A Neuroidal Architecture for Cognitive Computation

A Neuroidal Architecture for Cognitive Computation

Leslie G. Valiant

TR-04-98

Computer Science Group
Harvard University
Cambridge, Massachusetts

A Neuroidal Architecture for Cognitive Computation*

Leslie G. Valiant
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138

Abstract

An architecture is described for designing systems that acquire and manipulate large amounts of unsystematized, or so-called commonsense, knowledge. Its aim is to exploit to the full those aspects of computational learning that are known to offer powerful solutions in the acquisition and maintenance of robust knowledge bases. The architecture makes explicit the requirements on the basic computational tasks that are to be performed and is designed to make these computationally tractable even for very large databases. The main claims are that (i) the basic learning tasks are tractable and (ii) tractable learning offers viable approaches to a range of issues that have been previously identified as problematic for artificial intelligence systems that are entirely programmed. In particular, attribute efficiency holds a central place in the definition of the learning tasks, as does also the capability to handle relational information efficiently. Among the issues that learning offers to resolve are robustness to inconsistencies, robustness to incomplete information and resolving among alternatives.

1 Introduction

We take the view that intelligence is a large scale computational phenomenon. It is associated with large amounts of knowledge, abilities to manipulate this knowledge to derive conclusions about situations not previously experienced, the capability to acquire more knowledge, and the ability to learn and apply strategies of some

*This research was supported in part by grants NSF-CCR-95-04436, ONR-N00014-96-1-0550, and ARO-DAAL-03-92-G-0115

complexity. The large scale nature of the phenomenon suggests two prerequisites for constructing artificial systems with these characteristics. First, some theoretical basis has to be found within which the various apparent impediments to this endeavor that have been identified can be addressed systematically. Second, some large-scale experiments need to be conducted to validate the suggested basis – it is possible that the fundamental phenomena do not scale *down* and that small-scale experiments do not throw light on them.

Much effort has been devoted to identifying such a theoretical basis. One major thrust has been to develop definitions of capabilities that are functionally adequate. Functionalities that, even if realized, would not go far towards achieving a significant level of performance are of little interest. Another thrust has been the search for capabilities that are demonstrably computationally feasible. Functionalities that are computationally intractable are again of little direct interest. A third viewpoint is that of biological plausibility. Perhaps an understanding of how cortex is constrained to perform these tasks would suggest specific mechanisms that the other viewpoints are not able to provide.

The hypothesis of this paper is that for intelligent systems to be realized the sought after theoretical basis has not only to be discovered, but needs to be embodied in an *architecture* that offers guidelines for constructing them. Composed as these systems will be of possibly numerous components, each performing a different function, and each connected to the others in a possibly complex overall design, there will need to be some unity of nature among the components, their interfaces, and the mechanisms they use.

We shall describe a candidate for such an architecture. This candidate emerged from a study that attempted to look at the issues of functional adequacy, computational feasibility and biological constraints together [52]. We call the architecture *neuroidal* since it respects the most basic constraints imposed by that model of neural computation. One feature of that study was that it was a "whole systems" study. It addressed a range of issues simultaneously insisting on biological and computational feasibility, plausible and simple interactions with the outside world, and adequate explanations about the internal control of the system. While definitions of intelligence, thinking and consciousness have all proved elusive, it appears that the basic computational substrate that neural systems have to support to realize these phenomena might be identified more easily. We believe that the functions and mechanisms associated with this substrate characterize an area and style of computation that is suitably referred to as *cognitive computation*.

In abstracting an AI architecture from that neural study, as we do here, we dispense with some of the most onerous constraints considered there to be imposed by biology. In the mammalian brain these constraints include, among others, the sparsity of the interconnections among the components, the inaccessibility of the separate

components to an external teacher or trainer, and certain bounds on the weights of individual synapses. In artificial systems these constraints are not fundamental, and in order to maximize the computational power of the architecture we will free ourselves of them here.

Our purpose in describing the architecture is to suggest it as a basis for large scale experiments. The first critical question we ask is whether it is theoretically adequate: Can some of the obstacles that have been identified by researchers be solved, in principle, within the model? Are there further fundamental obstacles? The second critical question, which will need to be resolved empirically, is whether systems based on this architecture can indeed be constructed, that exhibit the phenomena outlined in our first paragraph to a more significantly extent than current systems.

As to the first question we observe that the McCulloch-Pitts model may be viewed as a valid architecture for intelligence, if one accepts threshold elements as representing the basic steps of cortical computation. Its shortcoming, clearly, is that it does not appear to offer useful guidelines for constructing systems.

Our architecture, in contrast, is designed to provide a design methodology. In particular, its main feature is that it comes with a set of associated algorithms and mechanisms. The basic constituents of the architecture are classical enough, being *circuit units* consisting of linear threshold elements, and short term memory devices called *image units*. The novelty is that for aggregates of such devices we can detail how the accompanying mechanisms can perform, at least in principle, a list of tasks that address significant problems. The bulk of this paper is devoted to enumerating these problems and describing how they can be addressed. Our purpose here is to point out that this broad variety of mechanisms can be supported on this single unified architecture, and that together, they go some way toward addressing an impressive array of problems.

There will remain the second question as to how to validate the architecture. As we shall suggest, this raises pragmatic issues that had previously received only limited attention. The mechanisms provided emphasize the centrality of learning for multiple purposes, from basic knowledge acquisition to ensuring robust behavior. To construct a system one would need to do a high level design as well as, possibly, some programming, although the main benefit of the architecture is the potential for *massive knowledge infusion* by learning.

If one is to realize a capability for massive knowledge infusion, we expect that there will be needed a significant new kind of effort directed towards the *preparation of teaching materials*. Ideally available resources such as dictionaries and annotated corpora of text should be usable. However, since the system will not operate identically to humans, the teaching materials needed cannot be expected to be identical to those that are effective for humans. The system and teaching materials need to be chosen so as to fit each other, and we believe that the development of these in tandem is a

major challenge for future research. Although, clearly, there is much data available in natural language form, aside from the issue of natural language understanding, there is the independent question of whether these materials, without annotations, make explicit all the commonsense and other information that would be needed.

To summarize, the main feature of the architecture is that it brings learning to the heart of the general AI problem. Recent advances in machine learning help us to distinguish those aspects of learning that have very effective computational solutions, from those for which none is known. Thus in a learning task the problem of finding new compound features that are particularly effective is not one to which general effective solutions are known. In contrast, we know that the presence of irrelevant features can be tolerated in very large numbers, without degrading learning performance, for some classes of functions. Our proposal is that these insights be incorporated into the design of AI systems. We claim that this approach does offer a new view of some of the now traditional issues of AI. It also raises some new questions particularly with regard to the training of such systems.

On a historical note it is interesting to recall Turing's 1950 paper on "Computing Machinery and Intelligence" in which he describes his "Test". While he considers the possibility that a programmed system might be built to solve it, he appears to come out in favor of a learning machine. In the subsequent history of research on general intelligent systems, the overwhelming emphasis has been on programmed systems. We believe that this was largely because there existed a mathematical basis for following this avenue, namely mathematical logic. However, in the intervening years there has been much progress both on the theoretical foundations of machine learning, as well as in the experimental study of learning algorithms. It is, therefore, appropriate now to reexamine whether a synthesis can be found that resolves Turing's dilemma.

2 The Architecture

A system having the proposed architecture is composed of a number of *circuit units* and a number of *image units*. The image units can be thought of as the *working memory* of the system. During reasoning processes that is where the intermediate results of the reasoning computations are stored. The circuit units are the long term repositories of knowledge. Their behavior can be updated *by both inductive learning and explicit programming*.

The circuit units are defined so as to ensure that they provably have certain desirable learning capabilities. For this reason, we define them to consist of one layer of linear threshold gates or elements. The circuit units have one layer of inputs that are the inputs to these gates, and one layer of outputs that are the outputs of the gates themselves. The weights of the threshold elements can be either learned or

programmed.

The intention is that each gate recognize some predicate. We say that it "fires" if such recognition has occurred in the sense that the output has taken value one or "true". The inputs may be Boolean or real valued. The latter is needed if some of the input devices or image units produce numerical values (or if we go outside the strict model by allowing, for example, the outputs, of gates to be numerical functions of the linear sums.).

The image units are short term repositories of information with unspecified format. Their outputs are regarded as preprogrammed *feature* detectors and the inputs include preprogrammed *inverse features*. For Boolean outputs, each output feature will have value one for some information contents and zero for others. As input the image may take information from a sensory device such as a camera. In addition the inputs to an image unit may be of the form of inverse features that modify the contents of the image so as to make some associated feature detectors fire. For example, the image unit may have a feature detector that fires if the image contains a depiction of an elephant, and it may also have an inverse feature that can create in the image the illusion of an elephant in the sense that the information it creates in the image makes the elephant feature detector fire. Thus image units can be used both to store information received from the outside world through sensory devices, as well as to store "imaginary constructs" that are useful for the system's internal computational processes. It is possible to have a number of such constructs in the image simultaneously in a novel combination. The system can then bring the power of its circuits to bear on a representation of a complex situation not previously represented in it. Both preprogrammed features and preprogrammed inverse features can be composed using the circuit units to create nodes that act as higher level features or inverse features. Thus the elephant instance may be more plausibly realized at a higher level in a hierarchy.

