteson *et al.*, 2007). However, these approaches lack the regulating structures that exist in our approach because they are not designed for object-based state representations. Furthermore, these approaches face the "curse of dimensionality". For environments involving objects, each object is represented by multi-dimensional features, and the total dimension of the feature space can easily become prohibitively expensive for learning if using unstructured state vectors that concatenate all the features. Our category learning system performs perceptual processing and dimension reduction: each hierarchy reduces the multi-dimensional subspace of corresponding functional features into a single dimension of hierarchical categories. Prior domain knowledge about functional roles and functional features helps to regulate such dimension reductions for object-based representations.

3.5 Discussion

In this chapter, we presented one specific implementation of our general approach. We focus on evaluating two novel features that are important for our system design. The first is to evaluate the successful integration of hierarchical representations in Soar-RL's existing coarse coding value function approximation scheme. The second is to confirm the importance of supporting multiple functional roles and functional features, which is part of our extension to COBWEB. We did not empirically evaluate the effect of category learning, because comprehensive analysis has been done in research related to COBWEB.

We created a complex domain that stresses the types of challenges we are trying to address. There are many parameters to configure in the evaluation environment, including the underlying distribution of data (Figure 3.7) as well as the model structure (Figure 3.6), and functions defining the probabilistic outcomes (Figure 3.8). The results are based on a particular setting of the parameters. In Chapter 5, we will systematically explore the space of data distribution in the evaluation. For example, when the distribution of data in the functional feature space has a more flat structure, the performance gains of using hierarchical structures will shrink. We can also increase the object diversity, which will result in deeper hierarchies. However, these quantitative changes are not expected to qualitatively change our conclusions.

Supporting the use of functional roles and functional features as a form of prior knowledge in our system can be practically useful with reasonable overhead. The observation is that only a relatively small set of functional roles and functional features are required to specify the general goals and basic needs of an agent. The benefit of providing such knowledge is that categorization more accurately reflects the functional similarities of objects without distractions from perceptual noise, and leads to significant improvement in RL.

3.6 Summary

We presented a novel, two-layer architecture for efficient value function approximation by integrating unsupervised hierarchical categorization with the existing RL system in Soar. Our system has two unique features. First, the value function approximation algorithm utilizes hierarchical structures to smoothly reconcile the tradeoff between over-specific and over-general categorizations, so that learning can be quicker and more accurate at the same time. Second, our system supports the use of prior domain knowledge about functional roles and functional features of objects to regulate learning. These properties are valuable for autonomous learning agents in a novel, complex, object-based environment. The empirical results have confirmed our hypothesis.

The major limitation of the learning system is that categorization relies on the set of functional features that are selected according to prior domain knowledge. Ideally, when such prior knowledge is not accurate, or unavailable, the agent should still be able to adaptively acquire functional categories based on the feedback from the environment, such as the reward value used to train the value function. In the next two chapters, we will address this issue from two different perspectives.

Chapter 4

Modeling Human Functional Category Learning

In the previous chapter, we showed that hierarchical categorization can be naturally integrated with a sparse coarse coding scheme for value function approximation. The algorithm automatically balances the transition from generalization to specialization. Meanwhile, it was also shown that the performance depends on how well the categorization hierarchy reflects the true functional relationship among objects. In the category learning system, categorization relies on a set of externally provided functional features, which begs the question how these functional features can be obtained beforehand and what should be done if they are not available to the system. In this chapter and the next, we will address this issue from two perspectives. In this chapter, we approach from a naturalistic perspective. We show that the learning system described in the previous chapter can adaptively select useful functional categories based on the feedback signal (reward value), even if the hierarchies are constrained by the predetermined feature set. Detailed analysis on the learning system reveals a computational model that can produce category learning behaviors qualitatively match with human. In the next chapter, we will approach from a functional perspective, which leads to a new algorithm formulated in a statistical optimization framework.

4.1 Background

We discussed category learning in the previous chapter from a purely functional perspective. There are many distinguishing characteristics of category learning in higher animals including human (Ashby & Maddox, 2005) and primates (Smith, 2010). The value of categorization is that it enables an individual to respond to a novel stimulus that resembles previously experienced stimuli with known responses. In this chapter, in addition to reconfirming the functionality of category learning in the context of an RL agent, we investigate several interrelated human category learning phenomena. We will show that our system generates three emergent effects found in human category learning: they are *basic level effect*, *context effect* and *expert effect*.

The basic-level effect is related to the notion of basic-level category as described by Rosch (1978). Consider the following two examples of abstraction hierarchies: furniture –> chair –> rocker –> cherry wood rocker, and machine –> vehicle –> car –> sedan –> Lincoln Town Car. The middle categories, chair and car, are basic categories, because they dominate both their subordinate and superordinate categories in terms how fast they can be recognized and how frequently they are used. It is generally believed that basic-level categories emerge unconsciously based upon interactions with our world.

The basic-level effect describes the phenomenon that categories at certain levels of abstraction dominate when there is no priming context. On the other hand, in certain task contexts, non-basic categories may become dominant. For example, in the context of considering what kind of cars to buy, the dominant categories are likely to be at the level of SUVs, sedans and wagons. In general, context effects may drive dominant categories to be either the subordinate or superordinate categories relative to the basic categories.

Finally, the level of expertise may also influence the dominant categories. For example, a car expert may immediately recognize the car being a Lincoln Town Car; a person familiar with chairs (such as a carpenter) may immediately recognize the chair being a cherry wood rocker. More learning experiences always drag down the level of dominant categories, and we call this phenomenon the expert effect (or simply the learning effect).

In summary, the traditional basic-level effect, context effect, and learning effects are all about category domination under different conditions. In this chapter, we develop a model of category domination based on the learning system introduced in the previous chapter, and analyze the three category learning effects using a concrete task.

4.2 Related Work

Category learning models have been traditionally focused on the basic-level effect, and there has been lack of computationally explicit models to explain how related category learning phenomena emerge from learning in a specific functional context, where a cognitive agent has to interact with the world to achieve some goals.

We have described our learning system in detail in the previous chapter. Our learning system consists of two components: a category learning system for hierarchical categorization, and a reinforcement learning system with sparse coarse coding value function approximation. In the following sections, we will briefly review works in human cognitive modeling that are related to our approach.

4.2.1 Hierarchical Category Learning

There is a long history of hierarchical category models that are inspired by human category learning. Quillian (1967) proposed the semantic network model, which can represent *isa* and *par-of* relationships among objects in a hierarchical structure. However, the semantic network model does not include a learning mechanism to build the structure. COBWEB (Fisher, 1987) is an algorithm that can incrementally form a hierarchical organization of categories. A previous version of the ICARUS cognitive architecture used a COBWEB-based system, called LABYRINTH for its declarative learning and memory (Langley *et al.*, 1991). Ambros-Ingerson *et al.* (1990) described a neurologically inspired hierarchical clustering algorithm, and Granger (2006) has demonstrated the plausibility of using such hierarchical clustering algorithm as a principled computational instruction for human cognition.

4.2.2 Reinforcement Learning

Hierarchical category learning provides the necessary representational basis, however the representation itself is insufficient for functional category learning because it has no direct connection to how the learned categories can be used. Another learning process is required to connect the category representations with the agent's intrinsic functional meanings. We consider reinforcement learning (RL) (Sutton & Barto 1998) as a candidate mechanism to establish such connections via incremental trial-and-error learning with feedback. The formulation of reinforcement learning as trial-and-error learning traces back to behaviorist psychology (Thorndike, 1991). The general actor-critic models of RL have been mapped to the basal ganglia structure in the brain (Barto, 1995). RL has also been considered as a model for human skill learning (Fu & Anderson, 2006). Cognitive architectures such as Soar (Laird, 2008) and ACT-R (Anderson *et al.*, 2004) both have a reinforcement learning mechanism. However, there has not been a computational model integrating category learning and RL in these cognitive architectures.

4.3 Evaluation

In this section, we will demonstrate how our learning system can qualitatively model several human category learning phenomenon and provides computational insights. In order to conveniently verify the qualitative match with human category learning behavior, we use the hunting task with the same data as shown in Figure 1.3 in Chapter 1. Different from Chapter 3, the task used here involves only pairs of prey and weapon objects, without the complications from obstacles and effects of shooting distance. The learning system used here is the same as in Chapter 3. We assume the hierarchies are fixed, and evaluate our learning system from two perspectives: how category learning affects RL and how RL affects category learning.

4.3.1 Category Learning Speeds RL

Hierarchical category learning can speed RL is one of the conclusions from Chapter 3, and we want to reconfirm it with the new task. Figure 4.1 compares the learning performance of using the hierarchical categorizations with a baseline that uses the leaf level nodes without generalization. This is the same comparison in Figure 3.9 in the previous chapter, although the task and data set are different. In this data set, there are two instances under each of the leaf nodes shown in Figure 1.3. For example, there are two instances of *Goat* that look different but have the same functional properties. Therefore, the size of the input space is: 16 (prey) times 12 (tools) equals 192. We evaluate the performance improvement during the course of learning. The agent is trained with uniformly distributed random samples from the input space with replacement. The learning rate is set at 0.1. For a given amount of training episodes, we evaluate the rates of correct decisions it makes if it follows the policy derived from the current value function. The final results are average from 300 independent trials. The result shows that the model successfully integrates hierarchical categorization to speed RL.

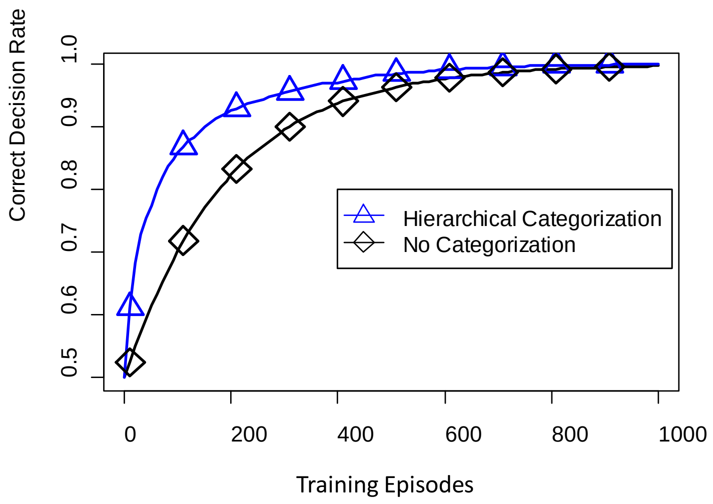


Figure 4.1: Comparison of learning with and without hierarchical categorization.

4.3.2 RL Shapes Category Learning

The main purpose of this chapter is to show how reinforcement learning influences category learning and result in behaviors that qualitatively match with human category learning. Therefore, we first define a category domination model to extract functionally salient categories from the learning system. Then we map the behavior of our learning system to several human category learning phenomenon based on the dynamics of these salient categories.

4.3.2.1 Category Domination Model

The detail of sparse coarse coding in our RL system has been described in Chapter 3. Figure 4.2 shows the same lattice structure as in Figure 3.4. The black dots in each grid spatially represent the specific input: hunting a deer with a crossbow. The gray areas represent the generalization effects when the more general rules fire. In this case, the agent receives a reward of +1 and each of the 16 rules participates in prediction and updating. Since a general rule (a larger cell) receives more training samples than a more specific rule (a smaller cell), it converges to the target value faster. On the other hand, the smaller cell will tend to compensate for the value in the context of the larger cell. In general, for a given input, there are multiple rules firing simultaneously, each coming from a different grid. We define the *dominant rule* as the rule with the highest absolute

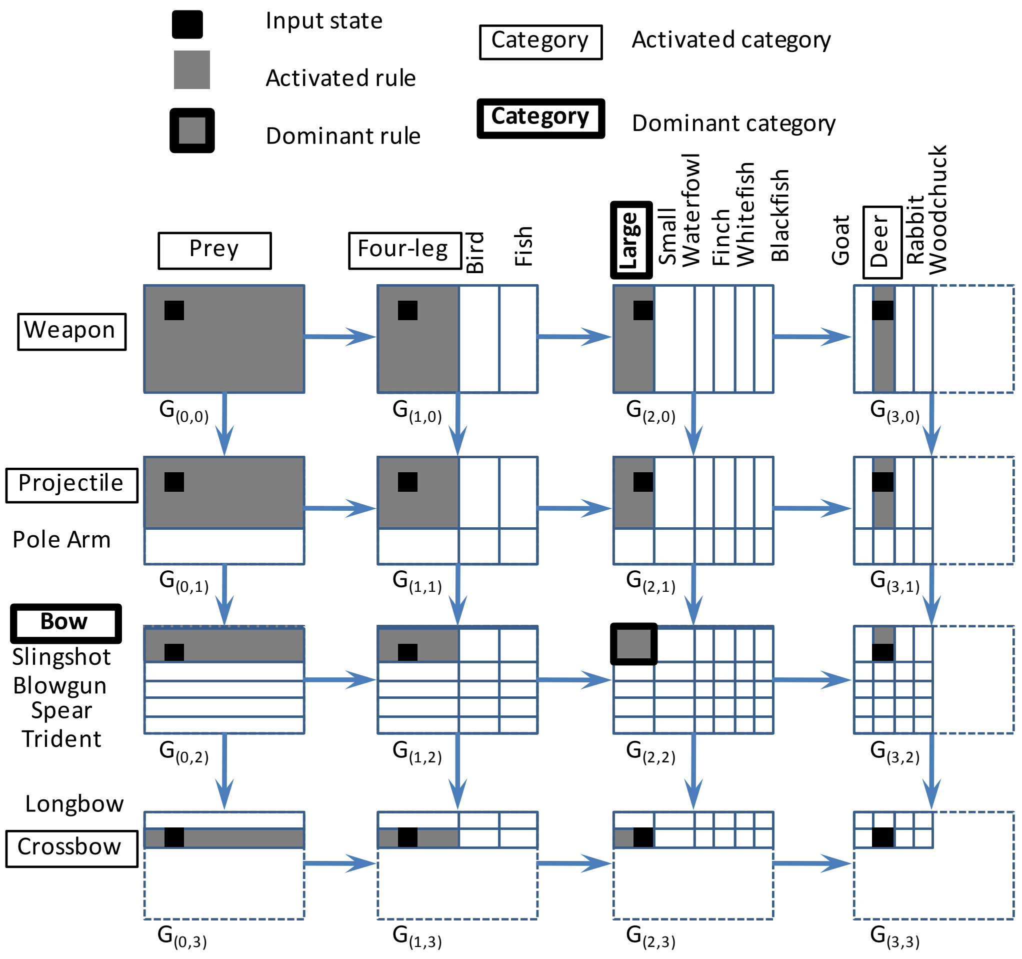


Figure 4.2: Coarse coding lattice and dominant categories.
The dominant rule and dominant categories are highlighted.

value, or equivalently the *winning cell* with largest magnitude in its weight:

$$WinningCell = ArgMax_C\{\|w(C)\|\}$$

Correspondingly, we define the *dominant categories* as the categories associated with the dominant rule. In the hunting task for a specific input, there are two dominant categories, one for prey and one for weapon. As highlighted in Figure 4.2, assume the rule testing *Large (Four-leg* animal*)* and *Bow*, the upper-left cell in grid $G_{(2,2)}$ is the dominant rule. It dominates all the rules that involve subtypes of *Large Four-leg* animal *(Deer and Goat)* or subtypes of *Bow (Crossbow and Longbow)* because those rules receive less training samples. It also dominates rules that involve more general categories, such as *Projectile weapon* and *Four-leg Animal*, because there are inconsistent updates for those rules that cancel out the updates. Consequently, the categories for *Large Four-leg* animal and *Bow* are the dominant categories in these particular situations. The general principle is that a rule simultaneously maximizing generality and consistency will dominate other rules. Intuitively, the associated dominant categories are more functionally salient than their superordinate and subordinate categories, since they are the sources contributing to most of the decisions made by the voting mechanism.

4.3.2.2 Basic-level Effect

We use the overall *domination rates* across all possible inputs as the measure of the context-free functional saliency of a category. We define basic level categories as those with the highest domination rates along a path. Figure 4.3a and 4.3b shows the domination rates of all the categories for prey and weapons after 1,000 training episodes. Since the ordering of inputs causes variations in the values/weights of the rules, we measure the mean domination rates across 300 independent learning trials, and the estimated standard errors for the means (not shown in the figure) are all less than 0.01. For example, the category for *Small Four-leg* animal dominates its superordinate and subordinate categories (including *Prey*, *Four-leg* animal, *Rabbit* and *Woodchuck*) in about 68% of all possible inputs. The category of *Rabbit* rarely dominates because its superordinate category perfectly captures the decision boundaries.