The circuit and image units will be composed together in a block diagram so that the outputs of some of the units are identified with the inputs of others. Typically image units will interface directly with circuit units, rather than other image units. Some inputs are identified with the outputs of input devices, and some outputs with the inputs of output devices. The block diagram may ultimately contain feedback or cycles.

Some rules are further specified for how the circuits can change in the process of knowledge acquisition. For the gates, linear threshold units in our case, some update rules are given, such as the perceptron algorithm [42] or Winnow [26], that specify the supervised inductive learning process. In addition some further rules are given to allow the acquisition of programmed knowledge. In particular new output gates may be added and the parameters or weights of that gate assigned appropriate values so that it fires under the intended conditions. These added gates can then also serve as

targets for inductive learning or as inputs to other units. One method of adding to the circuit is to add gates to represent various fixed functions of existing nodes so as to enrich the feature space. Thus one may add nodes that compute the conjunction of all pairs from a certain set of existing nodes. Another method of programming a circuit is to change a weight between an existing pair of nodes. Thus, one can create a subcircuit to realize "grass $\Rightarrow$ green" by making the weight from the node representing "grass" to that representing "green" larger than the threshold.

2.1 Overview

We shall outline here the properties that are required for a device to be an image unit. In the section to follow we shall describe one particular realization of image units that is based on predicate calculus and highlights how our approach can be used to generalize the capabilities of logic based AI systems.

An image unit will at any instant contain some data that we call the *scene* S. The scene itself consists of a number of *objects* $a_1, \cdots, a_s$ as well as information about the properties of the objects and about the relationships amongst them. This information may be represented, in principle, in any number of ways, of which predicate calculus is one.

The computational *processes* that we describe are entirely in terms of interactions between the circuit units and the image units. Since we are free to limit the number of *objects* in the image units to some moderate number Ω, say equal to 10, we are able to limit *ab initio* the computational cost of the so-called binding problem, that grows with this number.

Even when we use standard logical notation to describe the contents of the image or to label circuit nodes, our interpretation of it is slightly different from the usual. In particular the objects in our architecture are best viewed as internal artifacts in the system. They are not intended to refer to the external world in the same direct way as in more familiar uses of the predicate calculus. Further, the primary *semantics* that we ascribe to nodes in the system are the PAC semantics described in the next section, rather than the standard semantics of predicate calculus.

Each node of a circuit unit is in some state. Most simply, it is in one of two states that indicates whether or not the predicate is true of the current scene in the image unit. However, it can contain further information, such as a real number that expresses a confidence level for the validity of the predicate. (c.f. neuroids in [52]).

Each node can be thought of as representing an existentially quantified relation, e.g.

$$\exists x_1 \; \exists x_2 \; \exists x_3 \; R(x_1, x_2, x_3)$$

where the existentially quantified variables range over the objects $a_1, \cdots, a_\Omega$ in the scene. The threshold will *fire*, in general, if this expression holds for the current scene. While no general assumptions are made about how R is represented, it is assumed that when this relation is recognized to hold for the current scene, a corresponding *binding*, or mapping θ from $\{x_1, x_2, x_3\} \rightarrow \{a_1, \cdots, a_\Omega\}$ is made explicit in the system. For brevity, where this leads to no ambiguity, we shall use R variously to refer to the predicate computed, the value of the predicate or to the node itself.

An aggregate of circuit units is a directed graph in which each pair of nodes that is connected is associated with a *weight*. Each node has an update function that updates its state as a function of its previous state, the states of nodes from which there are directed edges to it as well as the weights of these edges. More particularly each circuit unit is a directed graph of depth one (i.e. one layer of input nodes, one layer of output nodes, and no "hidden layer") and the update functions are linear threshold functions.

An important aspect of the circuits is that a directed edge from node representing R_1 to a node representing R_2 contains binding information called the *connection binding* $R_1 \rightarrow R_2$. If the nodes represent $\exists x_1 \exists x_2 \exists x_3 R_1(x_1, x_2, x_3)$ and $\exists y_1 \exists y_2 \exists y_3 R_2(y_1, y_2, y_3)$ then the binding information could specify, for example, that x_1 and y_2 have to represent the same object but that x_2, x_3, y_1 and y_3 may be objects arbitrarily different from each other. For this reason there may be *several* connections from R_1 to R_2, each one corresponding to a different connection binding and having a different weight. For example, if R_2 represents the notion of grandparent and R_1 that of parent then in the definition of the former one may wish to invoke the latter twice with different bindings.

The scene S in an image unit contains information about which relations hold for which object sets. An aggregate of circuit units can be evaluated for S node by node. For a node that represents relation $\exists x_1, \cdots, \exists x_i R(x_1, \cdots, x_i)$ the set of all bindings $\theta\{x_1, \cdots, x_i\} \rightarrow \{a_1, \cdots, a_\Omega\}$ that make R hold will be evaluated. If the node R has inputs from nodes R_1 and R_2, and the gate at R evaluates the threshold function $R_1 + R_2 \geq 2$, then R will fire for a binding θ if *there exist* bindings θ_1, θ_2 of the variables of R_1 and R_2 that respect the respective connection bindings $R_1 \rightarrow R$ and $R_2 \rightarrow R$, and make both R_1 and R_2 true. We shall expand on this in the next section.

2.2 An Implementation

We shall describe an implementation that is based on predicate calculus, but extends it in the direction advocated in this paper, so as to allow for effective learning, with attribute efficiency and error resilience. In particular, a programmed system in which knowledge is described as *Horn clauses*, and *modus ponens* is used as the rule of inference, can be embedded into this framework. The central role in AI systems of

this language of description is discussed by Russell and Norvig ([45] pp 265-277). We are claiming, therefore, that in at least this one possible implementation of our architecture, it is the case that a significant class of existing programmed systems can be embedded. Systems so embedded would then have the addition benefits of a powerful learning capability.

Let us consider the following restriction of the language of predicate calculus (e.g. [45] p. 186). As constants we will choose the base objects $A_B = \{a_1, \cdots, a_\Omega\}$ of the image unit. We allow a set R_B of base relations of arities that are arbitrary but upper bounded by a constant α. We use no function symbols. We represent variable names by $\{x_1, \cdots, x_n\}$ and relations will be defined in terms of these.

The two features of predicate calculus that we exclude, therefore, are constants that refer directly to individuals in the world (e.g. Shakespeare), and functions that can be applied to them to refer to other individuals (e.g. Mother_of (Shakespeare).). In our system the only constants are the predefined base objects of the image, which are internal constructs of the systems. In order to refer to an individual like Shakespeare we shall use a unary predicate that can be applied to a base object. Thus Shakespeare(a_3) would express the fact that a_3 has the attributes of Shakespeare. Also, we dispense with function symbols by referring to the resulting object as a base object, and by expressing the necessary relation by an appropriate relation symbol. Thus instead of saying $x_1 =$ Mother_of(x_2) we would say Mother_of(x_1, x_2), where the latter is a binary relation. In these ways we ensure that the two linguistic restrictions, on constants and functions symbols, do not restrict what can be expressed.

A *term* will be therefore a base object a_i and an *atomic sentence* will be a single relation such as Mother_of $\in R_B$ applied to one or a collection of base objects (e.g. Mother_of(a_3, a_7).) A Horn *rule* will be an implication

$$R_{i_1}(x_{i_1,1}, \cdots x_{i_1,\delta(i_1)}) \ \wedge \cdots \ \wedge \ R_{i_r}(x_{i_r,1}, \cdots, x_{i_r,\delta(i_r)}) \\ \Rightarrow R_{i_{r+1}}(x_{i_{r+1},1}, \cdots, x_{i_{r+1},\delta(i_{r+1})}). \qquad (1)$$

where $R_{i_j} \in R_B$ and $\delta(k)$ is the arity of $R_k \in R_B$. We note that R_B may include the relation False that has zero arguments, that signifies logical impossibility and may be used meaningfully for $R_{i_{r+1}}$ on the right-hand side of an implication. Implications are to be interpreted as quantified universally over all the x variables that occur in them but these quantifiers are omitted for brevity.

A *binding* is a mapping from $\{x_1, \cdots, x_n\}$ to $\{a_1, \cdots, a_\Omega\}$. We say that a relation $R(x_1, x_2, x_3)$ is made true by θ if $R(\theta(x_1), \theta(x_2), \theta(x_3))$, or $R(\theta(\underline{x}))$ for short, holds. The derivation rule *modus ponens* is the following:

> "if $R_{i_1}(\underline{x}) \ \wedge \ \cdots \ \wedge \ R_{i_r}(\underline{x}) \ \Rightarrow R_{i_{r+1}}(\underline{x})$" is a rule,
> and if, for some binding $\theta, R_{i_j}(\theta(\underline{x}))$ holds for $1 \leq j \leq r$,
> then $R_{i_{r+1}}(\theta(\underline{x}))$ also holds.