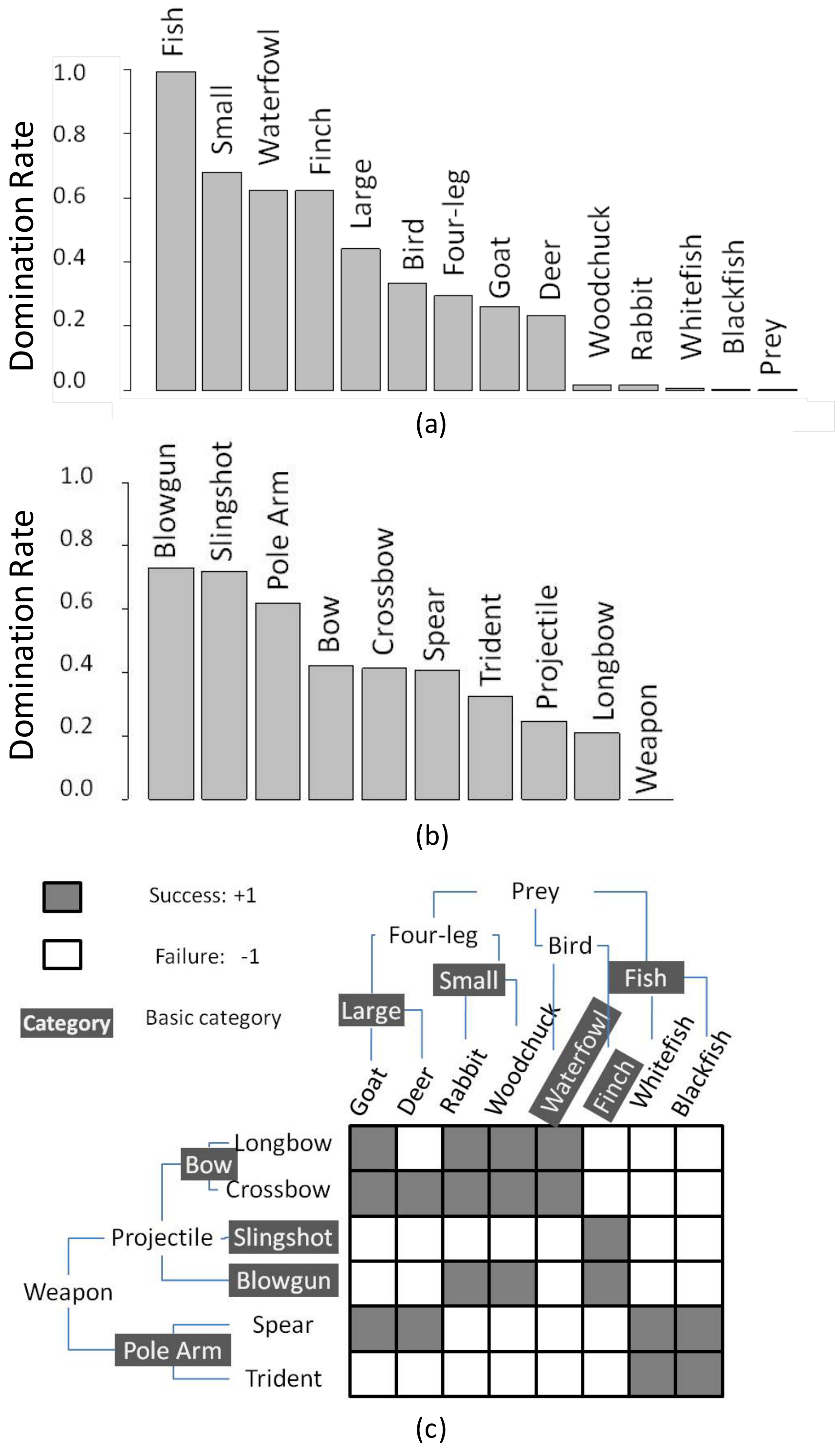


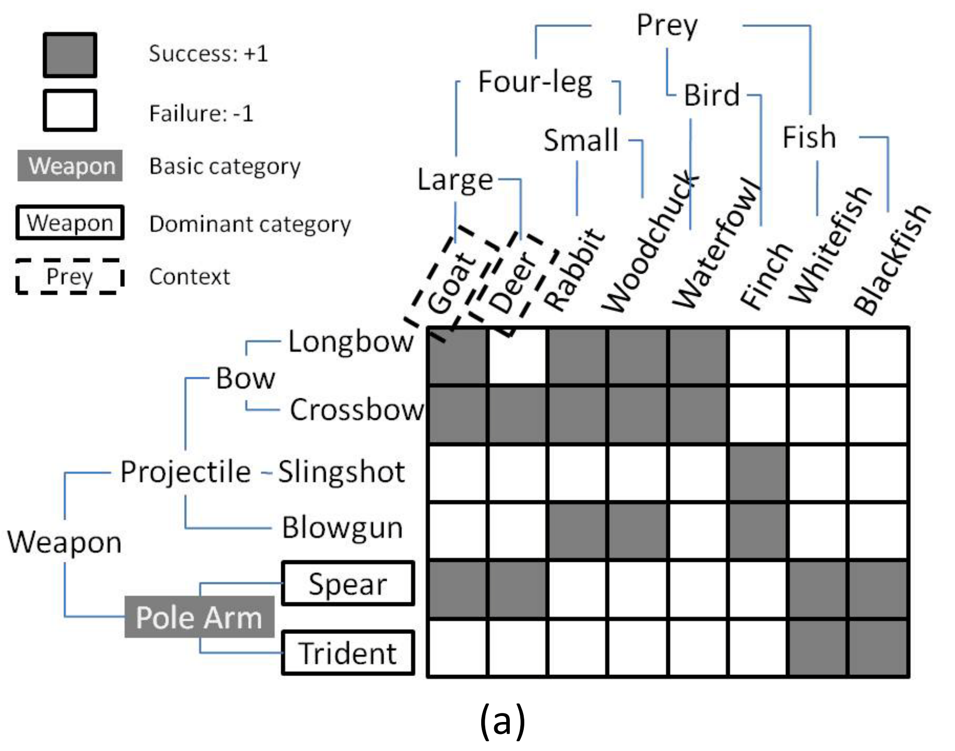
Figure 4.3: Illustration of basic-level effect.

(a) Domination rates of prey categories.(b) Domination rates of weapon categories. (c) Basic level categories are highlighted in shaded boxes.

Figure 4.3c highlights the basic-level categories in the hierarchies for prey and weapons. It exhibits qualitatively the same pattern as human category learning - basic categories appear in the middle of an abstraction hierarchy. As explained earlier, a rule that maximizes frequency and consistency will dominate, and the corresponding categories are dominant categories. More abstract categories are experienced with higher frequency, while more specific categories have higher consistency. Therefore, the maximum will generally appear in the middle. Look at some specific examples in Figure 4.3c, although *Bird* has higher frequency than its subordinate categories, it has low consistency. *Fish* has higher frequency than its subordinate categories with the same level of consistency, and therefore is the dominant category.

4.3.2.3 Context Effect

The basic-level effect describes the property of overall domination rates across the distribution of samples. For a specific instance (context), non-basic-level categories (the ones not having the highest overall domination rates) may dominate. Figure 4.4 shows two examples of context effect, where the shaded boxes represent basic-level categories, the solid line boxes represent the dominant categories of weapon, and the dotted line boxes represent the prey categories serving as the context. Figure 4.4a shows that the sub-basic-level category *Spear* and *Trident* dominates the basic level category *Pole Arm* when the prey (serving as the context) being hunted is *Goat* or *Deer*. Figure 4.4b shows that the superordinate category *Projectile* (weapon) dominates the three basic-level categories when the context is hunting *Fish*.



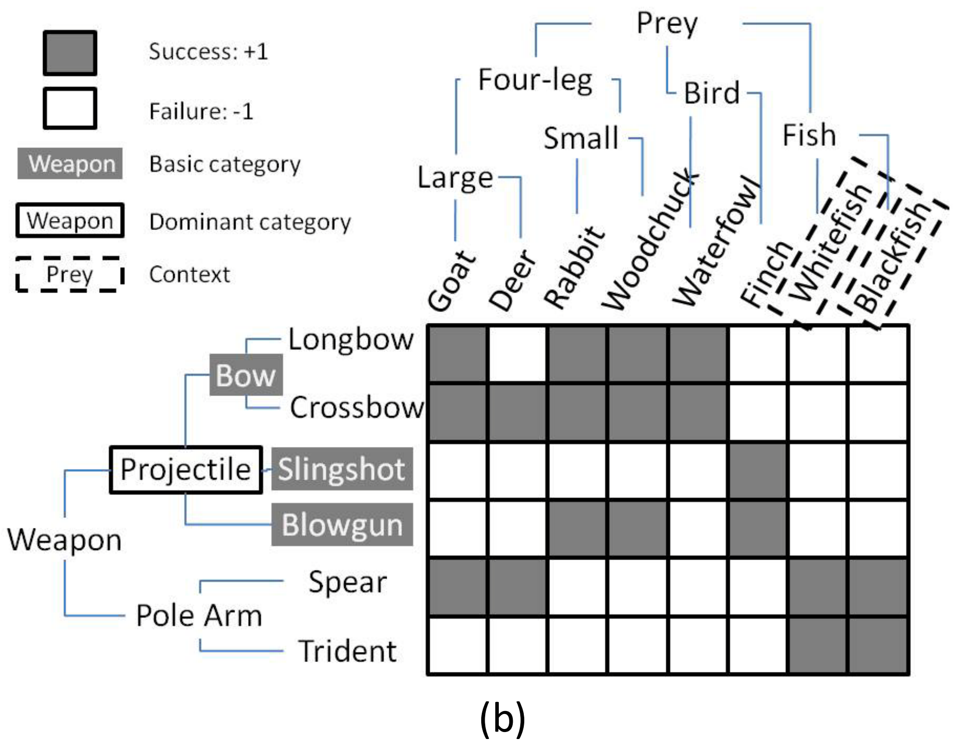


Figure 4.4: Illustration of context effect.

(a) Dominant categories are more specific than the basic level category. (b) Dominant category is more general than the basic level categories.

4.3.2.4 Expert Effect

The expert effect concerns the dynamic changes of dominant categories during learning. Figure 4.5 shows the dynamics of domination rates during learning 1,000 samples. Figure 4.5a shows the dynamics of the domination rates for the 14 prey categories. Figure 4.5b and 4,5d are the same as Figure 4.3a, showing the domination rates after 1,000 training. Therefore, the heights of the bars in Figure 4.5b match with the heights of corresponding curves' end point in Figure 4.5a. Similarly, Figure 4.5c and Figure 4.5d shows the domination dynamics for the 10 weapon categories. The trend is that the more general categories initially have higher domination rates because they cover more inputs and are trained with higher frequencies. More specific categories that have functional properties distinctive from other members under the same super-ordinate category have increasing domination rates as more and more experience is gained (such as the two subtypes of

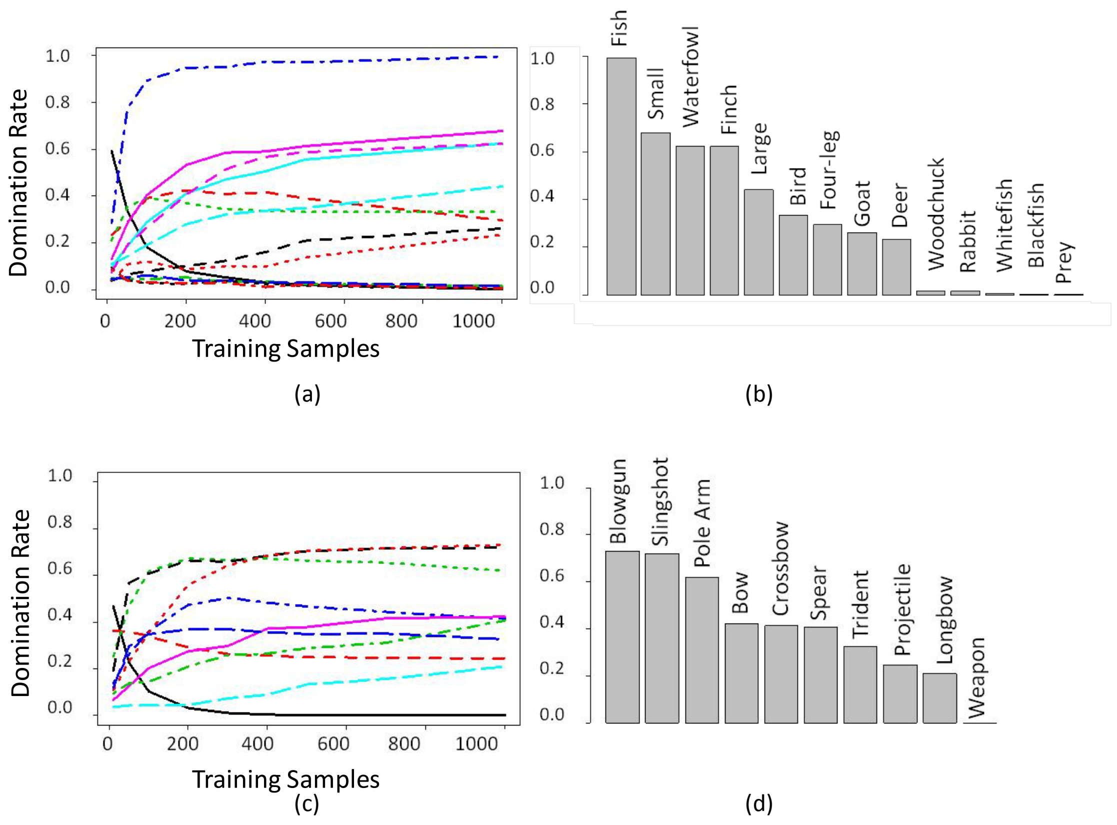


Figure 4.5: Dynamics of domination rates.

(a) and (c) show the dynamics of domination rates during learning. (b) and (d) show the domination rates after 1,000 training samples. (a) and (b) are for prey categories. (c) and (d) are for weapon categories.

birds). Meanwhile, their superordinate categories become less and less dominant (such as the general category *Prey*, *Four-leg* animal, and *Bird*) because the inconsistency tends to cancel out multiple updates. On the other hand, a perceptual category that does not have any functional differences from other members under the same superordinate category does not arise as a functionally salient category (such as *Rabbit*, *Woodchuck* and the two subtypes of *Fish*).

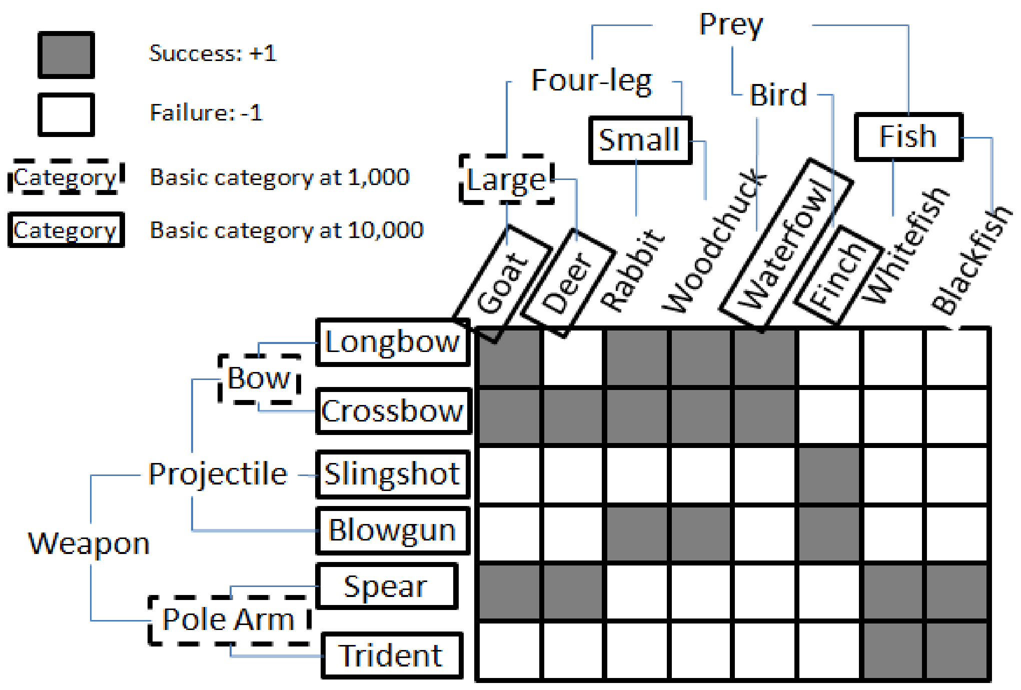


Figure 4.6: Illustration of expert effect.
Basic level categories are pushed down after more training.

After more training samples, some of those sub-basic-level categories at 1,000 training episodes may gain more domination rates. In Figure 4.6, dashed line boxes represent basic level categories after receiving 1,000 training samples, and solid line boxes represent basic level categories after receiving 10,000 training samples. The figure shows that when the agent receives more training samples and becomes an expert hunter, the basic level categories for *Large (Four-leg Animal)*, *Bow* and *Pole Arm* are further "pushed" down to more specific levels.