In a logic based system we would program a set of rules of the form (1) and consider the input to be a set of atomic sentences. We could then consider the output to be the set of all atomic sentences that can be deduced from the input by applying the rules using modus ponens in all possible ways. This output could be derived in the logical framework by means of forward chaining ([45]p. 273.).

Let us now describe the implementation of the neuroidal architecture into which this process embeds naturally. In this implementation the contents of the image will be simply the atomic sentences of the input. The circuits will implement the rules as follows: For each relation $R(\underline{x}) \in R_B$ there will be a gate in the circuit. We shall assume that each relation R occurs on the right-hand side in just one rule – otherwise we replace multiple occurrences of R on the right hand side by different names, say $R^1, R^2, \cdots, R^m$ and add a new rule $R^1 \vee R^2 \vee \cdots \vee R^m \Rightarrow R$.

For each Horn rule $R_{i_1} \wedge R_{i_2} \wedge \cdots \wedge R_{i_r} \Rightarrow R_{i_{r+1}}$ we shall make a *connection* to $R_{i_{r+1}}$ from each R_{i_j} $(1 \leq j \leq r)$. This connection will have the appropriate *connection binding*, defined below. At the $R_{i_{r+1}}$ node we shall implement the equivalent of an AND gate in terms of a threshold gate with threshold r. In other words, we regard the values of the R_{i_j} as Boolean $\{0, 1\}$, and have a gate that realizes.

$$R_{i_{r+1}} = 1 \quad \text{if and only if} \quad \sum_{j=1}^{r} R_{i_j} \geq r.$$

Executing this threshold gate will correspond, therefore, to performing one application of modus ponens.

To simulate OR gates, which are needed if multiple occurrences of the same relation occurs on the right hand side, we also use a threshold gate but have 1 as the threshold instead of r, i.e. $\sum R_{i_k} \geq 1$.

The further detail that needs to be clarified is the nature and meaning of the connection bindings. Each rule with r relations on the left defines r connections and connection bindings. Consider the following rule with $r = 2$:

$$R_1(x_1, x_2, x_3) \wedge R_2(x_3, x_4, x_5) \Rightarrow R_3(x_2, x_3, x_5). \tag{2}$$

This notation expresses the connection bindings implicitly. The binding of the connection from R_1 to R_3 says simply that the second parameter of R_1 binds with the first parameter of R_3 and the third parameter of R_1 with the second of R_3, and that there are no other constraints. The naming of the variables in each rule or gate can express this precisely. Note that in this example the implication is that any binding of x_1 that makes R_1 true can be combined with any binding for x_4 that makes R_2 true. For example, if elsewhere, we have the rules $R_4(x_7) \Rightarrow R_1(x_7, x_8, x_9)$ and $R_4(x_7) \Rightarrow R_2(x_{10}, x_7, x_{11})$, then it will be sufficient for these rules to be satisfied for distinct values of $\theta(x_7)$ since (2) did not require otherwise.

If *from* some R_i there is more than one connection, then for conceptual purposes it is simplest to think about a circuit in which the R_i is replicated so that each node has just one connection directed away from it and hence the circuit is a tree. In other words connection bindings restrict the multiple inputs to a node but never the multiple outputs. Note also that any correspondence between two variables occurring in two relations on the left hand side (i.e. x_3 in this example,) can be enforced only by having x_3 as an explicit variable in the right hand side relation, (i.e. R_3 in the example.) This can be circumvented by defining a node that represents a particular compound relation e.g. we could create a gate for computing $R_1(x_1, x_2, x_3) \wedge R_2(x_3, x_4, x_5)$, call it $R_5(x_2, x_5)$ and have an implication $R_5(x_2, x_5) \Rightarrow R_3(x_2, x_5)$ if we wish to avoid mentioning x_3 in R_3.

In this implementation we consider the network to evaluate for each gate all *possible* bindings $\theta : \{x_1, \cdots, x_n\} \rightarrow \{a_1, \cdots, a_\Omega\}$ so as to find all the ones, if any, that make the gate evaluate to one. To do this we shall, for simplicity, impose here the constraint that aggregates of circuit units are acyclic. We can then form a topological sort of their nodes (e.g. Knuth [23]) and for one such topologically sorted order evaluate each node in succession. The evaluation of each node is for all the Ω^d bindings θ, where $d < \alpha$ is the arity of the relation R at that node. (Allowing cycles would allow recursive rules to be expressed, but would make the evaluation mechanisms more complex. If the circuits express Horn rules, or other rules in which no negative weights occur, then evaluating circuits with cycles can still be done in time polynomial in Ω^α and the number of gates.)

In this graph the input nodes with no predecessors will represent relations themselves. To evaluate a gate at R, we simply enumerate all the Ω^d bindings of the d variables that appear in it. First we scan all the atomic sentences in the image that contain R and see which bindings make R true, before any rules are applied. Then for each binding, and for each predecessor node, if any, we determine whether that binding can make the predecessor true. Having done this for each predecessor, we can, for each binding compute the value of the gate at the current node, whether it is a disjunction, conjunction, or more generally, an arbitrary linear threshold gate. Note that the complexity of this task that is contributed by the binding problem is Ω^d. It is exponential in the *maximum arity* of the relations, and not in the number of base objects or the number of relations in a rule! (Note, however that this arity may be made larger by the restriction that all binding information is in the connections. This necessitates that if some correspondences among the variables need to be enforced on the left hand side of a rule, they must be made explicit on the right hand side. For example if we wish to represent father$(x, z) \wedge$ mother$(z, y) \Rightarrow$ grandfather(x, y) then in our representation grandfather will need to have a third argument, say t, that is to be identified with the two occurrences of z by the connection bindings.)

It is easy to verify that this evaluation algorithm computes all satisfying bindings

for each relation that is represented at some node, in exactly the same way as would applying modus ponens in all possible ways to the rules.

From what has been said it should be clear that our architecture is expressive enough that programmed systems based on the Horn rules we have described and modus ponens can be embedded into it. What the architecture adds to such systems is a capability for learning. The point is that the gates we allow are not just Boolean conjunctions and disjunctions, but linear threshold functions. This allows a host of learning algorithms, discussed in later sections, that provide a provable learning capability that is not known to be available in strictly logical representations.

In conclusion we note that there are theoretical results that show for certain classes of Boolean functions that extending the representation of the learner to threshold functions makes the original class polynomial time learnable, while restricting the learner to the minimal representation needed for expressing these functions would make the task NP-compete [40]. Our richer representation, therefore, has not only the obvious advantage of being able to express more, but has the additional computational benefits of making Boolean domains potentially easier to learn. Also, while the quoted result discusses the polynomial time criterion for learning, as we shall discuss at length, we also desire and seek the further advantages that learning be achieved with attribute-efficiency and error-resilience.

3 Semantics

We cannot expect to develop a set of robust mechanisms for processing representations of knowledge without a robust semantics for the representation. The emphasis here is both on the necessity of *semantics*, that relates the representation to some reality beyond itself, as well as *robustness* to the many uncertainties, changes and errors in the world or in communications with the world, that the system will need to cope with.

The need for semantics has explained the attractiveness of formal logic in AI research. It is the need for robustness that forces us to look beyond logic, at notions centered on learning. The semantics we shall describe here, PAC *circuit semantics*, or PAC *semantics* for short, is based on the notion of computationally feasible learning of functions that are probably approximately correct [50].

To explain the contrast in viewpoints consider the situation calculus described in McCarthy and Hayes [30]. There, a situation is "the complete state of the world", and general facts are relations among situations. Thus $P \Rightarrow Q$ means that for all situations for which P holds Q holds also. This is an all embracing claim about the universe that is difficult to grade gracefully and becomes problematic in the real world where authoritative definitions of P and Q themselves may be difficult to identify.

In contrast, PAC semantics makes qualified behavioral assertions about the computational behavior of a particular system in a particular environment. In PAC semantics P and Q would be defined as functions computed by fixed algorithms or circuits within a system that takes input through a fixed feature set from a fixed but arbitrarily complex world. The inputs range over a set X that consists of all possible combinations of feature values that the input feature detectors can detect. There is a probability distribution D over this set X that summarizes the world in all the aspects that are discernible to the feature detectors of the system. The P and Q could correspond to nodes in a circuit. The relationship between P and Q would be typically of the following form: if a random example is taken from D that satisfies P, then it also satisfies Q with a certain probability. The latter would, at best, be known to be in some range with a certain confidence. Thus the semantics is relative to both the computational and sensory abilities of the system, and refers to an outside world about which direct observations can be made only one observation at a time. The claims that are made about the semantics do not go beyond what is empirically verifiable in a computational feasible way by the system operating in the world. In addition to making observations from D, the system can also acquire rules by being told them. It can then use these as working hypotheses, subject to subsequent empirical testing by the system in the PAC sense, and make deductions using them. The general goal of the system is to learn from D the invariants of the world.

In constructing an artificial system we envisage that each circuit unit can be trained separately. The unit will take as inputs the outputs of input devices or other circuit or image units to which it is connected. Thus it would see the world through a set of features that are themselves filtered through the input devices and circuits of the system. In contrast, the outputs of the unit being trained will be directly accessible to the trainer, who can venture to train each such output to behave as is desired. We note that if the system consists of a chain of circuit units trained in sequence in the above manner then the errors in one circuit do not necessarily propagate to the next. Each circuit will be accurate in the PAC sense as a function of the external inputs – the fact that intermediate levels of gates only approximate the functions that the trainer intended is not necessarily harmful. At each internal level these internal feature sets may permit accurate PAC learning at that next stage.

Since our architecture can perform computations via the image units that are more dynamic than the conventional view of circuits allows the terminology of circuits is best viewed as an analogy. The more general computational functions of the system, including those that use the image for reasoning as outlined in §4.8, for example, all need to be effective in the PAC sense.

The key advantage of PAC semantics is that it gives an intellectual basis for understanding learning and thereby validates empiricism and procedural views of knowledge. Inductive learning will be the key to overcoming the impediments that

are to be enumerated in the next section. These will include defaults, nonmonotonic reasoning, inconsistent data, and resolving among alternatives. Intelligent systems will inevitably meet the dilemmas that these issues raise but they will have a learned response to all but the rarer manifestations of them.

We mention that PAC learning, when offered as a basis for intelligent systems or cognitive science, suggests the following view. The world is arbitrarily complex and there is little hope that any system will ever be able to describe it in detail. Nevertheless by learning some simple computational circuits such systems can learn to cope, even in a world as complex as it is. Most often these circuits will be deterministic. It is the complexities of the world and the uncertainties in the system's interactions with it that force the framework of the semantics to be probabilistic.

Having circuits that have a probabilistic rather than a deterministic interpretation is an extension that may be considered, but this appears to make the learning task computationally less tractable [17]. There is little evidence that probabilistic processes are central to human reasoning [49]. While we do not exclude extensions to probabilistic representations, we do not consider them here.

4 Some Algorithmic Mechanisms

In this section we shall enumerate a series of algorithmic techniques for manipulating knowledge in a system having the described architecture. We claim that these mechanisms address issues that are inescapable for large scale learning based AI systems. The mechanisms described here arose in the main in studies of formal models of various specific phenomena. Our observation is that they can be brought together and adapted to provide an overall methodology within a single framework.

4.1 Conflict Resolution

The circuit units will contain large numbers of nodes. In general each one corresponds to a concept or action, that has some reference to the world or to the internal constructs of the image unit. The semantics of each node can be defined in the PAC sense. Let us suppose that each one when regarded separately, is highly accurate, correct say on 99% of the natural input distribution. The problem that arises is that because of the sheer number of nodes, perhaps in the hundreds of thousands, on almost any one input a large number of the nodes will be inaccurate.

In a natural scene we may expect a certain moderate number of predicates that are represented in the circuits to hold and the corresponding nodes to fire. However, the large numbers of remaining predicates will be represented with some inaccuracy, a certain number of these additional nodes that should not fire will do so also. Further

some of these will be in semantic conflict with the correct ones. They may recommend inconsistent actions (e.g. "go left" as opposed to "go right") or inconsistent classifications (e.g. "horse" versus "dog".)

This is a fundamental and inescapable issue for which a technical solution is needed. A conventional approach is to suggest that each node be given a numerical "strength", and that in situations where several nodes fire in conflict the one with highest strength be chosen to be the operative one. This approach clearly needs some concrete technical mechanism for deriving the strengths. It also makes the assumption, which needs justification, that a single totally ordered set of numerical strengths is sufficient for the overall resolving process.

Our approach is the following: we have a large number of circuit nodes, computing functions $x_1, \cdots, x_n$, say, of the scene S. We assume that each x_i is correct in the PAC sense with high probability. Because of the sheer size of n, at any one time many of the x_i nodes will fire falsely, and we need a mechanism for resolving among them. The proposed solution is to have another set $y_1, \cdots, y_n$ of nodes where y_i corresponds to x_i. The purpose of y_i is the following: When y_i fires this will be taken to assert that x_i is true, and further that x_i is *preferred* over all the other x_j that may be firing. The implementation will have a circuit unit with x_j $(1 \leq j \leq n)$ as inputs and y_i $(1 \leq i \leq n)$ as outputs, and with a connection from each x_j to each y_i. Each y_i will be a linear threshold functions of the x_j. Thus if x_7, when firing, is to be preferred to all the x_j except for x_2, x_5 and x_8, whose firing should override that of x_7, then the appropriate linear inequality to be computed at y_7 will be

$$x_7 - x_2 - x_5 - x_8 \geq 1. \tag{3}$$

This will have the effect that y_7 will fire if and only if x_7 fires and none of x_2, x_5 or x_8 fires.

The force of this approach is two-fold. First, it is more expressive than the totally ordered strengths regime. For each x_i one can specify which x_j dominate it, independently of the other x_i. Second, the representation needed for expressing the preferences, namely linear threshold functions, are learnable in a very favorable sense as further discussed in subsection 4.2 below.

We are therefore suggesting that the conflict resolution problem can be solved by learning the correct resolutions from examples of past behavior. The justification of our architecture can be viewed as the possibility of repeated use of this same idea: that learning can resolve many otherwise fundamentally problematic issues, and that it can be realized effectively by the algorithms we describe. We chose to discuss the conflict resolution problem here, at the beginning, since it seems a particularly simple and convincing instance.

4.2 Learning From Few Cases

The problem with advocating systems based on massive knowledge bases is that one needs to specify mechanisms for coping with the issues of scale. In the previous section, for example, we suggested that a learning mechanism for linear threshold functions can address the issue of conflict resolution. As we shall invoke learning as the solution to a variety of other issues also, it is necessary to address the problem that the learning process itself has to face in the presence of a massive knowledge base.

The basic issue is fundamental and widely recognized. If there are n functions represented in the system, and each one can depend on any of the $n-1$ others, as a linear threshold (or some other) function, then there are potentially about n^2 parameters. Since n is large, say in the tens of thousands, n^2 is very large. With this backdrop it is a remarkable fact that biological learning systems appear to be able to learn from relatively few examples, certainly much fewer than reasonable estimates of the n (or n^2.) Some mechanism needs to be present to enable very high dimensional systems to learn from numbers of interactions with the world that are very small compared with this dimension.

The most relevant theory we know of how this can be done is that of *attribute-efficient learnability*. The phenomenon here is that for certain function classes of n variables one can *prove* that certain learning algorithms converge to a good hypothesis after a number of examples that depends on n not linearly, the canonical situation from dimensionality arguments, but much more slowly, sometimes *logarithmically*.

The phenomenon of efficient attribute efficient learning in the PAC sense was first pointed out by Haussler [13]. A striking and remarkable embodiment of this idea followed in the form of Littlestone's Winnow algorithm [26] for learning linear threshold functions. The algorithm is similar in form to the classical perceptron algorithm except that the updates to the weights are multiplicative rather than additive. The modification gives the algorithm the remarkable property that when learning a monotone k-variable Boolean disjunction over $\{x_1, \cdots, x_n\}$ the number of examples needed for convergence, whether in the PAC or mistake-bounded sense, is upper bounded by $ck \log_2 n$, where c is a small constant, [26, 27]. Thus the sample complexity is linear in k, the number of relevant variables, and logarithmic in the number of irrelevant ones.

Littlestone's Theorem 9 [26] adapted to the case when coefficients can be both positive and negative (his Example 6) has the following more general statement; For $X \subseteq \{0,1\}^n$ suppose that for the functions $g : X \to \{0,1\}$ there exist $\nu_1, \nu_2, \cdots, \nu_n \geq 0$ and $\bar{\nu}_1, \bar{\nu}_2, \cdots, \bar{\nu}_n \geq 0$ such that for all $(x_1, \cdots, x_n) \in X$

$$\sum_{i=1}^{n}(\nu_i x_i + \bar{\nu}_i(1-x_i)) \geq 1 \quad \text{if} \quad g(x_1,\cdots,x_n)=1$$

and

$$\sum_{i=1}^{n}(\nu_i x_i + \bar{\nu}_i(1-x_i)) \leq 1-\delta \ \text{ if } \ g(x_1,\cdots,x_n)=0.$$

Then WINNOW2 with $\theta = 2n$ and $\alpha = 1+\delta/2$ applied to the variable set $(x_1,\cdots,x_n,1-x_1,\cdots,1-x_n)$ makes at most the following number of mistakes:

$$\frac{16}{\delta^2}\frac{n}{\theta} + \left(\frac{5}{\delta} + \frac{14ln\theta}{\delta^2}\right)\sum_{i=1}^{n}(\nu_i + \bar{\nu}_i). \tag{4}$$

Here θ and δ are parameters of the algorithm and δ, which quantifies the margin by which positive and negative examples are separated, is a parameter of the distribution of examples. For a monotone disjunction of k out of n variables, we can have $\nu_i = 1$ for the k variables in the disjunction, with all the other $\nu_i = 0$, and all $\bar{\nu}_i = 0$. Then clearly $\delta = 1$. Hence (4) becomes $O(k \log n)$. In all these cases the algorithm can be adapted so that it has similar bounds in the PAC model [27].