4.4 Discussion

Traditional cognitive theories of category learning include two competing views: the prototype view **(**Rosch, 1973**)** and the exemplar view (Medin & Schaffer, 1978). The prototype view is based on the principle of cognitive economy (Rosch, 1978) and is supported by the existence of abstract category representation, such as linguistic lexicons and visual imagery, such as the word dog, cat and visual depictions of prototypical dogs and cats. However, there has been a shift of favor from the prototype view towards the exemplar view because exemplar models provide superior empirical results in a variety of experimental settings (Nosofsky & Zaki, 2002). A practical concern about the prototype view is that a prototype will fail to retain certain information that might be important for future decision making. More recent models reconcile the two extreme forms and rely on representations at multiple abstraction levels (Vanpaemel & Storm, 2008; Love *et al.*, 2004).

Our model is consistent with both the prototype and exemplar views. In addition, it explicitly models the learning process, and can deal with the more challenging situations where the input states involve multiple objects (such as prey and weapons). In terms of decision making, our model is more like exemplar based models, where the agent acquires information about specific inputs, and then makes generalizations to novel inputs based on perceptual similarity. In terms of category abstraction, our model agrees with prototype models. In particular, it predicts a similar trend as in the phenomenon of basic-level category (Rosch, 1978) where the most prominent categories reside in the middle of a categorization hierarchy.

Furthermore, our model predicts that category domination is context specific. For example, in Figure 4.4a, *Pole Arm* is the dominant category if the specific context is hunting *Fish* (all subtypes of *Pole Arms* are good for fishing). In a different context, however, *Spear* and *Trident* will dominate if the specific context is hunting a *Deer*. Our model explicitly supports the hypothesis that the “context-free” basic level categories, as described by Rosch, are the overall effects acquired across multiple functional contexts. Since the everyday activities related to common objects are largely the same across individuals, the context-free basic-level categories appear to be consistent across different cultures, as manifested by the consistency in common vocabularies from different natural language systems.

Our model does not involve a dedicated process of selecting functionally meaningful categories. Selection is achieved as an emerging by-product of the RL process. Our model cannot explain certain types of category learning that rely on deliberate reasoning or higher degrees of abstractions such as analogy. Such deliberate categorization is better described by rule based category learning models (Rouder & Ratcliff, 2006), or analogical reasoning processes such as in the structure-mapping engine (SME) (Falkenhainer *et al.*, 1989).

4.5 Summary

In this chapter, we present the first computational model that integrates hierarchical category learning and RL in a general cognitive architecture. The unique feature of this model is that it simultaneously captures how category learning affects behavior adaptation, and how behavior adaptation influences category learning in a functional context. Furthermore, our model supports the more challenging tasks involving multiple objects, which is common in object-based environment. We have shown that our learning system is able to adaptively discover functionally meaningful categories even it is constrained by fixed innate perceptual features for categorization. The behavior of the learning system qualitatively matches several related prominent human category learning phenomena: the basic level effect, the context effect and the expert effect. Although the Soar-RL model has been successfully applied to match animal behavior data (Wang & Laird, 2007), further empirical experiments are required to confirm its validity in our category learning model.

Chapter 5

Joint Optimization of Functional Categories and Value Function

In Chapter 3 and Chapter 4, we presented a learning algorithm that consists of two separate components: a hierarchical clustering component and a sparse coarse coding based value function approximation component. A major limitation of the algorithm is that the learned hierarchical categorization structure is constrained by a set of preselected features to reflect the intended categorization criteria. For example, in Chapter 3, prey are categorized by two criteria. One criterion is based on physical properties including two features: size and health. The other is based on sensing properties based on two features: bush-sensitivity and rock-sensitivity. In Chapter 4, although functionally salient categories are adaptively selected in the process of learning the value function, the hierarchical structure is still constrained by the fixed innate perceptual features, which limits the flexibility of the learning system.

In this chapter, we present a new algorithm based on a probabilistic model with latent class variables. Consistent with our general approach, the algorithm combines piecewise approximation with hierarchical categorization. Different from the previous two-step algorithm, in this new algorithm, the hierarchical functional categories and utility value approximation are jointly optimized.

In the following sections, we first revisit the background. Then we introduce the latent class model view for the problem of value function approximation in object-based environments, where the latent class variables are unobserved functional categories of the objects. Then we provide a hierarchical approximation learning algorithm with empirical evaluations. Finally, we discuss alternative approaches at a more abstract level and draw conclusions.

5.1 Background

The general problem we are considering is to learn a function whose inputs consist of objects, and the output is either a numeric or a symbolic value. Since we make no assumption about the underlying form of the function, the most obvious approach is to partition the input space and approximate the values locally. The local approximator for each piece can be in a very simple form, such as a linear function or even a constant. Accordingly the approximator is called piecewise linear or piecewise constant. The general piecewise approximation approach has been widely applied in engineering problems where the underlying function's form is complex. For example, finite element methods (Johnson, 1987), use such a technique for finding approximate solutions of partial differential equations.

In our situation, the inputs to the function consist of objects which may have many perceptual features, while only a subset of them are relevant to the output value. For such problems in high dimensional space, the critical decision is about how to effectively partition the input space to achieve efficient learning, especially when training data is scarce during the initial learning stage. Furthermore, our hypothesis is that when certain regularities (which will be characterized and evaluated later) exist, hierarchically partitioning objects can achieve both sample efficiency, and supports computational efficiency in an incremental learning algorithm. As demonstrated in Chapter 3, the most effective way is to partition objects into functionally distinctive categories based on their intrinsic functional properties. However, we cannot generally assume the functional properties are directly observable to the agent as a set of functional features as in Chapter 3. This presents a challenge to creating efficient learning algorithms because effective partitioning of the input space needs to be based on unobservable functional properties. Therefore, a different approach has to be taken. Instead of doing category learning and value function learning separately, the solution lies in combining category formation and value function approximation into a single structured optimization problem, assuming the perceptual features and the utility values are always observable.

In the following section, we first describe a structured probabilistic model of the problem, which attempts to capture unobservable functional properties via hidden discrete variables. We then formulate the value function approximation problem as an optimization problem based on the maximum likelihood framework. Then a hierarchical approximation approach is presented to achieve efficient incremental learning by utilizing a combinatorial hierarchical structure similar to the approach described in Chapter 3. Finally, a specific incremental learning implementation is presented and empirically evaluated with systematically generated synthetic data. The data probes varying degrees of object diversity as well as regularities that are expected to present in object-based natural environments.

5.2 A Probabilistic Latent Class Model

In object-based environments, objects are independent entities with their own perceptual features and intrinsic functional properties. Multiple objects can interact with each other to produce an outcome, which will be associated with a utility value (positive or negative). The goal is to learn the mapping from the objects' perceptual features to the outcome utility value. A natural way to model this function is to represent each type of object as a finite number of functionally distinctive categories. For example, the number of functionally distinctive categories of prey can be very large, but there are a fixed number of categories, beyond which the functional distinctions are too small to be relevant for decision making. For instance, a tiger and an elephant are very different prey, while the distinction between an African elephant and an Asian elephant, or the distinction between an Amur tiger and a Bengal tiger is negligible.

In Figure 5.1, the value function that maps prey and weapon to an outcome value is represented by a probabilistic graphic model with simple dependency structures. Hidden variables are represented as empty circles, and observable variables as filled circles. The distinctive functional categories for each type of object are represented by the discrete hidden class variables for weapon (c_1) and prey (c_2). Perceptual features are dependent on the class variable of the corresponding object. The output value depends on the combination of prey class and weapon class.

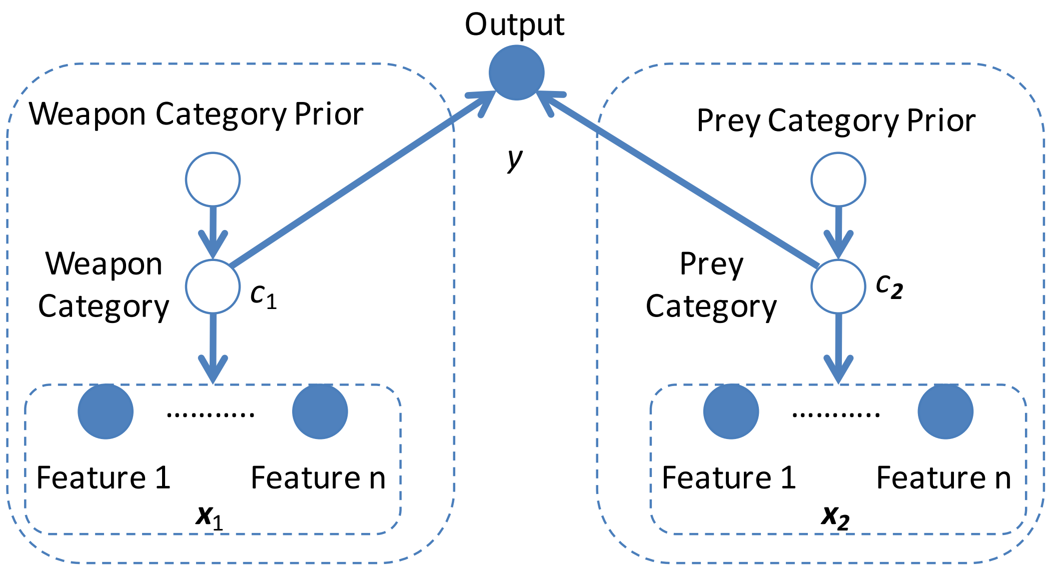


Figure 5.1: Latent class model of the environment.
Filled nodes are observed quantities, and empty nodes are unobserved hidden variables.

The three components of the probabilistic model are formally represented in equations (5.1) to (5.3). c_i represents the category for object i, and **c** represents the set of categories for all objects. We use ***x*** to denote the perceptual features of the objects, which are the input variables to the value function. $\boldsymbol{x}_i$ represents the perceptual features associated with object i. Finally, we use y to denote the output variable. The equations represent the probabilistic density function (PDF) of the distributions for these random variables. Equation (5.1) represents the background distribution of the categories for each object. Equation (5.2) represents the distribution of perceptual features for each object conditioned on the category. Equations (5.3) represents the distribution of output value conditioned on combinations of object categories. At this point, there is no assumption about specific functional forms of the PDFs.

$$P(\boldsymbol{c}) = \{P(c_i) \mid i \in [1, m]\} \qquad (5.1)$$

$$P(\boldsymbol{x} \mid \boldsymbol{c}) = \{P(\boldsymbol{x}_i \mid c_i) \mid i \in [1, m]\} \qquad (5.2)$$

$$P(y \mid \boldsymbol{c}) = \{P(y \mid [c_1, c_2 \ldots, c_m])\} \qquad (5.3)$$

5.2.1 Prediction

Equations (5.4) to (5.6) describe how the model predicts the output value when input perceptual features are provided. The category variable, c_i, for each object is a discrete variable with finite values as discussed earlier. We further assume that the membership of a category is exclusive, i.e., an object must belong to one and only one of the finite discrete classes. According to the model, the categories are first determined based on the posterior probability of each category given perceptual features, as in equations (5.4) and (5.5). Once the categories are determined, the output can be predicted based on the conditional distribution associated with the combination of categories as in equation (5.3). The entire prediction process is represented by equation (5.6), which is a function of the three component distributions in equations (5.1) to (5.3).

$$P(c \mid x)P(x) = P(c,x) = P(c)P(x \mid c) = \prod_{i=1}^{M} P(c_i)P(x_i \mid c_i) \tag{5.4}$$

$$P(c \mid x) \propto P(c)P(x \mid c) \tag{5.5}$$

$$P(y \mid x) = P(y \mid c = \arg\max_{c} P(c \mid x)) = P(y \mid c = \arg\max_{c} P(c)P(x \mid c)) \tag{5.6}$$

5.2.2 Maximum Likelihood Optimization Criterion

The maximum likelihood principle states that the choice of parameters (Θ), of a probabilistic model that gives higher likelihood to the training data, is more likely to give higher prediction accuracy (assuming uniform priors). In this learning problem, since the key is to find the most effective partition of the input space via the hidden class variables, we map the parameter space to the sample partition/categorization space of the training data $\{x, y\}$. The partition space is defined by all the possible partition/categorization rules. A categorization rule is a function mapping each instance to corresponding categories. As represented by equation (5.7), with category assignment determined, the parameters of the three component distributions, as represented by equations (5.1) to (5.3), can be estimated, again by the maximum likelihood principle. In this problem of predicting *y*, the objective is to maximize the likelihood of *y* conditioned on *x* for all training data. Under the assumption of independent training samples, the likelihood function of Θ can be obtained by multiplying the predicted probabilities of each output value (y_n) given the input features (x_n), as represented by equation (5.8).

We next derive the likelihood function in equation (5.8) as a compact form of the probabilistic density functions (PDFs) of the three component distributions in equations (5.1) to (5.3). Getting the compact form is critical for implementing the incremental learning algorithm later, because it makes it possible to directly evaluate the likelihood function instead of examining all the training samples as in equation (5.8).

$$\Theta = f : \{\boldsymbol{x}_n, y_n\} \to \boldsymbol{c} \Rightarrow \left\{P(\boldsymbol{c}), P(y \mid \boldsymbol{c}), P(\boldsymbol{x} \mid \boldsymbol{c})\right\} \qquad (5.7)$$

$$L(\Theta \mid \boldsymbol{x}, y) = \prod_{n=1}^{N} P(y_n \mid \boldsymbol{x}_n, \Theta) \qquad (5.8)$$

In equation (5.9), we rewrite the logarithm of the likelihood function as the sum of the log likelihood over all possible values of *y* and predicted categories *c* weighted by their joint probabilities. For simplicity of later derivations without loss of generality, equation (5.9) assumes *y* is a discrete variable. We use the superscript y^i to represent the *ith* value of *y*, and c^j to represent the *jth* value of category *c*. If otherwise *y* is continuous, the sum over *i* in equation (5.9) will be replaced by an integration operator. Also for simplicity, we use *c* to denote the combination of all the objects' classes, and use $\hat{c}$ to denote the predicted classes which consists of classes with the highest posterior probabilities computed by equation (5.6).

$$\lg L(\Theta \mid x, y) = \sum_{n=1}^{N} \lg P(y_n \mid x_n, \Theta) = \sum_{n=1}^{N} \lg P(y_n \mid \hat{c} = \arg\max_{c} P(c \mid x_n, \Theta)) \qquad (5.9)$$

$$\lim_{N\to\infty} \frac{1}{N} \lg L(\Theta \mid x, y) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \lg P(y_n \mid \hat{c} = \arg\max_{c} P(c \mid x_n, \Theta))$$

$$= \sum_{i} \sum_{j} P(y = y^i, \hat{c} = c^j \mid \Theta) \lg P(y = y^i \mid \hat{c} = c^j, \Theta)$$

$$= \sum_{i} \sum_{i} P(y = y^i, \hat{c} = c^j) \lg P(y = y^i \mid \hat{c} = c^j)$$

We further derive the form of the first component in the last line of equation (5.9), which is the joint probability of *y* and predicted category *ĉ*. In equation (5.10), we use the notation c^l -> c^j to represent the event that an instance comes from the *lth* category, but is predicted as the *jth* category. This quantity, although not in a closed form, can be evaluated efficiently. For simple distribution forms, such as in a naïve Bayes model, this quantity can be computed exactly by enumerating all possible values of *x*. For arbitrary distribution forms, a Monte Carlo approach can always be used to do the estimations with bounded computational cost. Here, we also assume that the estimated probability density functions (PDFs) are good approximations of the true distributions.

$$
\begin{aligned}
P(y=y^i,\hat{c}=c^j) &= \sum_k\sum_l P(y=y^i,\hat{c}=c^j,x=x^k,c=c^l) \qquad (5.10)\\
&= \sum_k\sum_l P(y=y^i \mid \hat{c}=c^j,x=x^k,c=c^l)\\
&= \sum_k\sum_l P(c=c^l)P(y=y^i \mid c=c^l)P(x=x^k \mid c=c^l)P(\arg\max_c P(c \mid x^k)=c^j)\\
&= \sum_l P(y=y^i \mid c=c^l)P(c=c^l)\sum_k P(x=x^k \mid c=c^l)P(\arg\max_c P(c \mid x^k)=c^j)\\
&= \sum_l P(y=y^i \mid c=c^l)P(c=c^l,\hat{c}=c^j)\\
&= \sum_l P(y=y^i \mid c=c^l)P(c^l \to c^j)
\end{aligned}
$$

where $P(c^l \to c^j)$ represents the probability that an instance from c^l is predicted as c^j

We combine equation (5.10) with (5.9), to get the final compact form in equation (5.11). We rewrite both $P(y=y^i|c=c^l)$ and $P(y=y^i|\hat{c}=c^j)$ as $P(y^i|\ c^l)$ and $P(y^i|\ c^j)$, since we assume the estimated probabilities and the true probabilities are the same.