For linear inequalities of the form (3), we see that the particular example given is equivalent to $\frac{1}{4}(x_7 + (1-x_2) + (1-x_5) + (1-x_8)) \geq 1$ so that $\delta = \frac{1}{4}$. In general if there were k negative terms then $\delta = 1/(k+1)$. In order to make the margin larger it is better to learn the negation of (3), namely $(1-x_7)+x_2+x_5+x_8 \geq 1$ so that $\delta = 1$. The generalization of this to k terms would also give $\delta = 1$ and hence the $O(k \log n)$ bound.

It appears that some mechanism for attribute-efficiency is essential to any large scale learning system. The effectiveness of Winnow itself has been demonstrated in a variety of experiments. A striking example in the cognitive domain is offered in the work of Golding and Roth on spelling correction [11]. Even in the presence of tens of thousands of variables, Winnow is able to learn accurately from few examples, sometimes fewer than 100.

The question arises whether attribute efficient learning is possible for more expressive knowledge representations. Recently it has been found that this is indeed the case. Under a certain projection operation attribute-efficient learning algorithms can be composed to yield algorithms for a more expressive knowledge representation that are still attribute efficient. In subsection 4.4 we shall discuss this further.

Finally, we note that attribute-efficient learning is closely related to the issue of *relevance*, which has been widely discussed in the AI literature. Conventionally one would expect to preprocess data to identify the attributes that are relevant to the classification or action in question. One would then eliminate the irrelevant attributes, and apply a learning algorithm to a database containing only the relevant ones. There is, however, no evidence that biological systems have such an explicit preprocessing

stage. Further, Winnow achieves the same overall effect implicitly, without explicitly identifying which variables are irrelevant in a preprocessing stage. What it offers therefore seems novel and important. (We note however that given an implicit method, such as Winnow, one can with some effort, explicitly identify the relevant variables by a binary search technique that needs about $k \log n$ applications of Winnow. This is an observation of A. Beimel.)

4.3 Learning Relations

The important point about our representation of a relation $R(x_1, \cdots, x_k)$ at a node is its duality of nature. It is Boolean in the sense that at any instant the node either fires or does not fire. On the other hand it is also relational in that at any instant it has some binding $\theta : \{x_1, \cdots, x_k\} \rightarrow \{a_1, \cdots, a_\Omega\}$, and the truth value taken depends on whether the relation holds for this particular binding of the variables to the objects in the image. This dichotomy exists also in other work on learning relations, such as inductive logic programming [45], but needs to be addressed in a different way here because of the circuit orientation. An overriding concern for us throughout is, of course, that the complexity of manipulating relations be controlled (c.f. [46]).

Suppose we have a connection. $R_1(x_1, \cdots, x_n) \rightarrow R_2(y_1, \cdots, y_k)$ and an associated connection binding that specifies $x_1 = y_2$ and $x_2 = y_3$. What does a strong weight on this connection mean? The interpretation of the semantics of circuit evaluation defined in § 2.1 and § 2.2 implies that a strong weight means that for any scene and any bindings of x_1, x_2 to objects for which R_1 is true, it is the case that R_2 should be "inclined" to be true also for *any* binding θ_2 s.t. $\theta_2(y_2)$ and $\theta_2(y_3)$ agree with the bindings of x_1, x_2. In other words, for any scene as far as the truth of R_2 is concerned for any one binding θ_2, the only influence of R_1 is via the question of whether there *exists* some θ_1 that makes R_1 true and agrees with θ_2 on the object pairs specified in the connection binding $R_1 \rightarrow R_2$.

Clearly we need to use the same semantics for learning as for evaluation. Further this is easy to do. If R is a node with connections from $R_1, \cdots, R_m$ and with m corresponding connection bindings, then during learning we shall for each *relational example* (i.e. a scene and a labelling of R for some or all bindings θ of its variables) evaluate $R_1, \cdots, R_m$ for each binding of their own variables. An example for the learning algorithm will then consist simply of truth values for $R_1, \cdots, R_m$ and a truth value for R. Hence the truth value taken of R is simply its truth value on θ. Also R_i will be taken to be true if and only if there exists *some* binding θ_i of the variables of R_i that makes R_i true and agrees with θ on the connection binding $R_i \rightarrow R$.

If the learning algorithm learns a function (of R in terms of $R_1, \cdots, R_m$,) that is consistent with the examples so presented, then the function that the resulting circuit evaluates will be consistent with these examples.

Given one relational example, the distribution of Boolean examples presented to the learning algorithm is not necessarily uniquely determined. The case in which all bindings of R are considered, and each given the same probability is only one choice, be it a natural one. If the vast majority of bindings give negative values for the truth of R, it may be advantageous for reasons of economy to sample from the negative examples. Also, in the case that cycles are allowed and the evaluation process recomputes a node several times, there are further choices to be made.

4.4 Learning More Complex Functions and Strategies

The previous section showed that the learning of linear inequalities with large margins can be done attribute efficiently, and therefore that mechanisms for doing that are ideal building blocks for our architecture. Clearly the intention of our architecture is that each new function that is learned or programmed be expressible in terms of old ones already represented in the system in a reasonably simple way. When discussing learning the crucial issue is *how far removed* the new function can be allowed to be from the old ones already represented, without necessitating a learning capability that is computationally intractable. In other words the issue is one of the *granularity* of the *relative* complexity of the successive functions that can be learned.

From what we have said linear threshold functions offer a level of granularity that is computationally attractive. The two questions raised therefore are: (1) is this level of granularity sufficiently large to offer a convincing approach to building cognitive systems and (2) can this granularity be enlarged (i.e. to richer knowledge representations) while retaining attribute efficient learnability.

Even when a function class is expressible as linear inequalities, this representation may be impractically large if many compound features need to be created. For example, for Boolean variables $\{x_1, \cdots, x_n\}$ each of the 2^{2^n} Boolean functions can be expressed as a disjunction of monomials over $\{x_1, \cdots, x_n, \bar{x}_1, \cdots \bar{x}_n\}$ and hence as a linear inequality, but this requires a variable for each potential monomial, and there are 3^n of these.

Examples of function classes that can be expressed as inequalities over just n variables are: disjunctions, conjunctions, and threshold-k functions. In the last case we would have

$$\sum_{i=1}^{n} z_i \geq k$$

where z_i is a variable over the reals that is given value 1 if $x_i = 1$, and zero otherwise. A further class that can be so expressed is that of 1-decision lists [41]. These test for a sequence of literals $y_{i_1}, \cdots, y_{i_m}$ where $y_{i_j} \in \{x_1, \cdots, x_n, \bar{x}_1, \cdots, \bar{x}_n\}$. If the literal y_{i_j} is true then the function is defined to be the constant c_j for $c_j \in \{0, 1\}$, otherwise

$y_{i_{j+1}}$ is tested. Decision lists can be expressed as linear inequalities over n variable $z_1, \cdots, z_n$ corresponding to $x_1, \cdots, x_n$. The size of the coefficients grows exponentially, however, with n. In the special case that most of the c_j have one value, this growth is more limited. In particular, it is shown in [55] that if c_j has value 1 for d of the m values of j then the decision list can be expressed as a linear inequality with integer coefficients, where the magnitudes of the coefficients, and also their sum, is upper bounded by $(2m/d)^d$. If $d << m$ then this is much better than the general upper bound of 2^m. Also, we see that this inequality can be expressed so as to fit the requirements of expression 4.2 for Winnow. We then have $\delta = (2m/d)^{-d}$ and the sum of the magnitudes of the coefficients bounded by 1, so that the mistake bound is quadratic in $(2m/d)^d$ and logarithmic in n.

The question of characterizing the classes of linear inequalities that are learnable attribute efficiently remains unsettled. The possibility that decision lists can be so learned has not been excluded.

Another question is whether attribute efficient learning can be achieved by classes of functions beyond linear inequalities. A positive answer to this is provided by projection learning, which is is a technique that has been shown to extend the scope of attribute efficient learnability. It allows algorithms that are attribute efficient to be composed so as to obtain learning algorithms for more expressive representations that are still attribute efficient. Since Winnow is the paradigmatic attribute efficient algorithm currently known, the present applications of projection learning are based on Winnow itself.

The basic idea is that for Boolean variables $x_1, \cdots x_n$ we define a set $J = \{\rho_1, \cdots, \rho_r\}$ of *projections* that each map $\{0,1\}^n$ to $\{0,1\}$. For example, we could have $r = 2n$ and for each literal $\ell \in \{x_1, \cdots, x_n, \bar{x}_1, \cdots, \bar{x}_n\}$ we have a projection ρ_ℓ defined as $\rho_\ell(\underline{x}) = 1$ iff $\ell = 1$ on $\underline{x}$. This is the class of single variable projections. An alternative class has $r = 2^k$ with each member of J being a conjunction $l_1 l_2 \cdots l_k$ where $l_i \in \{x_i, \bar{x}_i\}$, i.e. if $\rho = x_1 \bar{x}_2 \bar{x}_3$ then $\rho(\underline{x}) = 1$ iff $x_1 \bar{x}_2 \bar{x}_3 = 1$.