$$\lim_{N\to\infty}\frac{1}{N}\lg L(\Theta\,|\,x,y)=\sum_i\sum_j\sum_l P(y=y^i\,|\,c=c^l)P(c^l\to c^j)\lg P(y=y^i\,|\,\hat{c}=c^j) \quad (5.11)$$

$$=\sum_l\sum_j P(c^l\to c^j)\sum_i P(y^i\,|\,c^l)\lg P(y^i\,|\,c^j)$$

$$=\sum_l\sum_j P(c^l\to c^j)\sum_i\left(P(y^i\,|\,c^l)\lg P(y^i\,|\,c^j)-P(y^i\,|\,c^l)\lg P(y^i\,|\,c^l)+P(y^i\,|\,c^l)\lg P(y^i\,|\,c^l)\right)$$

$$=\sum_l\sum_j P(c^l\to c^j)\sum_i\left(P(y^i\,|\,c^l)\lg\frac{P(y^i\,|\,c^j)}{P(y^i\,|\,c^l)}+P(y^i\,|\,c^l)\lg P(y^i\,|\,c^l)\right)$$

$$=\sum_l\sum_j P(c^l\to c^j)\sum_i P(y^i\,|\,c^l)\lg P(y^i\,|\,c^l)-\sum_l\sum_j P(c^l\to c^j)\sum_i P(y^i\,|\,c^l)\lg\frac{P(y^i\,|\,c^l)}{P(y^i\,|\,c^j)}$$

$$=-\sum_l P(c^l)E(P(y\,|\,c^l))-\sum_l\sum_j P(c^l\to c^j)D_{KL}(P(y\,|\,c^l)\,||\,P(y\,|\,c^j))$$

where

$E(P)$ represents the entropy of distribution P

$D_{KL}(P\,||\,Q)$ represents the Kullback-Leibler divergence from ditribution P to Q

for Gaussian distribution $N_{\mu,\sigma}$ with mean μ and standard deviation σ:

$$E(N_{\mu,\sigma})=\frac{\ln(2\pi e\sigma^2)}{2}$$

$$D_{KL}(N_{\mu_1,\sigma_1}\,||\,N_{\mu_2,\sigma_2})=\frac{(\mu_1-\mu_2)^2+\sigma_1{}^2-\sigma_2{}^2}{2\sigma_2{}^2}+\ln(\sigma_2/\sigma_1)$$

5.2.3 Intuition of the Optimization Criterion

Equation (5.11) shows that maximizing the likelihood function is equivalent to minimizing the sum of two components. The first is the total entropy of the output value's conditional PDF weighted by the prior probability of corresponding discrete classes. The second is the estimated total "confusion rates", P(c^l -> c^j), between pairs of the discrete classes weighted by the Kullback-Leibler divergence (KL divergence) (Kullback & Leibler, 1951) between the two corresponding PDFs. The first component about entropy is the same as in the standard Decision Tree (Quinlan, 1986) splitting criterion. The intuition is that a good functional classification should always separate instances with different output values. The second component about confusion rate weighted by KL divergence is new. This is because in contrast to a Decision Tree's deterministic splitting, our model uses probabilistic classifications as shown in equations (5.4) to (5.6). If the two classes has high confusion rates, there will be more errors in the predictions of the output value – especially when the confused pair has very different output values. This is not an assumption of a standard Decision Tree, which makes deterministic splits and assumes zero confusion rates.

To gain further understanding of the intuition behind the optimization criteria represented by equation (5.11), we compare an algorithm using the optimization criterion in equation (5.11) to two baselines. One baseline is a simple Naïve Bayes model, and the other is a Decision Tree. We use toy data with binary input and output to evaluate the performance. The simple Naïve Bayes model directly classifies instances based on the output value (that is why it is called simple in addition to being Naïve Bayes), and therefore minimizes the first component about the output variable’s entropy to be zero (all the samples in the same class have the same output value). The Decision Tree is restricted to deterministically split each perceptual feature (therefore always guaranteeing zero estimated confusion rates), and among all possible splits, the one that minimizes the first entropy component is chosen. From the perspective of equation (5.11), a standard Decision Tree is a special case assuming a single-attribute sparse model without noise, which could be a poor choice in high dimensional and noisy environments. There are extensions of decision tree algorithms to allow for fuzzier splits (Yuan & Shaw, 1995) and multivariate splits (Brodley & Utgoff, 1995). We will not consider those specific algorithms here, because they are designed to learn a single hierarchical classification for the entire state feature vector. In order to learn the object-based compositional structure as shown in Figure 5.1, the algorithm must be able to simultaneously construct multiple category hierarchies, one for each object. Therefore, we design a new incremental learning algorithm, and the advantages of doing multivariate and fuzzy splitting are naturally included in our optimization criterion in equation (5.11).

We compare the algorithms using three different data sets with binary input features and output values. The simple Naïve Bayes learner classifies the instances into two classes solely based on the output value. The Decision Tree splits the data into two classes using on the standard entropy based splitting criteria, and recursively splits until either all the input features are the same (so that no further split is possible) or all the output values are the same within the partitions (so that there is no need to split). Our algorithm recursively groups the instances into two classes as in the Decision Tree, and it assumes independent feature distribution as in a naïve Bayes model. The performance is evaluated in terms of sample efficiency – improvement in prediction accuracy as the number of training instances grow. Prediction accuracy is evaluated by using 100 independent random samples to estimate the prediction accuracy at fixed intervals, until reaching 200 training samples. The final results are the average of 100 such independent runs. In this section, we only focus on the effect of the optimization criterion and omit the detailed procedure of our learning algorithm, which will be discussed in the next section.

As shown in Figure 5.2, the first data set is drawn from an XOR function which is nonlinearly separable. The naïve Bayes classifier fails to learn the XOR function because it builds two naïve Bayes models, one for class *a*, and one for class *b*. The two models have the same aggregated feature distribution under the independent features assumption. Therefore, the naïve Bayes classifier is unable to discriminate the two classes. Initially, the naive Bayes classifier is able to make better than random predictions because it can

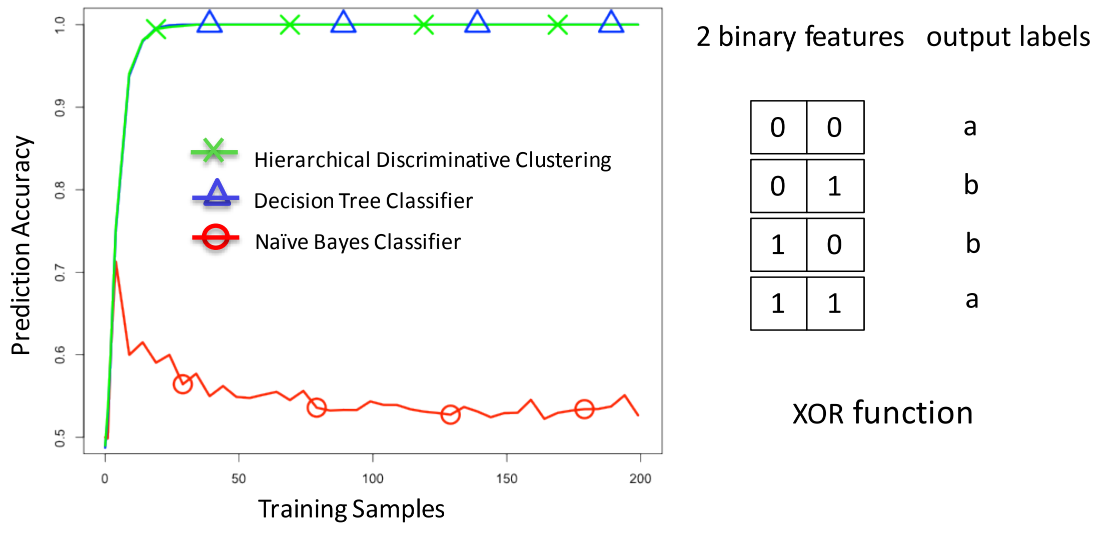


Figure 5.2: Comparison of different classifiers using the XOR function.

correctly predict the output for some instances when the learner has only accumulated a few training samples. For example, when there is only one training instance, it can give correct prediction for ¼ of the situations (there are totally 4 unique samples). After more instances are accumulated for each class model, the performance degenerates to randomness. Both our algorithm and decision tree classifier avoid such situations by always keep highly discriminative partitions.

The second data set is drawn from a linearly separable function with multiple noisy binary features as shown in Figure 5.3. There are two classes: class *a* consists of 10 noisy 0s, and class *b* consists of 10 noisy 1s. For linear separable classes, naïve Bayes models can perfectly construct the decision boundary to discriminate the two classes from each other. For the decision tree classifier, however, its iterative feature splitting strategy becomes inefficient when the decision boundary lies in a hyper-plane spanning multiple feature dimensions. Our algorithm is not affected because it is not restricted to univariate boundaries.

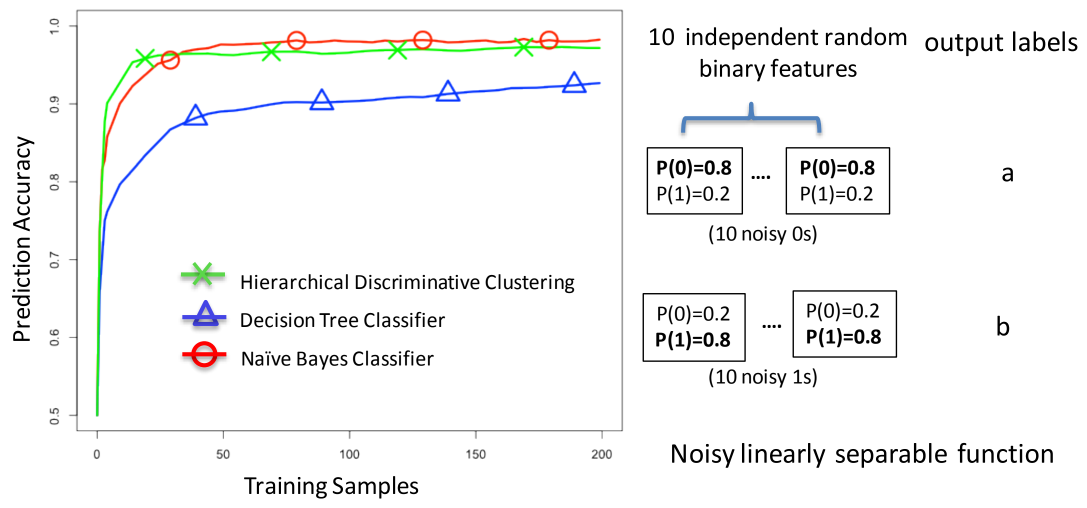


Figure 5.3: Comparison of different classifiers using a noisy linearly separable function.

Figure 5.4 shows the comparison on the third data set, which is drawn from a noisy XOR function that combines the characteristics from the previous two data sets. Our algorithm outperform both the decision tree classifier and the naïve Bayes classifier.

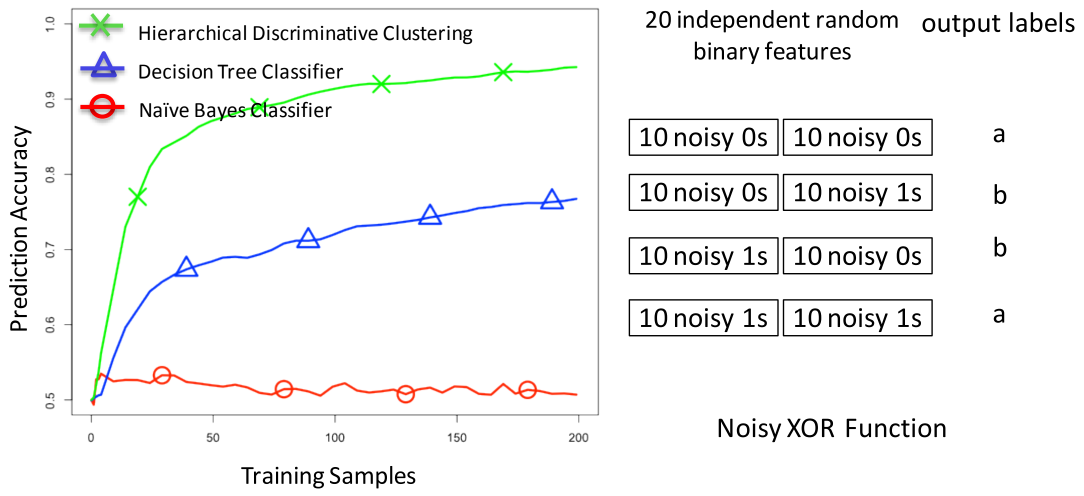


Figure 5.4: Comparison of different classifiers using a noisy XOR function.

The results are summarized in Table 5.1. The simple Naïve Bayes classifier is sensitive to nonlinearly separable classes, because it simply relies on the output labels without actively searching for discriminative boundaries. Both the decision tree classifier and our algorithm are robust against such situations by employing active searching in the classification space. However, the decision tree classifier is sensitive to noisy multivariate boundaries that span across multiple dimensions. Our algorithm consistently performs well in all situations.

Table 5.1: Comparison of the three classifiers.

Classifiers	**Nonlinear separable**	**Multivariate noisy boundary**
Simple Naïve Bayes	sensitive	robust
Decision Tree	robust	sensitive
Our Algorithm	robust	robust

5.3 An Hierarchical Approximation Algorithm

At the end of the previous section, we demonstrated the advantage of using our optimization criteria for classification: robust against nonlinear and multivariate noisy boundaries. To deal with the challenge of object diversity, the learning algorithm must scale with large number of classes. Consistent with our general approach of using hierarchical categorization, in this section, we develop an incremental learning algorithm with hierarchical approximation.

5.3.1 Learning the Hierarchies

As shown in equation (5.7), the parameter space Θ corresponds to the sample partition space. Exhaustively enumerating all possible sample partitions is NP-hard. The number of possible partitioning has the form of the *Sterling number of the second kind* (Sharp, 1968), which grows exponentially with the number of samples.

In order to find the optimal partition, not only tractably but also incrementally, we apply a divide-and-conquer strategy to hierarchically partition the samples. At each level of the hierarchy, not every possible partition is enumerated but only the mostly likely ones. An incremental approximation algorithm is possible by resorting to restricted local greedy search with a bounded computational cost. The structure of the hierarchical approximation algorithm is shown in Figure 5.5.

Like all greedy algorithms, it assumes a certain degree of continuity in the model parameter (Θ) space regarding to the objective function, which is the conditional likelihood function in equation (5.11). For example, if the current sample partition is optimal with regard to the training samples received so far, after receiving the next training instance, the new optimal structure should be within a vicinity of the current sample partition. Good partitions can be efficiently explored by using a greedy local search that takes advantage of the hierarchical structure, which is similar to the algorithm in shown Figure 3.3. Like all greedy algorithms, it may result in local optimal (suboptimal) solutions. Therefore, it is important to empirically evaluate the algorithm. Next, we provide a particular implementation of the algorithm and empirically evaluate the algorithm's performance with systematically generated synthetic data.

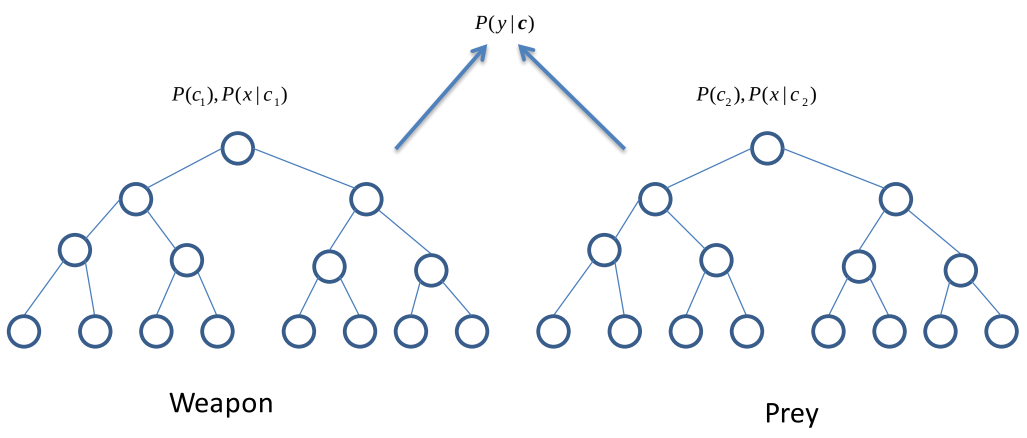
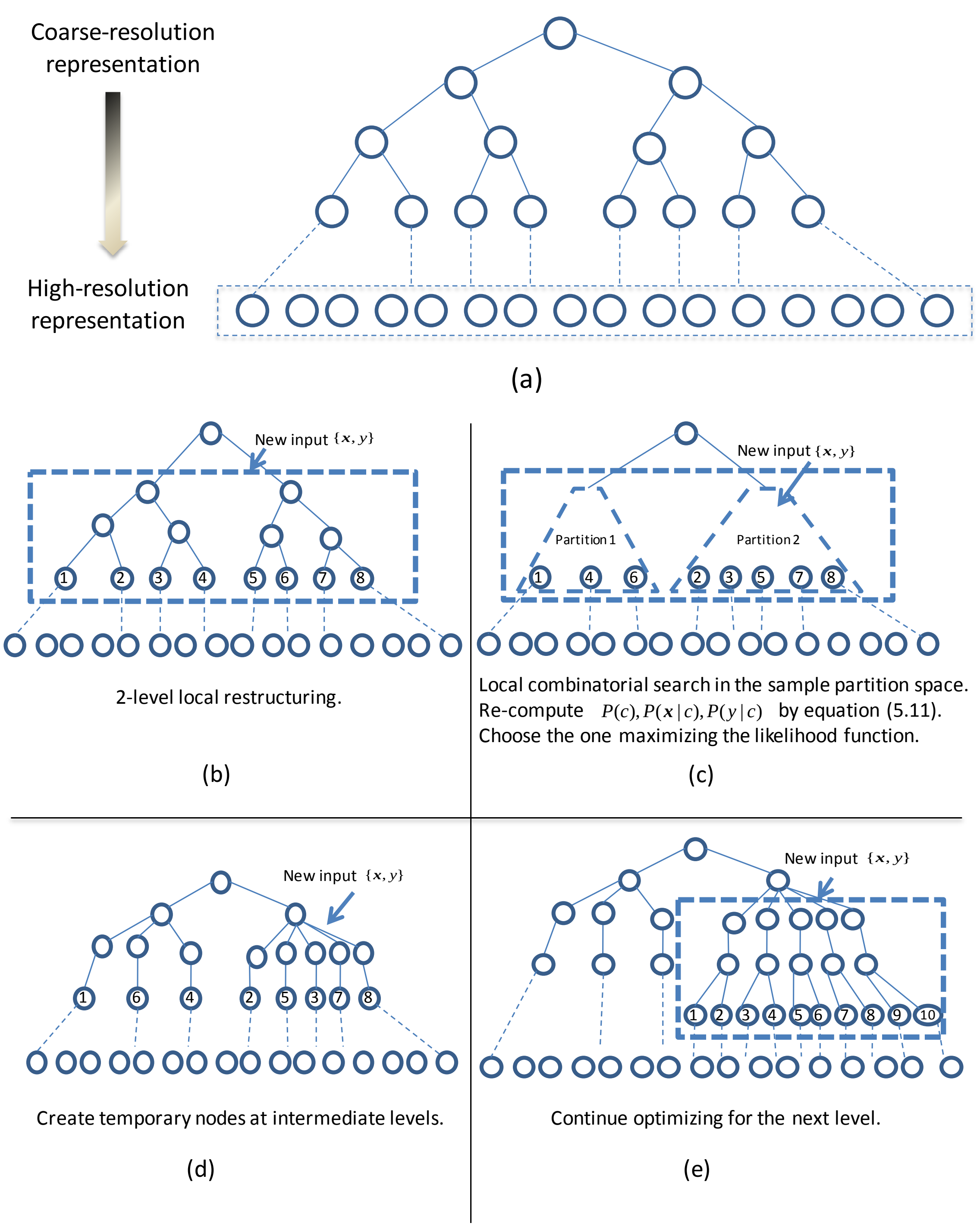


Figure 5.5: Hierarchical representation of the value function.

The model is expected to have the underlying structure as shown in Figure 5.1. The output is conditioned on combinations of class pairs.

Figure 5.6 shows how the incremental algorithm updates the structure with a new training instance, where the function involves only a single category hierarchy. For multiple hierarchies, the algorithm descends the hierarchies simultaneously and the output distributions are associated with combinations of nodes in the hierarchy. The same general procedure can be used for more than two hierarchies.



Coarse-resolution representation
High-resolution representation
(a)
New input $\{x, y\}$
1 2 3 4 5 6 7 8
2-level local restructuring.
(b)
New input $\{x, y\}$
Partition 1
Partition 2
1 4 6 2 3 5 7 8
Local combinatorial search in the sample partition space.
Re-compute $P(c), P(x \mid c), P(y \mid c)$ by equation (5.11).
Choose the one maximizing the likelihood function.
(c)
New input $\{x, y\}$
1 6 4 2 5 3 7 8
Create temporary nodes at intermediate levels.
(d)
New input $\{x, y\}$
1 2 3 4 5 6 7 8 9 10
Continue optimizing for the next level.
(e)

Figure 5.6: Demonstration of the incremental learning algorithm.

(a) shows the current functional category hierarchy. (b)-(e) illustrate the top-down incremental restructuring procedure. The situation is for one hierarchy. Generalizing to multiple hierarchies is straightforward. Dashed boxes represent the scope of local search for the optimal partition.

In Figure 5.6a, the current optimal partition is represented as a hierarchy, which has coarser (more aggregated) resolutions at the top and finer resolutions at the bottom. The algorithm can be configured to use more than two partitions at each level, although in Figure 5.6 it always uses two partitions. It can also be configured to use different number of levels for its local searching scope as represented by the dashed boxes. In Figure 5.6c, in order to find the optimal partitions, the algorithm must re-estimate the PDFs of the three component distributions represented by equations (5.1) to (5.3). The re-estimation is straightforward for simple forms such as independent feature multivariate Gaussian distribution and the Naïve Bayes model. The algorithm descends the hierarchy along the path that incorporates the new training instance. Since the computations are restricted locally and therefore bounded by some constant, the complexity of integrating a new instance is logarithm with regard to the total number of training instances. To control the optimality/computation tradeoff, allowing more levels of the hierarchy to be restructured in one step can increase optimality with more computational cost. Using heuristics in local search can potentially improve the tradeoff ratio (less computational cost without much sacrificing of optimality).

In our implementation, we choose to use simple distribution forms for the three component distributions. More specifically, we use categorical distributions for the class prior distribution represented by equation (5.1), independent variable distribution (as in Naïve Bayes models) for perceptual features represented by equation (5.2), and Gaussian distribution for the output variable represented by equation (5.3). Given these distribution forms, the compact form of the likelihood function as shown in equation (5.11) can be directly evaluated and the update as required in Figure 5.6c can be efficiently performed. Other forms of distribution are possible as long as the above computations can be done efficiently. Exploring those options is beyond the scope of this thesis.

5.3.2 Prediction

As described in equations (5.4) to (5.6), each input object is first mapped to the most likely category in the corresponding hierarchy. Then the prediction is based on the aggregated information associated with the combination of categories. As the algorithm dynamically grows the hierarchies with new samples being incorporated, the predicted output value will be based on the available aggregated information at the most specific level in the hierarchy.

5.4 Empirical Evaluation

To evaluate our algorithm, we need a data set from object-based domains. The input to the algorithm is a state representation consists of the feature vectors of multiple objects. The output of the algorithm is a number reflecting the utility value of the state, or a symbolic label reflecting the class of the state. In addition, the data must contain diverse functional categories. We were unable to find an existing machine learning data set that matches all these criteria. Readily available data sets are either not about object-based domains, or do not consist of diverse enough functional categories. There are ontology systems (Lenat & Guha, 1990; Antoniou & van Harmelen, 2003) that do include diverse natural object categories. However, they do not include data for specific task contexts with the utility value of a state determined by multiple interacting objects.

5.4.1 Synthetic Data

We evaluate the algorithm under varying degrees of object diversity and regularity using systematically generated synthetic data. We decompose the distribution space into two components: the functional space and the perceptual feature space. In the functional space, functional diversity is about how many distinctive functional categories there are, and the regularity is about the existence of hierarchical distribution patterns. In the perceptual feature space, perceptual diversity is about how many perceptual features are available, and the regularity is about the existence of discriminative perceptual features. The parameters are summarized in Table 5.2.

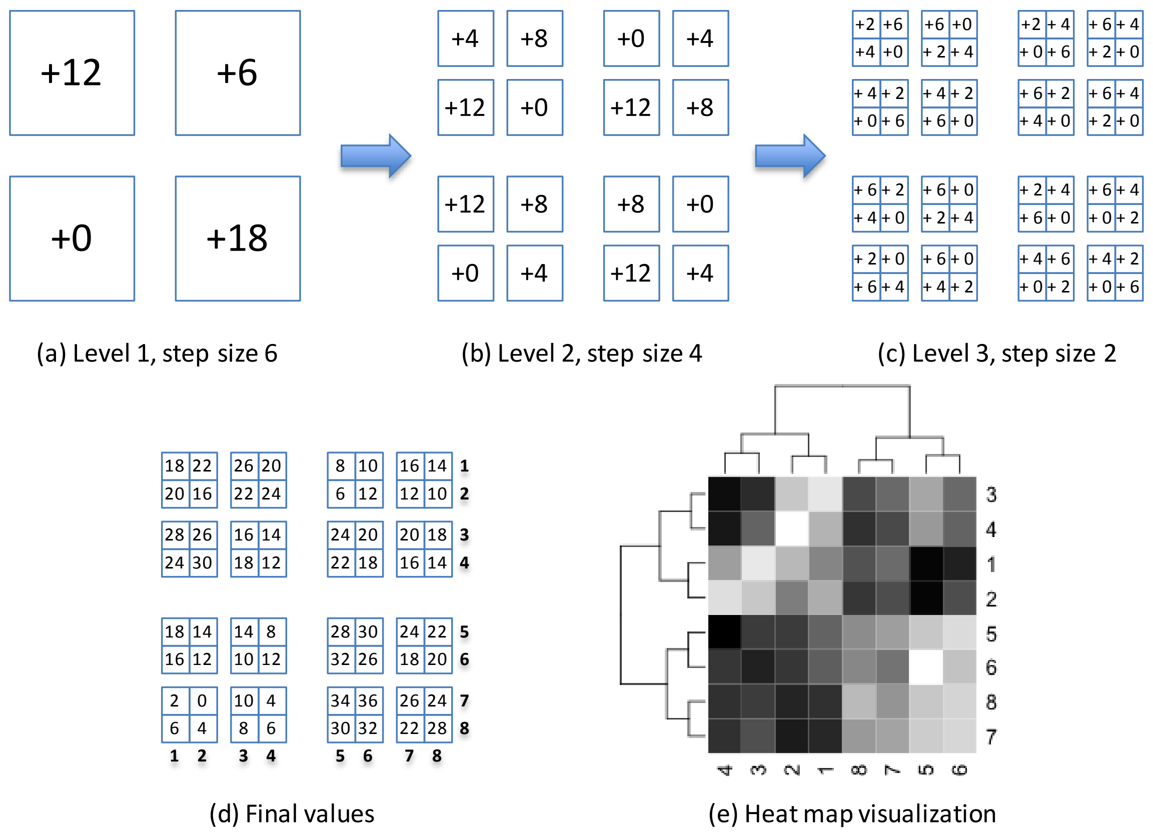
Table 5.2: Parameters for data distribution.

	Functional space	**Perceptual feature space**
Structural diversity	Total number of categories	Total number of features
Regularity	Has hierarchical pattern	Has discriminative features
Random noise	Gaussian noise in output value	Error rate in each feature

5.4.1.1 Functional space

First of all, we use a *hierarchical random walk model* to generate distribution patterns in the functional space, which can be configured to generate arbitrary patterns that are not necessarily hierarchical.

Figure 5.7 shows the generating process for a value function involving two objects, each having 8 distinctive functional categories in a three-level hierarchy. At each level of the hierarchy, a random perturbation with certain magnitude (step size) is added to the output value for each grid representing a distinctive combination of categories. In Figure 5.7a, the step size at the first level is 6, which makes the difference among the four grids to be apart by 6. The assignment is randomly picked for each grid. In Figure 5.7b, further perturbations are introduced at the second level with step size 4 within each of the four grids of level one. Then the third level perturbation is introduced as shown in Figure 5.7c. The final output value of the function is shown in Figure 5.7d, and it is visualized using


(a) Level 1, step size 6 (b) Level 2, step size 4 (c) Level 3, step size 2

(d) Final values (e) Heat map visualization

Figure 5.7: An example of synthetic data generation and visualization.
The function consists of two objects. Each object has a 3-level category hierarchy.

the *heatmap* function in R (R Development Core Team, 2010) in Figure 5.7e where darker color indicates a smaller value. The column and row numbers in Figure 5.7d and 5.7e are the identifiers for the object instances. The *heatmap* function uses the default hierarchical cluster algorithm, which is the hierarchical agglomerative clustering with complete-link method, to perform clustering independently for the columns and rows. It reorders the instances accordingly and normalizes the color gradients for each row. For example, row 7 column 6 has the highest value 36, and row 6 column 5 has a lower value 32. However, the latter has a brighter value because the grey scale gradients are normalized row-wise.

The process in Figure 5.7 generates the hierarchical structure for a particular set of parameters, and further random Gaussian noise can be added to each grid to generate instances of samples. In this simple example, due to a sequence of decreasing step sizes, an apparent hierarchical pattern in the output values can be observed in Figure 5.7e.

5.4.1.2 Perceptual feature space

Perceptual diversity is controlled by the total number of binary features for each object. For example, 3 binary features can produce 8 perceptually distinctive instances and 10 binary features for 1,024 instances. Regarding the regularity, i.e., the relationship between perceptual features and functional categories, our basic assumption is that there exist good discriminative perceptual features to separate the underlying functional categories at each level of the functional hierarchy. For example, the two distinctive values of one binary feature can discriminate two categories, a combination of several binary features can discriminate multiple categories. If otherwise, no good discriminative features exist for some functionally distinctive categories, that part of variance will become intrinsic noise, and thus be unlearnable by any methods. Given the above “good features exist” assumption, the learning problem is to “select” the good features from the pool of all available features.

Random noise can also be introduced to individual perceptual features. In the previous demonstration task as shown in Figure 5.2 to Figure 5.4, we already demonstrated how our model handles noise in the inputs.

5.4.2 Baselines

We include two alternative algorithms to compare with our algorithm. One is a neural network; the other is a nearest neighbor algorithm. Both of them are general function approximators with no assumptions about the functional forms of the underlying data, and they can be trained incrementally. We also include two baselines based on the same hierarchical categorization algorithm, except that their hierarchies are obtained in ways different from the utility value based joint optimization criterion.

2-layered Neural Network (a multilayer perceptron, MLP, with one hidden layer) baseline is trained with the standard back propagation algorithm (Rumelhart, 1986). The input units map to the input features of the objects, and there is a single output unit for the utility value. The network is fully connected. The number of hidden units and learning rate are chosen based on testing of pilot data.

Nearest Neighbor baseline stores all training instances (learning is trivial). To predict the output, it picks the instance that best matches the input perceptual features. In the case of binary features, the matching score is the cardinality of matched features. If there is a tie, the prediction is based on the average of all the best matches.

Unsupervised Hierarchies baseline uses hierarchical categories based on perceptual features and the unsupervised criterion as described in equation (3.1) in Chapter 3. It uses the same procedures to update and predict utility values, but without using the joint optimization criterion as described in equation (5.11). Since the distributions of synthetic data in the perceptual feature space do not have the same hierarchical patterns as in the functional space, the unsupervised hierarchical clustering algorithm, which solely relies on perceptual features, cannot construct the correct functional categorizations and is expected to have poor performance.

Optimal Hierarchies baseline uses the true functional hierarchies that are used to generate the testing data. It uses the same update and prediction procedures, but with fixed optimal hierarchies. Thus, this baseline reflects the optimal learning that can possibly be achieved.

5.4.3 Results and Analysis

We compare our algorithm, piecewise function approximation with joint optimization hierarchies, with the four baselines under different levels of object diversity and different degrees of regularity. The range of data is generated by the model in Figure 5.7, using different configurations of the parameters as shown in Table 5.2. The performance is evaluated by comparing the learning curves of mean squared error (MSE) in the prediction, up to 300 training samples. MSEs are evaluated by using 1000 random independent samples at fixed intervals during training. The final result is the average of 100 independent runs.

To begin with a simple case, we first evaluate the performance for target functions that have low functional diversity and high regularity. We choose the step sizes [6, 4, 2], which generates functional categories with a 3 level hierarchical structure as shown in Figure 5.7. Visualizations of three instantiated functions with the step sizes [6, 4, 2] are shown in Figure 5.8a. The evaluation is based on the average of learning curves for 100 such functions, one for each independent trial. Note that the heatmap function does not necessarily recover the original functional hierarchies used to generate the data. For example, in the leftmost of Figure 5.8a, the true categorization at the first level for the rows should be {1, 2, 3, 4} and {5, 6, 7, 8}, while the heatmap function gives {1, 2, 5, 6, 7, 8} and {3, 4}. This reflects the complexity in the distribution generated by multiple steps of random perturbations.

Figure 5.8b to Figure 5.8d compare the performance under different perceptual diversity and output noise. Figure 5.8b uses 3 binary perceptual features, which is the minimum required to discriminate the 8 functional categories for each object. Nearest neighbor, unsupervised hierarchies and joint optimization hierarchies all achieve performance close to the optimal baseline. MLP learns much slower. In Figure 5.8c, the perceptual diversity is increased by introducing 7 random perceptual features to each object, therefore there are 1,024 perceptually distinctive instances and 8 functional distinctive categories for each object. In Figure 5.8d, functional diversity is further increased by introducing random Gaussian noise with standard deviation 3 to the output values of generated instances.