For each $\rho \in J$ we consider the function $f(\underline{x})\rho(\underline{x})$. Clearly this equals zero for $\underline{x}$ s.t. $\rho(\underline{x}) = 0$, and it equals $f(\underline{x})$ otherwise. The hope is that for some of the restrictions ρ, the function $f(\underline{x})\rho(\underline{x})$ can be learned more accurately than $f(\underline{x})$ can directly, and that between the various choices of ρ, the $f(\underline{x})\rho(\underline{x})$ that are learned do cover the whole domain $\{0,1\}^n$ of $f(\underline{x})$.

Suppose that for the various choices of $\rho \in J$ the function learned that approximates $f(\underline{x})$ on $\rho(\underline{x}) = 1$ is $f'_\rho(\underline{x})$. Then $\sum f'_\rho(\underline{x})\rho(\underline{x})$ is taken as the approximation of f that has been learned. The main result in [55] states that if the $f(\underline{x})\rho(\underline{x})$ belong to a class that is learnable attribute efficiently on the restricted domain $\{\underline{x} \mid \rho((\underline{x}) = 1\}$ by an algorithm A say, and if a disjunction can be learned attribute efficiently by an algorithm B that shares certain specified properties with Winnow, then the function $\sum f_\rho(\underline{x})\rho(\underline{x})$ can be learned attribute efficiently, in the sense that the needed sample

complexity depends linearly on the number of relevant $\rho's$ and the number of relevant variables in all the f_ρ, but only logarithmically on the total numbers r and n.

There is a variety of directions for which this can be used to extend attribute efficient learning beyond linear threshold functions.

First, suppose that we learn a function of the form $\sum_{\rho \in \bar{J}} f_\rho(\underline{x})\rho(\underline{x})$ where $\bar{J} \subseteq J$ for J the class of single variable projections, and f_ρ are conjunctions. Then if the conjunctions are assumed to be of length at most k, and the number of elements of $\bar{J}$ is at most m, then the sample complexity of learning these will be linear in m and k, and logarithmic in r and n. Furthermore, the circuit will need r nodes for the various f_ρ, n nodes for $x_1, \cdots, x_n$, and therefore rn weights to update, or $O(n^2)$ in the case of single variable projections. Thus we are learning somewhat economically a subclass of DNF formulae that have m monomials, each with at most $k+1$ literals. To learn these via general $(k+1)$-DNF learning methods [50] would be much more expensive, and intractable if r and n are both large. For example, if we constructed all the $(k+1)$-monomials we would have about n^{k+1} of them, which exceeds n^2 if $k > 1$.

A second class of functions is given by $\sum_{\rho \in R} f_\rho(\underline{x})\rho(\underline{x})$ where J is unchanged but f is the class of disjunctions. Here the disjunctions will be assumed to contain at most k literals, and the sum $\sum_\rho$ to contain at most m nonzero terms. The result will again be a subclass of DNF with at most 2 literals in each of the at most km terms. While the overall complexity of learning this as a 2-DNF is not too different, it would require an architecture with n^2 nodes, rather than the $O(n)$ nodes needed here.

A third way that projection learning can be used is best viewed as an application outside the architecture that is attribute efficient in a weaker sense. Consider a sequential covering algorithm, as in Rivest [41], for learning decision lists, or Khardon's extension to propositional production rule systems [19]. In the simplest case such a covering algorithm works as follows: it looks successively for a literal, say $\bar{x}_3$, that is the most predictive single literal, in some sense, of the function f being learned. For the case $\bar{x}_3 = 1$ it will predict the most likely Boolean outcome for the function. For the remaining subdomain, $x_3 = 1$ in this case, it will then repeat the process of finding the next most predictive literal. Proceeding in this way it will obtain a decision list or production system. We can extend these algorithms by projection learning as follows: Instead of looking for a best literal we look for a best projection ρ from a class J. Also, instead of predicting the most likely result on the subdomain where $\rho = 1$, we learn a new hypothesis for the subdomain $\rho = 1$, and in the decision list structure substitute the learned function, say a conjunction or disjunction instead of true and false, at that point in the list. The procedure is then repeated on the subdomain where either $\rho = 0$ or the last hypothesis is false. This learning procedure can be viewed as attribute efficient if the learning algorithm used in the subdomains are attribute efficient, and if J is chosen to be itself small. Thus a preliminary analysis,

by, for example, simple Winnow may yield a candidate set of variables that are most relevant (e.g. have high weights) and this small set may be chosen as the set J of projections.

We note that the main motivation of projection learning is to learn more effectively in cases in which a representation more complex than linear thresholds is needed. In any one application domain we may have no *a priori* reason to believe that such a representation is necessary. What we expect to find is that projection learning will always yield at least as good results as simple Winnow, (possibly at the expense of more examples) and may yield better results when linear representations are insufficient.

An area in which complex representations may need to be learned is that of strategies and plans. Here production systems are widely believed to have useful expressivity [38]. Khardon has shown that a rich class of these can be learned using decision list algorithms [13]. The dilemma here is that no attribute-efficient learning algorithm is known for decision lists, unless the degree d is small, as explained in the previous section. Hence we may have to look at the flatter projective structures to find representations of such strategies that are learnable attribute efficiently.

4.5 Learning as an Approach to Robustness

Systems will acquire knowledge both by inductive learning as well as by explicit programming. Errors and inconsistencies may occur in either kind of input, and mechanisms are needed for coping with these.

In the case of inductive learning the issue of noise has been studied extensively. On the theoretical side a range of noise models have been considered, ranging from a malicious adversarial model [3] to the more benign random classification noise model, where the only noise is in the classification of the examples and this is random [3]. At least for the more benign models there are some powerful general techniques for making learning algorithms cope with noise in some generality [16].

For the problem of learning linear separators there exist theoretical results that show that there is no fundamental computational impediment to overcoming random classification noise [5, 8]. Currently somewhat complex algorithms are needed to establish this rigorously. In practice, fortunately, natural algorithms such as the perceptron algorithm and Winnow, or the linear discriminant algorithm, behave well on natural data sets which are often noisy, and for which there is no *a priori* reason to believe that linear thresholds should work at all. This empirical evidence lends credence to the use of linear threshold algorithms for complex cognitive data sets.

When knowledge is acquired by programming the issue of coping with noise also arises. The view we take here is that we use the same PAC semantics for programmed rules as for inductively learned rules. Thus the system would have high confidence in a rule that agreed with many examples. A programmed rule is therefore one which is

treated as a working hypothesis, and easily discarded if evidence builds up against it.

In addition to the opportunity to discard rules there is also one for refining them. Thus we may have programmed rules $P \Rightarrow R$ and $Q \Rightarrow \neg R$ as working hypotheses, but discover that they do not hold in all cases. In particular, it may happen that in some relatively rare cases both P and Q hold and therefore the rules contradict each other. It may be that after sufficiently many observations it is found that $P \wedge Q \Rightarrow \neg R$ is a reliable rule. In that case we would refine $P \Rightarrow R$ to $P \wedge \neg Q \Rightarrow R$. The main point is that the dilemma that arises from two potentially contradictory rules is resolved by learning, even in the case that the original rules themselves are programmed rather than learned.

4.6 Learning as an Approach to the Problem of Context

It has been widely observed that rules that express useful commonsense knowledge often need qualification – they hold only in certain contexts. Thus a rule $Q \Rightarrow R$ may hold if context P is true, but not necessarily otherwise. Frames in the sense of Minsky [33] can be viewed as contexts that have a rich set of associated rules. The PAC semantics of such a rule is that on the subdomain in which P holds, $Q \Rightarrow R$ is the case, at least with high probability. In our architecture the simplest view to take is that it can cope easily with a context P if it has a node for recognizing whether an input satisfies P. If there is a node in a circuit unit that recognizes P, then a subcircuit that implements $P \wedge Q \Rightarrow R$ will implement exactly what is wanted, the rule $Q \Rightarrow R$ applying in the subdomain in which P holds. The question remains as to how domain sensitive knowledge can be learned. One answer is suggested immediately by projection learning; each concept R that is learned is also learned for the projections defined by every other concept P that has been previously learned or programmed. Thus if we have nodes for P and R then a complex set of conditions Q that guarantee R in the context of P can be learned from examples, for any P and R. This would be, of course, computationally onerous unless the choice of P is restricted somehow.

4.7 Learning as an Approach to Incomplete Information and Nonmonotonic Phenomena

Systems that attempt to describe the world declaratively run into methodological problems that arise from the fact that at any time there will be captured only incomplete knowledge about the world. The difficulty is that such systems need to take a generic view of how to treat incomplete information – they need a uniform theory, such as circumscription or the closed world assumption that takes positions on how to resolve the unknown [10, 29, 31].