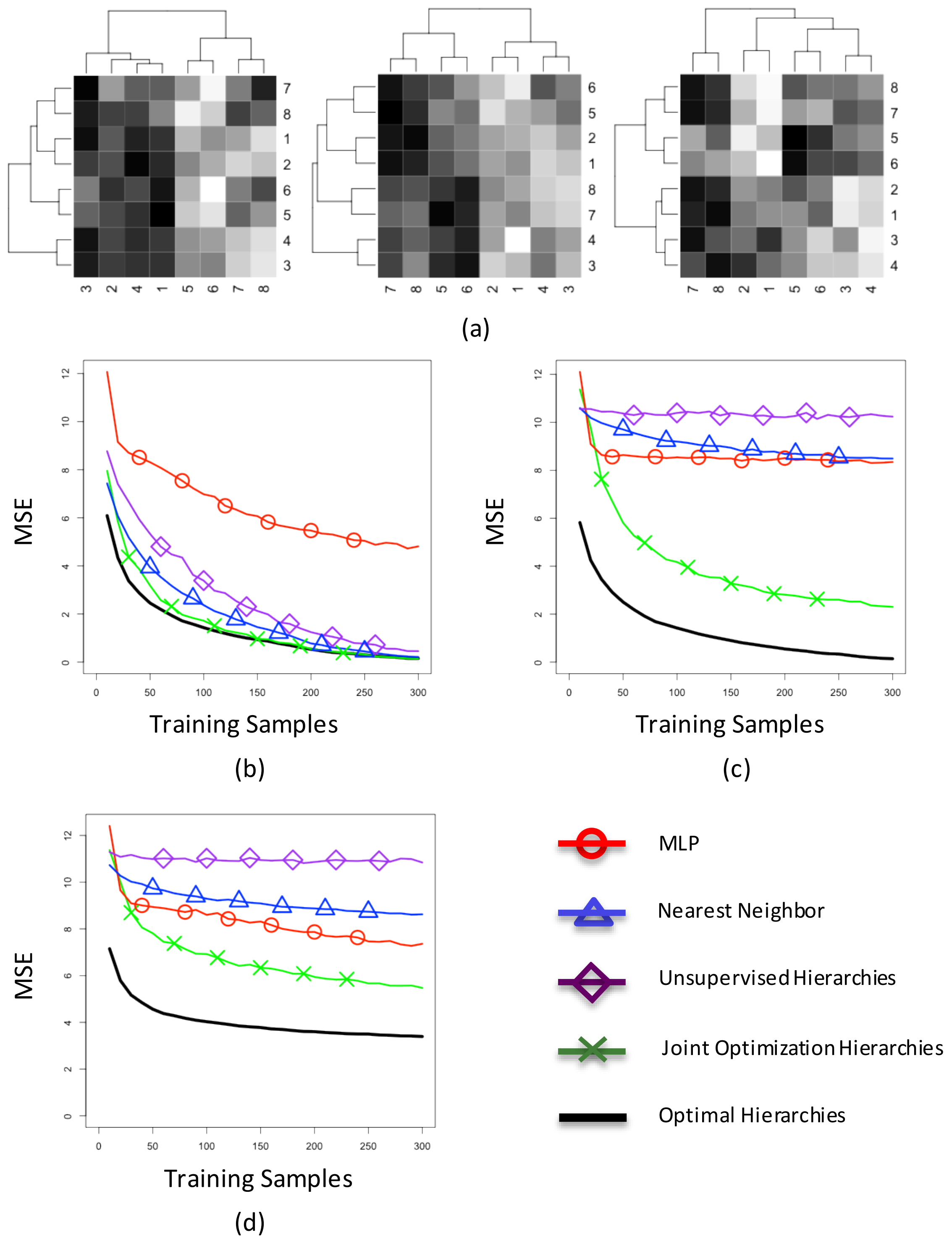


Figure 5.8: Comparison of learning performance for hierarchies with 3 levels.

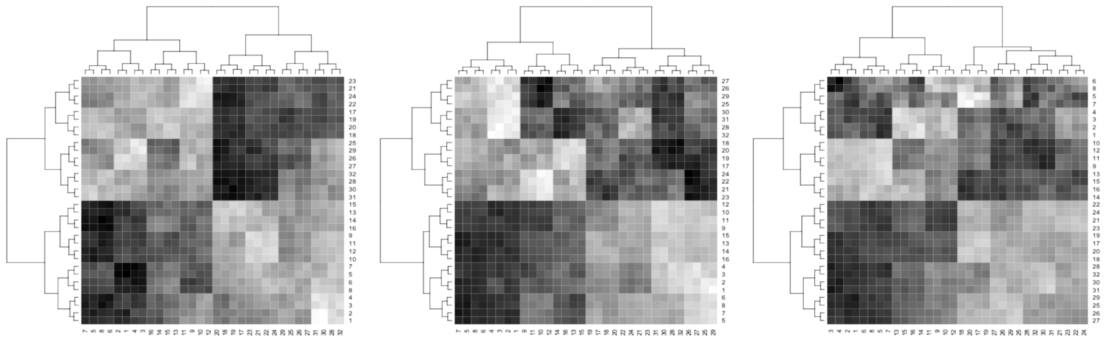
The conditions are: step sizes [6, 4, 2]. (a) visualization of distribution pattern in the functional state space. (b) 3 binary perceptual features (no irrelevant features); no noise in output value. (c) 10 binary perceptual features (7 irrelevant features); no noise in output value. (d) 10 binary perceptual features (7 irrelevant features); Gaussian noise with standard deviation 3 in output value.

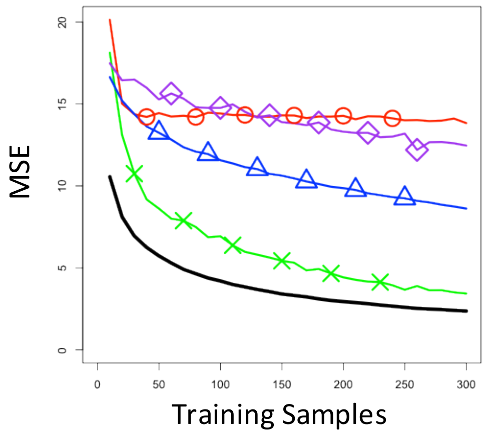
Figure 5.8 illustrates that for a given diversity and regularity in the underlying functional category hierarchies, how perceptual diversity and random noise in output affects the performance. The MLP baseline generally learns slower because it is designed to approximate smooth functions and is more globally constrained. In our data set, the target function is rather unsmooth especially at the lower levels, which causes significant interference among the learning units in MLP. In such situations, a winner-take-all learner with local approximation will generally perform better. The nearest neighbor algorithm baseline is a winner-take-all algorithm; however, it is sensitive to irrelevant features. On the other hand, the neural network can adaptively adjust the weights of features based on their relevance to the output value. That is why in Figure 5.8c, the neural network learns faster than nearest neighbor initially, but stops improving after certain point. It can be seen that when perceptual diversity increases as in Figure 5.8c, the nearest neighbor algorithm, along with the unsupervised hierarchies baseline, are the most affected due to sensitivity to irrelevant features. The performances are not differentiated very much by adding random noise in the output. In Figure 5.8d, we can see that all algorithms have a certain level of noise tolerance, because they are not affected as much as the optimal learning curve.

In the following tests, we increase the degree of functional diversity by adding more perturbation steps, and vary the functional regularity by using different step sizes. For each regularity pattern, we perform the same set of comparison as shown in Figure 5.8.

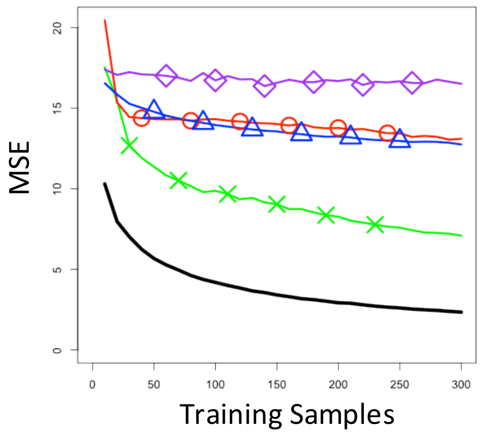
5.4.3.1 Increase functional diversity

Figure 5.9a shows the visualization for three instantiations of functions that have a 5-level hierarchical structure with step sizes being [10, 6, 4, 2, 1]. Therefore, there are 32 distinctive functional categories for each object. Figure 5.9b to Figure 5.9d compare the performance under different levels of perceptual diversity and noise. Figure 5.9b uses 5 perceptual features, which is the minimum required to discriminate the 5-level hierarchy (32 distinctive functional categories) for each object. In Figure 5.9c, the total number of perceptual features for each category is increased to 10 (5 irrelevant features). In Figure 5.9d, Gaussian noise with standard deviation 3 is added the output values. The trends are the same as in Figure 5.8.

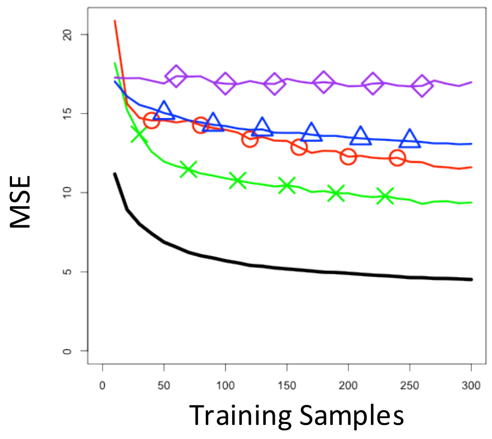

(a)



(b)



(c)



(d)

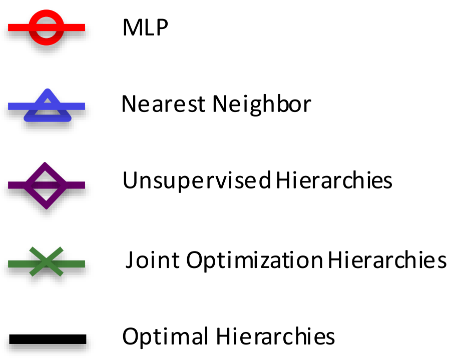


Figure 5.9: Comparison of learning performance for hierarchies with 5 levels.

The conditions are: step sizes [10, 6, 4, 2, 1]. (a) visualization of distribution pattern in the functional state space. (b) 5 binary perceptual features (no irrelevant features); no noise in output value. (c) 10 binary perceptual features (5 irrelevant features); no noise in output value. (d) 10 binary perceptual features (5 irrelevant features); Gaussian noise with standard deviation 3 in output value.

5.4.3.2 Reduce functional regularity

Both of the previous data sets have high regularity in the functional space, although with different degrees of diversity. Our algorithm (joint optimization hierarchies) consistently outperforms the other baselines because it can take advantage of such hierarchical distribution assumptions. In order to better understand the potential capability of our algorithm under more adversarial conditions, we perform the same set of testing using different patterns of step sizes that generate functions with less degrees of regularity.

Figure 5.10a shows the visualization for functions with step sizes being [3, 6, 4, 2, 1], where the second step size is the largest. As a consequence, it exhibits a pattern with smaller regions of consistent value blocks compared to Figure 5.9a.

Same as the previous settings, Figure 5.10b to Figure 5.10d compare the performance under different levels of perceptual diversity and noise. In Figure 5.10b, the relative performance is the same as in previous experiments. However, in Figure 5.10c, when irrelevant features are added to the input, our algorithm only performs on par with the nearest neighbor and MLP baselines. This is because the reduced regularity makes it more difficult to discover the true hierarchies.

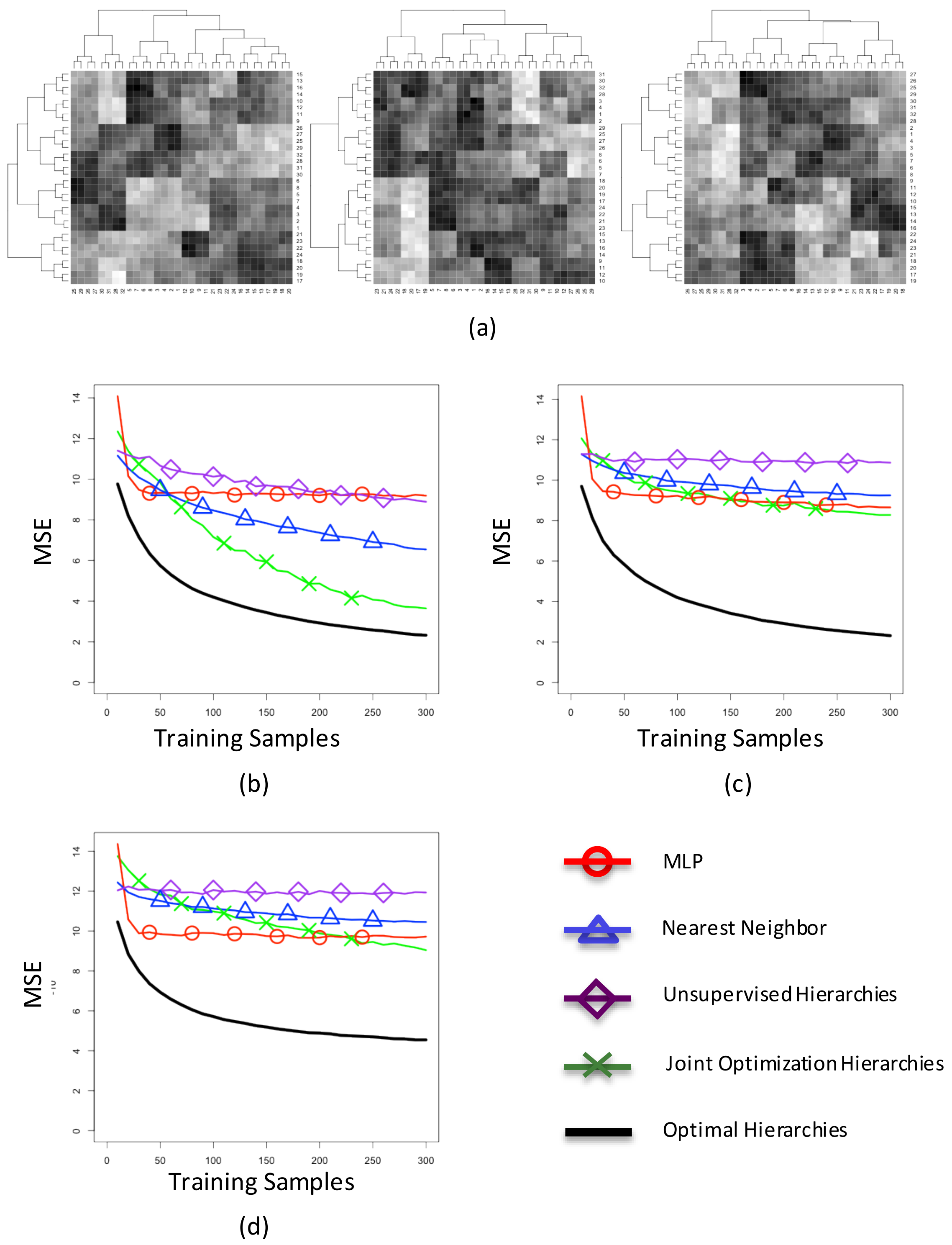


Figure 5.10: Comparison of learning performance with reduced regularity.

The conditions are: step sizes [3, 6, 4, 2, 1]. (a) visualization of distribution pattern in the functional state space. (b) 5 binary perceptual features (no irrelevant features); no noise in output value. (c) 10 binary perceptual features (5 irrelevant features); no noise in output value. (d) 10 binary perceptual features (5 irrelevant features); Gaussian noise with standard deviation 3 in output value.

Figure 5.11a shows the visualization for functions with step sizes being [6, 5, 4, 3, 2], where the consecutive step sizes are only different by 1. The hierarchical distribution pattern is less obvious comparing to previous data sets.

Same as the previous settings, Figure 5.11b to Figure 5.11d compare the performance under different levels of perceptual diversity and noise. In Figure 5.11b, when there is no other types of noise, our algorithm is only marginally better than the MLP and nearest neighbor baselines. When different types of noise are added to the data in Figure 5.11c and Figure 5.11d, performances of the different algorithms are indistinguishable.

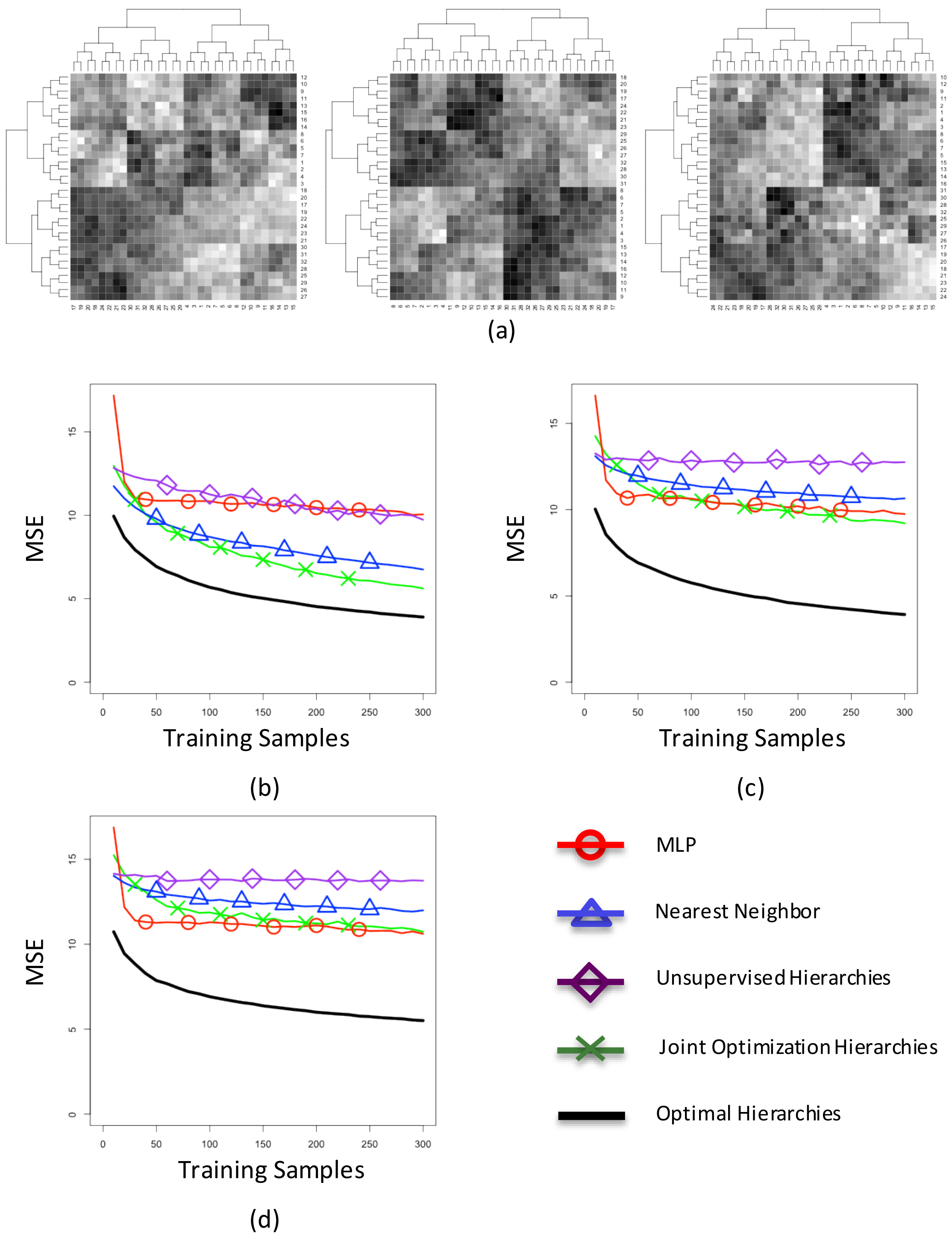


Figure 5.11: Comparison of learning performance with reduced regularity.

The conditions are: step sizes [6, 5, 4, 3, 2]. (a) visualization of distribution pattern in the functional state space. (b) 5 binary perceptual features (no irrelevant features); no noise in output value. (c) 10 binary perceptual features (5 irrelevant features); no noise in output value. (d) 10 binary perceptual features (5 irrelevant features); Gaussian noise with standard deviation 3 in output value.

Figure 5.12a shows the visualization for functions with step sizes being [2, 2, 2, 2, 2]. The hierarchical pattern is reduced to almost randomness.

Same as the previous settings, Figure 5.12b to Figure 5.12d compare the performance under different levels of perceptual diversity and noise. In this situation, the neural network algorithm (MLP baseline) becomes the best option in Figure 5.12c and Figure 5.12d. This is because the neural network does not make any assumption about the distribution of the data, while for our algorithm, making the hierarchical assumption incurs an extra cost when the expected pattern is not presented in the data. However, the relative magnitude of the learnable information in the data is largely reduced, as revealed by the position of the optimal baseline.

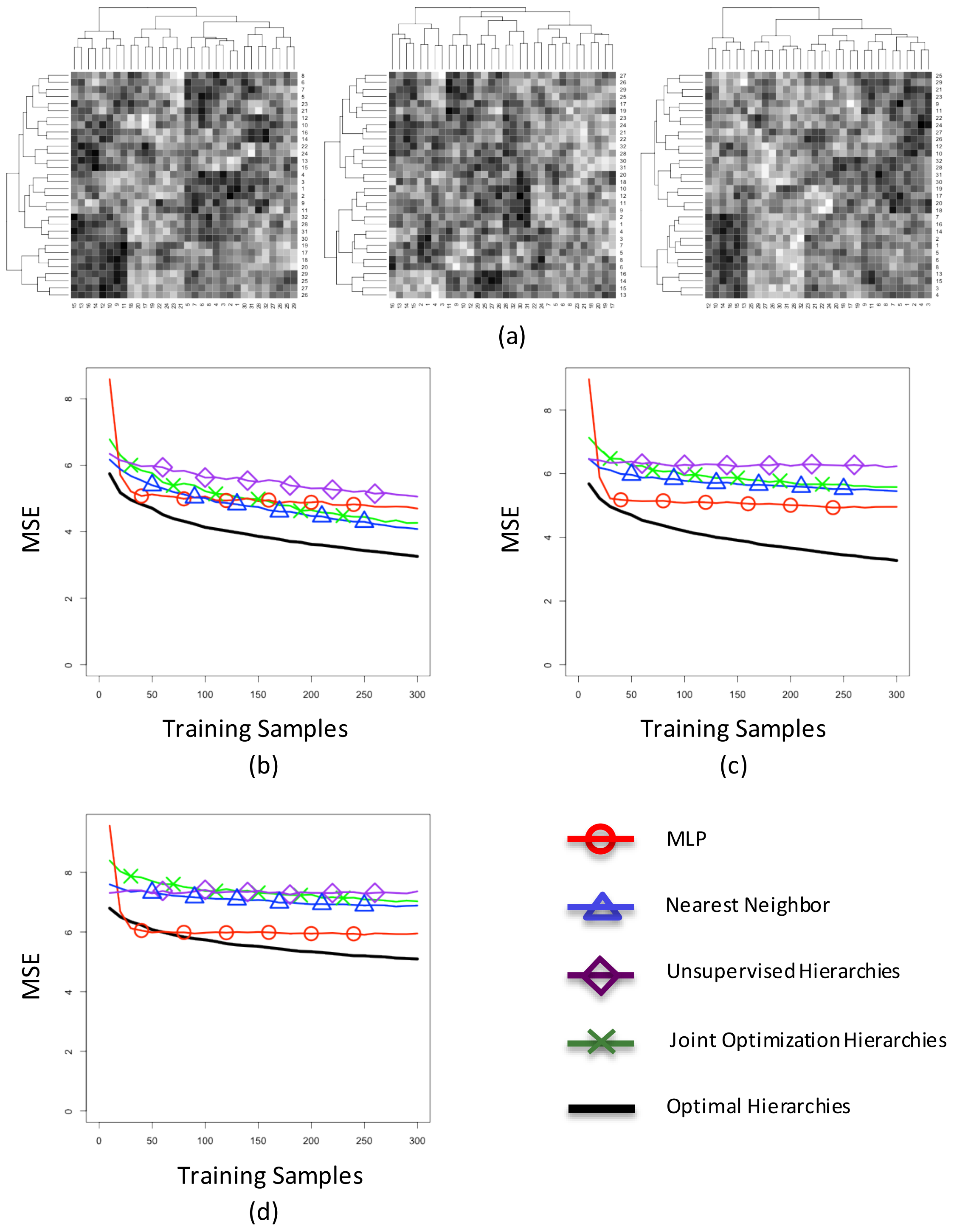


Figure 5.12: Comparison of learning performance with reduced regularity.

The conditions are: step sizes [2, 2, 2, 2, 2]. (a) visualization of distribution pattern in the functional state space. (b) 5 binary perceptual features (no irrelevant features); no noise in output value. (c) 10 binary perceptual features (5 irrelevant features); no noise in output value. (d) 10 binary perceptual features (5 irrelevant features); Gaussian noise with standard deviation 3 in output value.

5.4.3.3 Analysis

Our algorithm is robust against increased functional diversity, but sensitive to reduced functional regularity. It consistently outperforms nearest neighbor and neural network with different degrees of object diversity when abundant regularity is present. When the regularity is low, the performance is indistinguishable from, or slightly worse than the nearest neighbor and the neural network baselines. From another perspective, the difference in performance can be attributed to the lack of regularization in the nearest neighbor and neural network baselines in the general sense (here regularization has the same meaning as constraints). For the nearest neighbor algorithm, the desirable regularization should help it only keep a small set of instances that focus on the relevant features, as opposed to keeping all instances and thus being distracted by irrelevant features. For the neural network algorithm, the desirable regularization should make the connection weights' updates gradually progress towards more specific categories and prevent it from interfering with the general categories once learned. However, nearest neighbor and neural network algorithms do not have a natural way of the proper regularization for hierarchical distribution pattern. For example, the common regularization techniques for neural networks include: limiting the number of hidden units, constraining network connectivity, weight decay, early stopping, input/output transformation and weight sharing. These techniques are aimed at different situations, but none of them capture the hierarchical distribution pattern. On the other hand, the assumption about hierarchical distribution pattern is built-in with our algorithm, which will yield superior performance when such regularity is presented in the data.

5.5 Compare with Other Approaches

The latent hierarchical clustering model we have presented in this chapter is specifically designed for object-based environments. Our algorithm achieves nonlinear function approximation via piecewise approximation. It can adaptively choose the piecewise segment boundaries by learning hierarchically organized functional categories of the component objects. We briefly review other commonly used function approximation approaches. The purpose is to understand the key differences in the underlying assumptions of these alternatives.

Linear models are among the most commonly used approximators. Linear regression (Draper & Smith, 1998) models are used when the output value is numeric. For classification, there are linear discriminant analysis (LDA) (McLachlan, 2004), and the logistic regression model (Agresti, 2007). The major limitation of linear models is that the target function is restricted to be in linear form with a set of variables. These variables can be either a direct measure of raw features, or a transformed measurement via a *link function* as in the generalized linear model (GLM) framework (Nelder & Wedderburn, 1972). The advantages of linear models are simplicity, interpretability, and efficient computability. However, the linear assumption limits the flexibility of the model.

Support vector machines (SVM) are classifiers based on the maximum margin principle (Cortest & Vapnik, 1995). SVMs can be adapted to different tasks and domains by the appropriate choice of kernel functions (Schölkopf & Smola, 2001). For example, SVM using graph kernels has been successfully used to predict the function of proteins (Borgwardt *et al*., 2005), which are represented by graph structures. However, it is not clear how the combinatorial structure and hierarchical distribution pattern can be captured by kernel functions.

Radial basis function (RBF) networks use linear combinations of radial basis functions to achieve nonlinear function interpolation. They are conceptually similar to k-nearest neighbor algorithms. The difference is the former keep a fixed number of basis functions and adaptively learn a set of weights, one for each basis function using least squares criterion. K-nearest neighbor algorithms can dynamically expand their instance base, and do not need to learn the weights (they will weigh equally the top k matches). RBF networks and k-nearest neighbor algorithms have been successfully used in various applications such as handwritten digit recognition (Lee, 1991). The performance of a RBF network relies on the set of manually selected basis functions. Since we do not assume there is sufficient prior knowledge to properly choose the basis functions in complex object-based environments, we use the nearest neighbor algorithm in our evaluation, which can serve as a special case representative of this class of algorithms.

Multilayer perceptron (MLP) are feedforward neural networks with one or more hidden layers trained by the back-propagation algorithm (Rumelhart *et al*., 1986). A MLP can approximate nonlinear functions by using nonlinear sigmoid activation functions (such as logistic function, or hyperbolic tangent function) in the hidden layer. Compared to linear regression models, the extra hidden layers in a neural network perform adaptive feature learning. A MLP with one hidden layer is included as one of the baselines in our evaluation. Using more hidden layers increases the MLP's function approximation capacity, but generally slows down learning because of the increased parameter space. Although there are methods to improve the learning speed for deep neural networks (Hinton *et al.*, 2006), we do not expect adding more layers can improve the MLP's performance in learning functions with hierarchical distribution patterns.

Convolutional neural networks are multilayer neural networks with constrained connections (LeCun *et al.*, 1990). Their connection structures are designed to model images of objects by composing local invariant features at different scales (Lee *et al.*, 2009). From the perspective of function approximation, convolutional networks rely on similar principles such as latent representation and hierarchical structure to achieve efficient nonlinear function approximation. The key difference is that the hierarchy in convolutional neural networks are based on spatial decompositions (compositional hierarchy), while in our model, the hierarchy is based on sample partitions (discriminative hierarchy). For example, in a convolutional network, the hidden layers contain representations of local feature, each being a component of the object. The hidden layers in our model form categories of objects, with each category representing a subset of the sample population. It is possible to integrate a convolutional network as the perceptual processing layer as shown in the bottom of Figure 3.1, which can extract higher level features from pixels.

Conditional random fields (CRFs) are discriminative probabilistic graphical models (Lafferty, 2001). In addition to the ability of modeling dependency structures, CRF models can have latent variables (Gunawardana *et al.*, 2005), they can support kernels (Lafferty, 2004), and can be trained incrementally with stochastic gradient descent methods (Vishwanathan *et al.*, 2006). These properties make CRFs powerful machine learning tools for models with highly interdependent variables, such as natural language processing problems (Sha & Pereira, 2003; Settles, 2004) and biological sequence modeling (DeCaprio *et al.*, 2007). However, it is not clear how CRF models can capture the distribution assumption of hierarchical categorization, which is essential for making efficient generalization in object based environment according to our general assumption.

5.6 Summary

One major limitation of the algorithm introduced in Chapter 3 is that hierarchical category learning and value function approximation are performed in separate learning components. As a consequence, the formation of categories does not depend on feedback from the value function learning process. This limits the performance of the system in domains where prior knowledge about the functional feature set is not accurate.

The algorithm introduced in this chapter aims to address this limitation. In the new algorithm, hierarchical categorization and value function approximation are integrated into a single optimization step: internal categories are formed synchronously to optimize the performance of value function approximation. The algorithm effectively selects the functional features in the learning process.

The important assumption designed into our learning algorithms is the existence of hierarchical structure in the distribution of the objects' functional space. We created a parameterized data generation model that can produce data sets with different patterns of hierarchical distributions. The evaluation is based on systematically generated data set, ranging from simple data set with obvious hierarchy structures to more complex data set without observable patterns. Our algorithm consistently outperforms the baseline algorithms, and the difference margin is correlated with the degree of hierarchical pattern presented in the data set.

We also briefly review other commonly used function approximation approaches. The advantage of our algorithm, comparing to the other alternatives, is that the assumption about the hierarchical category distribution pattern is built into the learning algorithm.

Chapter 6

Conclusion

Our research presents a synthesis of hierarchical category learning and value function approximation, which are traditionally formulated as separate machine learning problems. We first identify and analyze the challenges and assumptions for the problem of long-lived learning agent in complex object-based environments. We provide novel implementations exploiting the distribution assumption of hierarchical categorization. Our work not only provides efficient value function approximation algorithms, but also offers computational insights to human category learning.

6.1 Discussion

The key to the success of our approach, compared to alternative value function approximation approaches, is using hierarchical categories as internal representations. The hierarchical structure not only supports flexible generalization to achieve sample efficiency, but also supports incremental learning algorithms to achieve computational efficiency.

The existence of hierarchical distribution patterns in the functional space of objects, as illustrated in the evaluations of Chapter 3 and Chapter 5, is a critical assumption for our approach. Although it is not the intention of the thesis to prove the general existence of such patterns in natural environments, this is a reasonable assumption based on converging observations. Using hierarchies to organize natural objects has been a practice in AI systems (Quillian, 1967; Lenat & Guha, 1990), as well as a recurring theme in human cognitive models (Mervis & Rosch, 1981; Kemp & Tenebaum, 2006). The results in Chapter 4 also support such assumption.

Hierarchical structure has already been exploited in many machine learning algorithms. Hierarchical clustering algorithms are often used as data analysis tools for visualizing the general pattern of data distribution in high dimensional feature space. However, these are unsupervised learning algorithms, and do not directly support learning a value function. There are also supervised learning algorithms that leverage on hierarchical structures, such as kd-tree based value function approximation (Vollbrecht, 2000), adaptive tile coding (Whiteson, 2007) and the general decision tree algorithm (Quinlan, 1986). However, these algorithms hierarchically partition the entire state feature vector, which does not capture the compositional structure of object-based environments. As a consequence, it results in less efficient learning as shown in the 3rd comparison (Figure 3.11) in Chapter 3.