PAC circuit semantics offers an advantage here – it resolves the unknown separately in each case by using information learned from past experience of cases where similar features to the case in hand were similarly unknown. A motivating observation here is that gates in a circuit take values at *all* times. Consider the paradigm of nonmonotonic reasoning exemplified by the widely discussed example of the bird called Tweety [10] . The system is invited to take positions on whether Tweety flies both before and after it is revealed to it that Tweety is, in fact, a penguin. Suppose that in a brain, for example, there is a two-valued gate intended to recognize penguins. This would take value 1 say, when a penguin is identified, in the PAC sense, and 0 when what is seen is identified in the PAC sense as not being a penguin. However, this gate must also take some value in cases where conventionally one would say that the system has not determined whether the object in question is a penguin or not. Suppose that in these cases the gate takes the value 0. Then we could say that a 1 value means {penguin} and a 0 means {not penguin, undetermined}. In this sense circuits represent internally the three cases {yes, no, undetermined} even if no explicit provision has been made for the third case. This means that learning and rule evaluations in the system are carried out with a semantics in which the undetermined value of each variable is represented. This is true both when a predicate is represented by a single gate having just two possible values (whether representing {yes, undetermined}/{no} or {yes}/{no,undetermined} by the two values) and also when there is a more explicit representation, say by a pair of binary gates whose four possible values do distinguish {yes}, {no}, and {undetermined}. Hence once the system has reached stability in learning it will cope with instances of incomplete information exactly as it does with ones with complete information [52]. One could say further that in natural learning systems instances of incomplete information are the natural ones, and usually the only ones available.

Pursuing this example further, suppose that the system has a rule "bird = 1 and penguin = 0 $\Rightarrow$ flies = 1". Suppose also that the gates "bird", "penguin", and "flies" all have value 0 if the truth of the corresponding predicate is undetermined in the system. Suppose further that this rule has been found to be consistent with most *natural cases* experienced by the system in accordance with the probability distribution D that describes the external world. It will then follow that for instances in which birdhood is confirmed but penguinhood is undetermined, it will be a reasonable conclusion that flies will be true and that the corresponding gate should indeed have value 1 to indicate that. This will be a valid inference since the same was true for past cases in the PAC sense. Thus the undefined is resolved here by learning from the real world distribution D, as seen through the feature set of the system. In this example D captured the fact that in the particular source of examples to which this system was exposed, in the majority of cases where a bird was identified but nothing was said about penguinhood, the bird did indeed fly.

In artificial systems, of course, the undefined value $*$ may be treated more explicitly. For example, gates may take three values $\{0, 1, *\}$, or, closer to the spirit of our architecture, the three values may be represented by a pair of binary gates.

Defaults may be viewed from this same perspective. Ignorance of the value of a predicate is rightly interpreted as not relevant to the value of another if in natural past cases of ignorance of the former, a certain consequence was true with high probability for the latter. We regard defaults as rules where certain predicates are assumed to have the undetermined value in the precondition. Their validity arises from the fact that they had proved accurate in the past, or that they could be deduced from those that had. Further examples are given in [43, 52, 53].

4.8 Reasoning

Much effort has been put into seeing whether the various formalisms that have been suggested for reasoning in AI, at least those outside the probabilistic realm, can be formulated within predicate calculus. The general answer found to this question is in the affirmative, in the sense that many of the various alternative formalisms, including those based on graph models, can be reexpressed in terms of the predicate calculus. Some of these alternative formalisms, and the necessary transformations are described by Russell and Norvig [45].

Instead of reformulating this body of work so as to fit in with our architecture, we will be content here to claim that a substantial part of it can be supported directly without change. In particular, we described an implementation of our architecture that can support modus ponens on Horn clauses. It follows that our architecture can do reasoning in all these frameworks whenever the translation results in Horn clauses. Further, it can do this by simple circuit evaluation of the contents of an image.

In addition, our architecture has capabilities for reasoning beyond those provided by circuit evaluations alone. In particular, the circuits and the image units can be used together more proactively within the reasoning process. The evaluation of a circuit may result in various actions on the image, such as the addition of further objects, the addition of statements of further relationships, or the deletion of an object. Thus a circuit may cause the execution of a step of a more complex reasoning strategy. In evaluating a scene, a circuit may add an object representing something already in the scene but at a later time, and add a relationship that expresses the future condition of the object. In other words the circuits may be able to depict a likely scenario following on from the current one. The way the future scenario is depicted can depend critically on the current one. In other words we need circuits that are able to make useful modifications to the scene in a highly content dependent manner.

The upshot is that the circuits will contain, in addition to information about the external world as implied in previous sections, further information that concerns

the strategies that are to be used for the internal deliberations of the system. These deliberations will be seriously restricted, of course, by computational constraints, such as limits on the number of objects allowed in a scene.

4.9 System Design

It has been asked whether in our architecture one still has to design AI systems, exactly as in the old way, and all that is offered in addition are some mechanisms for the automatic adjustment of weights. While in some sense the answer to this is yes, the more important point is that the learning capability offers things that are qualitatively different. First it provides a behavioral semantics. Further it offers a unified methodology for dealing with conflict resolution, nonmonotonic phenomena, incomplete information, robustness, and the problem of context among others. Above all, it offers a viable approach for infusing knowledge into a system in bulk.

Clearly conventional design principles offer a start in designing systems in this domain also. In a programmed system one expects to construct various modules for various functionalities, and have these interface with each other. In particular, one may have a hierarchy of functions, lower level ones processing the inputs directly, and higher level ones processing the outputs of the lower level ones. Thus an acyclic graph of circuit units, in which some low level modules mimic the modules that would be used in, say, a language system, is one possible starting point.

There are also some design choices that are unique to the architecture. For creating new compound features there are various possibilities. Because of the centrality of attribute efficient learning algorithms, an obvious method is to generate large numbers of combinations in a systematic simple way. Any relevant ones so discovered will be valuable, and the irrelevant ones will do little harm. One approach is to generate for any variable set the set of conjunctions of all pairs of them. Another choice is to create conjunctions of just those pairs that occur with high correlation. More generally one can generate some set of *polynomialy generable* combinations [51]. The intention is that large numbers of variables, even if most are irrelevant, will not degrade performance in the presence of attribute efficient learning algorithms.

We are suggesting that the way to evaluate this architecture is by experimentation. In that connection we note here one aspect. Most general approaches to AI attempt to describe a uniform method for building a knowledge base starting from a blank slate. Facts and reasoning about the most universal concepts such as time and space are then formalized in the same framework as is more specialized knowledge (e.g. [25]). In the neuroidal framework there is room reserved for treating the universal concepts differently from the others. In particular, the features and inverse features of the image units can be used to implement efficiently certain universal knowledge, such as to do with low-level visual processing, and reasoning about time or space. These

functions may be complex and specialized enough that implementing them directly in the same framework as the more general knowledge would introduce unnecessary difficulties. Thus interpreting a depiction of a three dimensional scene may be done in the first instance by computational mechanisms that are preprogrammed in the image units and expert at that task, and are possibly totally unrelated to the more general knowledge manipulation capability of the circuit units. This dichotomy may be analogous to that in biological systems between knowledge available at birth and acquired during evolution, and that learned during life. However, we expect that some circuit units will be programmed and do not exclude the possibility that some image unit functions are learned.

The choice of what features to implement in the input devices and image units poses significant questions. In current machine learning applications the choice of feature set is often critical to success. This phenomenon is probably accentuated when we scale up to more general cognitive problems. In particular, the training data may have to match the chosen feature set in some way. If, as is most likely, the feature set chosen is significantly different from that used by the human nervous system, the teaching materials that will succeed can be expected to be different from those that are effective for teaching humans.

5 Validating the Architecture

What we are claiming is that several of the generic difficulties that have been encountered in designing scalable AI systems are technically solvable within the described architecture. To give convincing evidence for this claim one would need to construct a system that uses general knowledge on the scale and with the success envisioned here. A first experimental question, therefore, is how this architecture might be bootstrapped to create a system whose performance in some cognitive area is indisputably superior to existing ones.

In designing a system important choices need to be made regarding the feature detectors of the images and of the input devices. If the main source of information for the system is visual, for example, then problems with interpreting visual inputs are likely to be inherited by the system. Our view is that the building of a database that would comprise the first level of internal knowledge in the system for such circumstance will have to be pursued in a careful and purposeful manner. It would have to acknowledge and compensate for the poverty of what we might be able to program currently as feature detectors. The Golding-Roth paper [11] is a good example, however, of how one might start. They take a linguistic corpus and create complex features out of single words, grammatical parts of speech of single words, and boolean conjunctions of these for sets of words that occur in close proximity in the text. Such

features, though artificial, do capture useful information about meaning beyond what can be gleaned from examining each word in isolation. By using such a set of features they are able to learn to disambiguate certain pairs of words that humans often confuse. Although the feature set is imperfect and artificial, the system is able to create a useful level of knowledge that is a level of abstraction above the individual words.

By analogy one would hope to build a larger knowledge base in a succession of similar steps. This would be done by a managed combination of inductive training operations, and the use of programmed knowledge as contained, for example, in dictionaries. We believe that learning will need to play a large part of the overall task if we are to achieve the various kinds of robustness discussed in this paper. Purely programmed systems would appear to have an inherent limitation in this regard.

It is quite likely that if all this is to be achieved, natural language inputs will play a major role. Corpora suitably annotated with the appropriate levels of knowledge will have to be created. Substantial infrastructure will need to be manufactured to provide the teaching materials for these systems. This would compensate for any shortcoming in the range of preprogrammed features that we are able to realize. Although large amounts of reliable knowledge are already available in linguistic format, there are many obstacles to preparing or selecting useful linguistic teaching materials. Much commonsense knowledge is nowhere encoded, and much text produced by humans contains linguistic constructs that are currently difficult for machines to analyze. These facts make the preparation of the teaching materials challenging, but, we believe, not insurmountable

6 Conclusion

We have described a series of arguments that suggest that many of the problems that have been identified as obstacles to AI at a conceptual level, can be solved if one gives inductive learning a suitable central role. In this respect the proposed architecture differs from other general approaches to cognitive architectures that have been described, such as [1], [37], and [38], in which inductive learning plays a much smaller role.

We note that our use of PAC semantics suggests a modified Turing test. His basic criterion for whether a machine could think was that the performance of the machine should be indistinguishable from that of a human to an interrogator communicating via a teleprinter [47]. The significance of this informally stated criterion is that it is a purely *behavioral* one. What PAC semantics offers is a precise way of formulating such behavioral criteria. In particular it insists that both the task being learned, and the distribution of inputs over which learning is performed and performance measured, need to be identified.

The tasks in which we are interested here are those involving a large amount of knowledge that has not been systematized into an exact science. A possible area in which one might hope to test the performance of a system at such a task is that of intelligent word processing. One can imagine a word processor that not only detects misspelled words, but does a succession of more and more intelligent tasks, such as suggesting alternative words and phrasing, or noting confusions or inconsistencies, much as a teacher marking an essay might. There appears to be a continuum of tasks of increasing difficulty in this area. Even when restricted to the disambiguation tasks previously mentioned, systems could invoke more and more world knowledge in their suggestions, and their perceived performance would increase correspondingly. One can imagine experiments in which commentary to writers is provided variously by humans and machines, and the writers' task is to distinguish which one was the case. Such an instance of a modified Turing Test would therefore refer to a concrete real world distribution of cases generated, for example, by twelve year old students, and created independently of the context of the test. Each such distribution would define a different task and therefore a different test. For an empirical validation of our architecture one would need to show that systems can be built that pass such specific tests of ever higher levels of difficulty.

7 Acknowledgment

I am grateful to Roni Khardon, Dan Roth, and Rocco Servedio for their helpful comments on various versions of this paper.

References

[1] J.R. Anderson. *The Architecture of Cognition.* Harvard University Press, 1983.

[2] J.R. Anderson. *Rules of the Mind.* Erlbaum, Hillsdale, NJ, 1993.

[3] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[4] A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, 1992.

[5] A. Blum, et al. A polynomial time algorithm for learning noisy linear threshold functions. In *Proc. 37th IEEE Symp. on Theory of Computing*, pages 330–338, 1996.

[6] T. Bylander. Learning linear threshold functions in the presence of classification noise. In *Proc. 7th ACM Conference on Computational Learning Theory*, pages 340–347, 1994.

[7] N. Cesa-Bianchi, et al. How to use expert advice. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 382–39, 1993.

[8] E. Cohen. Learning noisy perceptrons by a perceptron in polynomial time. In *Proc 38th IEEE Symp. on Foundation of Computer Science*, pages 514–523, 1997.

[9] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Inf. and Computation*, 82(3):247–266, 1989.

[10] M.L. Ginsberg. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, Los Altos, CA, 1989.

[11] A.R. Golding and D. Roth. Applying Winnow to context-sensitive spelling correction. In *Proc 13th Int. Conf. on Machine Learning*, pages 182–190, San Francisco, CA, 1996. Morgan Kaufmann.

[12] J.Y. Halpern. An analysis of first-order logics of probability. In *Artificial Intelligence Journal*, volume 46, pages 311–350, 1990.

[13] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[14] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4:7–40, 1989.

[15] M. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

[16] M.J. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proc. 25th ACM Symp. on Theory of Computing*, New York, 392-401 1993. ACM Press.

[17] M.J. Kearns and R.E. Schapire. Efficient distribution-free probabilistic concepts. *J. Comput. Syst. Sci.*, 48(3):464, 1994.

[18] M.J. Kearns and U.V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[19] R. Khardon. Learning to take actions. In *Proc. National Conference on Artificial Intelligence*, pages 787–792. AAAI, 1996.

[20] R. Khardon and D. Roth. Learning to reason. In *AAAI94*, pages 682–687. Morgan Kaufmann, 1994.

[21] R. Khardon and D. Roth. Learning to reason with a restricted view. In *Proc. 8th ACM Conference on Computational Learning Theory*, pages 301–310, 1995.

[22] J. Kivinen and M.K. Warmuth. The perceptron algorithm vs. Winnow: linear vs. logarithmic mistake bounds when few input variables are relevant. In *Proc. 8th ACM Conference on Computational Learning Theory*, pages 289–296, 1995.

[23] D. Knuth. *The Art of Computer Programming*, volume 1. Addison Wesley, Reading, MA, 1973.

[24] P. Langley, D. Klahr, and R Neches. *Production System Models of Learning and Development*. MIT Press, 1987.

[25] D.B. Lenat, et al. CYC: Toward programs with common sense. In *CACM*, volume 33:8, pages 30–49, 1990.

[26] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[27] N. Littlestone. From on-line to batch learning. In *Proc. 2nd Workshop on Computational Learing Theory*, pages 269–284, 1989.

[28] J. McCarthy. Programs with commonsense. In *Proc. Teddington Conference on the Mechanization of Thought Processes*, London, 1959. HMSO.

[29] J. McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[30] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In D. Michie, editor, *Machine Intelligence*, volume 4, New York, 1969. American Elsevier.

[31] D. McDermott and J. Doyle. Nonmonotonic logic I. *Artificial Intelligence*, 13(1), 1980.

[32] G.A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

[33] M. Minsky. A framework for representing knowledge. In P.H. Winston, editor, *The Psychology of Computer Vision*, New York, 1975. McGraw Hill.

[34] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[35] S. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, pages 363–391, 1990.

[36] S. Muggleton. Inductive logic programming: derivations, successes and shortcomings. *SIGART Bulletin*, 5(1):5–11, 1994. Also other articles in same volume.

[37] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[38] A. Newell and H.A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[39] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Los Altos, CA, 1988.

[40] L. Pitt and L.G. Valiant. Computational limits on learning from examples. *J. ACM*, 35:965–984, 1988.

[41] R.L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[42] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

[43] D. Roth. Learning to reason: the non-monotonic case. *Proc. Int. Joint Conf. Art. Intl.*, pages 1178–118, 1995.

[44] D. Roth. A connectionist framework for reasoning: Reasoning with examples. In *Proc. National Conference on Artificial Intelligence*, pages 1256–1261. AAAI, 1996.

[45] S. Russell and P. Norvig. *Artificial Intelligence*. Prentice Hall, Upper Saddle River, NJ, 1995.

[46] M. Tambe, A. Newell, and P.S. Rosenbloom. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5:299–348, 1990.

[47] A.M. Turing. Computing machinery and intelligence. *Mind 59*, pages 433–460, 1950. (Reprinted in Collected Works of A.M. Turing: Mechanical Intelligence, (D.C. Ince, ed.), North-Holland, 1992).

[48] A.M. Turing. Solvable and unsolvable problems. *Science News*, 31:7–23, 1954. (Reprinted in Collected Works of A.M. Turing: Mechanical Intelligence (D.C. Ince, ed.) North-Holland 1992).

[49] A. Tversky and D. Kahnemann. Causal schemata in judgments under uncertainty. In *Progress in Social Psychology*, Erlbaum, Hillsdale, NJ, 1977.

[50] L.G. Valiant. A theory of the learnable. *Comm. of ACM*, 27(11):1134–1142, 1984.

[51] L.G. Valiant. Learning disjunctions of conjunctions. In *International Joint Conference on Artificial Intelligence*, pages 560–566, Los Angeles, CA, 1985. Morgan Kaufmann.

[52] L.G. Valiant. *Circuits of the Mind.* Oxford University Press, 1994.

[53] L.G. Valiant. Rationality. In *Proc. 8th Ann. Conference on Computational Learning Theory*, pages 3–14. ACM Press, 1995.

[54] L.G. Valiant. A neuroidal architecture for cognitive computation. Technical Report TR-11-96, Harvard University, 1996.

[55] L.G. Valiant. Projection learning. Technical Report TR-19-97, Harvard University, 1997.