Our approach leverages on having prior knowledge built into the learning algorithm. According to the “no free lunch” theorems (Wolpert & Macready, 1997), there is no general-purpose universal optimization algorithm. One algorithm outperforms another because it is specialized to the specific problem under consideration, and there is generally a performance advantage in incorporating prior knowledge of the problem into the algorithm. In practice, what are the specific forms of prior knowledge, and how they can be injected into the algorithm, depend on technical details of the implementation. In our approach, we present two forms of prior knowledge. The hierarchical category structure can be considered as a very general form of prior knowledge, which is used across different tasks. Furthermore, in the first algorithm (Chapter 3), the hierarchical structure ca be controlled by selected functional feature sets, which are more specific prior knowledge and can be tuned for different tasks.

The general advantage of using hierarchical representations for value function approximation is that the system can achieve both fast learning and accurate learning, which is otherwise a dilemma (section 3.3.2.3). Conceptually, the system first learns the value function at a coarser level, then progress down towards a higher resolution at more specific levels as more training samples are received. Intuitively, it appears to require local approximators be recursively processed at different resolution levels both for learning and prediction. This is the case for the second algorithm presented in Chapter 5.

In the first algorithm presented in Chapter 3, however, learning and predicting the value function is performed in a linear form of the coarse coding basis variables, where the coefficients are simultaneously updated. Linear models generally assume, or prefer, that the input variables are independent. A unique feature in our system is that the input variables to the linear approximator are coarse coding variables with a systematic dependent structure (Figure 3.4). The original motivation for this implementation is because a linear coarse coding learning mechanism has already been implemented in Soar-RL (Nason & Laird, 2005), and we only need to test how well does it perform when coarse coding variables have such dependency structures. The result is somewhat surprising: the same general linear mechanism achieves the desired general-to-specific learning utilizing the hierarchical structures. We have also use used the same mechanisms to faithfully replicate rat behavioral data in a maze environment without object-based representations (Wang & Laird, 2008).

6.2 Contributions

This section summarizes the major contributions.

Efficient value function approximation algorithms. The unique feature of our approach is the integration of hierarchical category learning into the value function approximation algorithm. Hierarchical category learning forms symbolic categories, which are used as representational basis to approximate the value function. We developed two specific algorithms based on this general design. The first algorithm (Chapter 3), as the initial step, combines two separate learning algorithms: a hierarchical clustering algorithm adapted from COBWEB (Fisher, 1987), and a sparse coarse coding based function approximator implemented in Soar-RL (Nason & Laird, 2005). The second algorithm (Chapter 5) coherently integrates hierarchical categorization and value function approximation into a single joint optimization problem. Both algorithms achieve efficient learning of value functions, and scale well with object diversity.

Computational model of human category learning. The observation of naturally occurring hierarchies in human category learning (Mervis and Rosch, 1981) is one major source of inspiration for this research. In return, our research provides computational insights to human category learning (Chapter 4).

Extend the capabilities of a general cognitive architecture. Soar is a general cognitive architecture, which has been serving as a framework for building robust, long-lived, interactive online learning agents (Laird, 2008). The learning system presented in Chapter 3 extends the category learning capability of Soar. Together with other recent extensions (Nason & Laird, 2005; Nuxoll & Laird, 2007; Lathrop & Laird, 2009; Wintermute, 2010), this work makes Soar a more powerful framework for creating intelligent agents with improved functionality.

6.3 Future Directions

We identify the following areas of work that can directly follow from this thesis.

Combine prior knowledge with utility driven optimization. In the first algorithm (Chapter 3), the category hierarchy is controlled by prior knowledge in the form of preselected functional feature sets, and does not depend on the performance of value function learning. In the second algorithm (Chapter 5), which has greater flexibility, the hierarchical structures are jointly optimized with the utility value function. Nevertheless, there is still the advantage to incorporate prior knowledge especially during the initial learning stage when the training samples are insufficient for making useful inferences. It is technically straightforward to numerically combine the information from preselected functional feature sets with the optimization criteria in the second algorithm. However, how to appropriately balance the two factors is an empirical question, which must be explored with specific domains and tasks.

Qualitatively model human category learning using the second algorithm. In Chapter 4, we use the first algorithm (Chapter 3) to generate category learning behaviors that qualitatively match with human. The successful matching is not a coincidence of the specific implementation, but a general property of integrating hierarchical category learning with utility based value function approximation. The second algorithm in Chapter 5 follows the same general design. Therefore, we expect the same kind of qualitative category learning behaviors. In order to model the human category learning behaviors, salient symbolic categories must be extracted from the system. We define a category domination model for the first algorithm in section 4.3.2. Similarly, we must define a salient category extraction criterion for the second algorithm.

Integrate the second algorithm into the Soar cognitive architecture. The second algorithm (Chapter 5) is designed to achieve utility based category learning without relying on preselected functional feature sets. The implementation respects the general constraints of a cognitive agent, which are incremental learning and scalability with large knowledge base by using hierarchical structures. In order to achieve the ultimate goal of creating robust long-lived agent, it is desirable to integrate the algorithm into the Soar cognitive architecture, which has already been equipped with many useful capabilities. An initial integration is to directly include the second algorithm as an independent module for deliberately learning and predicting value functions, with an interface similar to other deliberate modules such as the semantic memory module (Derbinsky & Laird, 2010) and the mental imagery module (Lathrop & Laird, 2009). In a more parsimony integration, the system should share this value function approximation module with the architectural RL mechanism. This may lead to modifying the existing Soar-RL implementation and require exploring new use cases with extensive empirical evaluations.

Sequential decision making. Both the original motivation and evaluation tasks are closely related to the reinforcement learning problem, where an agent incrementally adjusts its behavior based on a numeric feedback (reward) from the environment. Among numerous topics in the field of reinforcement learning, our approach is a special case of value function approximation via state aggregation. More specifically, our state aggregation is achieved by hierarchical aggregation of individual objects. In addition to value function approximation, classic reinforcement learning problems involve sequential decision making in stochastic environments which are normally modeled as Markov Decision Processes (MDPs). In such models, the value functions in the context of the MDPs are incrementally estimated simultaneously with the value function approximation, which complicates the convergence of learning. Theoretic analysis about convergence with value function approximation has been a major area in the field of RL (Singh *et al.*, 1995; Maei *et al.*, 2009). In this thesis, we make two simplifying assumptions to eliminate such convergence concerns. First we assume the availability of stationary state values for training, and second, we assume that there is only a one shot decision to be made based on the value function. It is an important future work to empirically characterize the convergence properties of our value function approximation algorithm in sequential decision making tasks.

Integration with spatial abstractions. Since this research focuses on object category learning, another simplifying assumption we have made is about the spatial aspect. Our algorithms form abstract symbolic categories for independent objects, without exploiting abstract spatial relations among objects. One important topic for future work is to integrate our hierarchical object categorization approach with spatial relationship abstraction approaches (Kuipers, 2000; Wintermute, 2010). This integration will enable more powerful generalization in spatial domains. For example, in the hunting task in Chapter 3, we only aggregate states having the same distance-to-prey. If there are more abstract spatial predicates such as “close”, “far”, and “very far” *etc.*, it would be possible to aggregate more states together and achieve higher degree of generalization.

Bibliography

Agresti, A. (2007). Building and applying logistic regression models. *An Introduction to Categorical Data Analysis*. Hoboken, New Jersey: Wiley.

Ambros-Ingerson, J., Granger, R., & Lynch, G. (1990). Simulation of paleocortex performs hierarchical clustering. *Science*, *247*, 1344-1348.

Anderson, J., Bothell, D., Byrne, M., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036-1060.

Antoniou, G., & van Harmelen, F. (2003). Web Ontology Language: OWL. *Handbook on Ontologies in Information Systems*, 67–92.

Ashby, E. G., & Maddox, W. T. (2005). Human category learning. *Annual Review of Psychology*, 56, 149-178.

Barto, A. G. (1995). Adaptive critics and the basal ganglia. *Models of Information Processing in the Basal Ganglia*. MIT Press.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A.J., & Kriegel, H. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21(1), 147-156

Bradtke, S. J., & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1), 33-57.

Brodley, C. E., & Utgoff, P. E. (1995). Multivariate decision trees. Machine Learning, 19(1), 45-77.

Clogg, C. C. (1995). Latent class models. *Handbook of statistical modeling of social and behavioral sciences*, Ch.6, 311-359. New York: Plenum.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273-297

DeCaprio, D., Vinson, J. P., Pearson, M. D., Montgomery, P., Doherty, M., Galagan, J. E. (2007). Conrad: Gene Prediction using Conditional Random Fields. *Genome Research 17*(9), 1389–1396.

Derbinsky, N., Laird, J. E. (2010). *Extending Soar with Dissociated Symbolic Memories.* Symposium on Human Memory for Artificial Agents, AISB (2010).

Diuk, C., Cohen, A., & Littman, M. L. (2008) An object-oriented representation for efficient reinforcement learning. *Proceedings of 25th International Conference on Machine learning*, 240-247.

Draper, N.R., & Smith, H. (1998). *Applied Regression Analysis*. Wiley Series in Probability and Statistics.

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous Localization and Mapping (SLAM): Part I The Essential Algorithms". *Robotics and Automation Magazine*, 13, 99–110

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43, 7-52

Falkenhainer, B., Forbus, K., & Gentner, D. (1989). The structure-mapping engine - algorithm and examples. *Artificial Intelligence*, 41(1), 1-63.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2), 139-172.

Forgy, C. L. (1982). Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1), 17-37.

French, R. M. (1991). Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. *Proceedings of the 13th Annual Cognitive Science Conference*, 173-178.

Frigui, H., & Krishnapuram, R. (1997). Clustering by competitive agglomeration. *Pattern Recognition*, 30(7), 1109-1119.

Fu, W., & Anderson, J. R. (2006). From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology: General, 135*, 184-206.

Gennari, J. H., Langley, P., & Fisher, D. H. (1989). Models of incremental concept formation. *Artificial Intelligence*, 40, 11-61

Granger, R. (2006). Engines of the brain: the computational instruction set of human cognition. *AI Magazine., 27*(2), 15-32.

Gunawardana, A., Mahajan, M., Acero, A., & Platt, J. C. (2005). Hidden conditional random fields for phone classification. *Proceedings of Eurospeech.*

Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*(7), 1527-1554.

Johnson, C. (1987). *Numerical solution of partial differential equations by the finite element method*. Cambridge University Press.

Jolion, J. M., Meer, P., & Bataouche, S. (1991). Robust clustering with applications in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8), 791-802.

Kemp C. & Tenebaum, J. (2006). The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31), 10687-10692.

Kuipers, B. (2000). The Spatial Semantic Hierarchy. *Artificial Intelligence, 119*(1-2), 191-233.

Kuipers, B., Modayil, J., Beeson, P., MacMahon, M., & Savelli, F. (2004). Local metrical and global topological maps in the hybrid Spatial Semantic Hierarchy. *IEEE International Conference on Robotics and Automation (ICRA-04).*

Kullback, S., & Leibler, R.A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22 (1), 79–86

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the 18th International Conf. on Machine Learning*, 282–289.

Lafferty, J., Zhu, X., & Liu, Y. (2004). Kernel conditional random fields: Representation and clique selection. *The 21st International Conference on Machine Learning (ICML).*

Laird, J. E. (2008). Extending the Soar cognitive architecture. *Proceedings of the First Conference on Artificial General Intelligence.*

Langley, P., McKusick, K. B., Allen, J. A., Iba, W. F., & Thompson, K. (1991). A design for the ICARUS architecture. *ACM SIGART Bulletin*, *2*(4), 104-109.

Lathrop, S. D., & Laird, J. E. (2009). Extending cognitive architectures with mental imagery. *Proceedings of the Second Conference on Artificial General Intelligence.*

LeCun, Y., Baser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, R., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems 2*.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the Twenth-Sixth International Conference on Machine Learning (ICML).*

Lee, Y. (1991). Handwritten digit recognition using k nearest-neighbor, radial-basis function, and backpropagation neural networks. *Neural Computation*, 3, 440-449.

Lenat, D. B., & Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* Addison-Wesley.

Love, B. C., Medin, D. L., & Gureckis, T. M. (2004). SUSTAIN: A network model of category learning. *Psychological Review*, *111*(2), 309-332.

Maei, H. R., Szepesvari, C., Batnaghar, S., Precup, D, Silver, D., & Sutton, R. S. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. *Advances in Neural Information Processing Systems* 22, 1204–1212.

McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are Complementary Learning Systems in the Hippocampus and Neocortex. *Psychological Review*, 102(3), 419-457.

McLachlan, G. J. (2004). *Discriminant Analysis and Statistical Pattern Recognition*. Wiley Interscience.

McShea, D. W., & Brandon R. N. (2010). Biology’s First Law: The Tendency for Diversity and Complexity to Increase in Evolutionary Systems. University of Chicago Press.

Medin, D. L., & Schaffer, M. M. (1978). Context theory of classification learning. *Psychological Review*, 85(3), 207-238.

Mervis, C. B., Rosch, E. (1981). Categorization of natural objects. *Annual Review of Psychology*, 32, 87-115

Nason, S., & Laird, J. E. (2005). Soar-RL: integrating reinforcement learning with Soar. *Cognitive Systems Research*, 6(1), 51-59.

Nelder, J. & Wedderburn, R. (1972). Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General)*, 135 (3), 370–384.

Nosofsky, R. A., & Zaki, S. R. (2002). Exemplar and prototype models revisited: Response strategies, selective attention, and stimulus generalization. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 28(5), 924-940.

Nuxoll, A. M., & Laird, J. E. (2007). Extending cognitive architecture with episodic memory. *Proceedings of the 22nd AAAI National Conference on Artificial Intelligence.*

Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science, 12*(5), 410-430.

Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1), 81-106.

R Development Core Team. (2010). R: A languate and environment for statistical computing. http://ww.R-project.org.

Rosch, E. (1973). Natural categories. *Cognitive Psychology*, 4(3), 328-350.

Rosch, E. (1978). Principles of categorization. *Cognition and Categorization*, 27-48. John Wiley & Sons Inc.

Rouder, J. N., & Ratcliff, R. (2006). Comparing exemplar-and rule-based theories of categorization. *Current Directions in Psychological Science*, 15(1), 9-13.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing*, 318–362.

Russell, S. J., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Schölkopf, B., & Smola, A. J. (2001). *Learning with Kernels.* The MIT Press.

Settles, B. (2004). Biomedical named entity recognition using conditional random fields and rich feature sets. *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA).*

Sha, F., & Pereira, F. (2003). Shallow parsing with conditional random fields. *Proceedings of Human Language Technology, NAACL.*

Sharp, H. (1968), Cardinality of finite topologies. *Journal of Combinatorial Theory* 5, 82–86

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. *Proceedings of Neural Information Processing Systems 7*, 361–368.

Smith, J. D., Chapman, W. P., & Redford, J. S. (2010). Stages of category learning in monkeys (Macaca mulatta) and humans (Homo sapiens). *Journal of Experimental Psychology: Animal Behavior Processes*, 36(1), 39-53.

Sutton, R.S. (1996). Generalization in reinforcement learning: successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8*, pp. 1038-1044. MIT Press.

Sutton, R. S., & Barto, A. G. (1998). Reinforcement Learning: An Introduction. *Adaptive Computation and Machine Learning*. The MIT Press.

Thorndike, E. L. (1911). *Animal Intelligence*, Macmillan.

Vanpaemel, W., & Storms, G. (2008). In search of abstraction: The varying abstraction model of categorization. *Psychonomic Bulletin & review*, *15*(4), 732-749.

Vishwanathan, S.V.N., Schraudolph, N. N., Schmidt, M. W., & Murphy, K. (2006). Accelerated training of conditional random fields with stochastic meta-descent. *International Conference on Machine Learning (ICML)*, 969–976.

Vollbrecht, H. (2000). Hierarchical function approximation in kd-Q-learning. *Proceedings of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, 466-469

Walsh, T. J. (2010). Efficient learning of relational models for sequential decision making. Ph.D. thesis, Rutgers, The State University of New Jersey, New Brunswick, NJ.

Wang, Y., & Laird, J. E. (2007). The importance of action history in decision making and reinforcement learning. *Proceedings of the 8th International Conference on Cognitive Modeling (ICCM)*. Ann Arbor, MI

Wang, Y., & Laird, J. E. (2010). A computational model of functional category learning in a cognitive architecture. *Proceedings of the 10th International Conference on Cognitive Modeling (ICCM)*. Philadelphia, PA

Wang, Y., & Laird, J. E. (2010). Efficient value function approximation with unsupervised hierarchical categorization for a reinforcement learning agent. *Proceedings of the 9th International Conference on Intelligent Agent Technology (IAT)*. Toronto

Whiteson, S., Taylor, M. E., & Stone, P. (2007). Adaptive tile coding for value function approximation. *Technical Report AI-TR-07-339,* University of Texas at Austin.

Wintermute, S. (2010). *Abstraction, Imagery, and Control in Cognitive Architecture* (PhD Thesis). University of Michigan.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

Yuan, Y, & Shaw, M. J. (1995). Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69, 125–139