Robot Baby 2001

Paul R. Cohen[1], Tim Oates[2], Niall Adams[3], Carole R. Beal[4]

[1] Department of Computer Science. University of Massachusetts, Amherst
cohen@cs.umass.edu
[2] Department of Computer Science. University of Maryland, Baltimore County
oates@cs.umbc.edu
[3] Department of Mathematics. Imperial College, London
n.adams@ic.ac.uk
[4] Department of Psychology, University of Massachusetts, Amherst
cbeal@psych.umass.edu

Abstract. In this paper we claim that meaningful representations can be learned by programs, although today they are almost always designed by skilled engineers. We discuss several kinds of meaning that representations might have, and focus on a functional notion of meaning as appropriate for programs to learn. Specifically, a representation is meaningful if it incorporates an indicator of external conditions and if the indicator relation informs action. We survey methods for inducing kinds of representations we call structural abstractions. Prototypes of sensory time series are one kind of structural abstraction, and though they are not denoting or compositional, they do support planning. Deictic representations of objects and prototype representations of words enable a program to learn the denotational meanings of words. Finally, we discuss two algorithms designed to find the macroscopic structure of episodes in a domain-independent way.

1 Introduction

In artificial intelligence and other cognitive sciences it is taken for granted that mental states are representational. Researchers differ on whether representations must be *symbolic*, but most agree that mental states have *content* — they are *about* something and they *mean* something — irrespective of their form. Researchers differ too on whether the meanings of mental states have any causal relationship to how and what we think, but most agree that these meanings are (mostly) known to us as we think. Of formal representations in computer programs, however, we would say something different: Generally, the meanings of representations have no influence on the operations performed on them (e.g., a program concludes q because it knows $p \rightarrow q$ and p, irrespective of what p and q are about); yet the representations *have* meanings, known to us, the designers and end-users of the programs, and the representations are provided to the programs *because* of what they mean (e.g., if it was not relevant that the patient has a fever, then the proposition `febrile(patient)` would not be provided to the program — programs are designed to operate in domains where meaning matters.). Thus, irrespective of whether the contents of mental states have any causal influence on what and how *we* think, these contents clearly are intended (by us) to influence what and how our programs think. The meanings of representations are not irrelevant but we have to provide them.

If programs could learn the meanings of representations it would save us a great deal of effort. Most of the intellectual work in AI is done not by programs but by their creators, and virtually *all* the work involved in specifying the meanings of representations is done by people, not programs (but see, e.g., [27, 22, 14]). This paper discusses kinds of meaning that programs might learn and gives examples of such programs.

How do people and computers come to have contentful, i.e., meaningful, mental states? As Dennett [10] points out, there are only three serious answers to the question: Contents are learned, told, or innate. Lines cannot be drawn sharply between these, in either human or artificial intelligence. Culture, including our educational systems, blurs the distinction between learning and being told; and it is impossible methodologically to be sure that the meanings of mental states are innate, especially as some learning occurs in utero [9] and many studies of infant knowledge happen weeks or months after birth.

One might think the distinctions between learning, being told, and innate knowledge are clearer in artificial systems, but the role of engineers is rarely acknowledged [8, 30, 12]. Most AI systems manipulate representations that mean what engineers intend them to mean; the meanings of representations are exogenous to the systems. It is less clear where the meanings of *learned* representations reside, in the minds of engineers or the "minds of the machines" that run the learning algorithms. We would not say that a linear regression algorithm knows the meanings of data or of induced regression lines. Meanings are assigned by data analysts or their client domain experts. Moreover, these people select data for the algorithms with some prior idea of what they mean. Most work in machine learning, KDD, and AI and statistics are essentially data analysis, with humans, not machines, assigning meanings to regularities found in the data.

We have nothing against data analysis, indeed we think that learning the meanings of representations *is* data analysis, in particular, analysis of sensory and perceptual time series. Our goal, though, is to have the machine do *all* of it: select data, process it, and interpret the results; then iterate to resolve ambiguities, test new hypotheses, refine estimates, and so on [5]. The relationship between domain experts, statistical consultants, and statistical algorithms is essentially identical to the relationship between domain experts, AI researchers, and their programs: In both cases the intermediary translates meaningful domain concepts into representations that programs manipulate, and translates the results back to the domain experts. We want to do away with the domain expert and the engineers/statistical consultants, and have programs learn represention and their meanings, autonomously.

[5] An early effort in our laboratory to automate applied statistics was Rob St. Amant's dissertation [26, 25]; while it succeeded in many respects, it had only weak notions of the meaning of representations, so its automated data analysis had a formal, syntactic feel (e.g., exploring high leverage points in regression analysis).

One impediment to learning the meanings of representations is the fuzziness of commonsense notions of meaning. Suppose a regression algorithm induces a strong relationship between two random variables x and y and represents it in the conventional way: $y = 1.31x - .03$, $R^2 = .86$, $F = 108.3$, $p < .0001$. One meaning of this representation is provided by classical inferential statistics: x and y appear linearly related and the relationship between these random variables is very unlikely to be accidental. Note that this meaning is not accessible in any sense to the regression algorithm, though it could be known, as it is conventional and unambiguous. [6] Now, the statistician might know that x is daily temperature and y is ice-cream sales, and so he or his client domain expert might assign additional meaning to the representation, above. For instance, the statistician might warn the domain expert that the assumptions of linear regression are not well-satisfied by the data. Ignoring these and other cautions, the domain expert might even interpret the representation in causal terms (i.e., hot weather causes people to buy ice-cream). Should he submit the result to an academic journal, the reviews would probably criticize this semantic liberty and would in any case declare the result as meaningless in the sense of being utterly unimportant and unsurprising.

This little example illustrates at least five kinds of meaning for the representation $y = 1.31x - .03$, $R^2 = .86$, $F = 108.3$, $p < .0001$. There is the *formal* meaning, including the mathematical fact that 86 % of the variance in the random variable y is explained by x. Note that this meaning has nothing to do with the denotations of y and x, and it might be utter nonsense in the domain of weather and ice-cream, but, of course, the formal meaning of the representation is not about weather and ice cream, it is about random variables. Another kind of meaning has to do with the *model* that makes y and x denote ice cream and weather. When the statistician warns that the residuals of the regression have structure, he is saying that a linear model might not summarize the relationship between x and y as well as another kind of model. He makes no reference to the denotations of x and y, but he might, as when he warns that ice cream sales are not normally distributed. In both cases, the statistician is questioning whether the domain (ice cream sales and weather) is faithfully represented by the model (a linear function between random variables). He is questioning whether the "essential features" of the domain are represented, and whether they are somehow distorted by the regression procedure.

The domain expert will introduce a third kind of meaning: he will interpret $y = 1.31x - .03$, $R^2 = .86$, $F = 108.3$, $p < .0001$ as a statement about ice cream sales. This is not to say that every aspect of the representation has an interpretation in the domain—the expert might not assign a meaning to the coefficient $-.03$—only that, to the expert, the representation is not a formal object but a statement about his domain. We could call this kind of meaning the *domain* semantics, or the *functional* semantics, to emphasize that the interpretation of a

[6] Indeed, there is a sense in which stepwise regression algorithms know what F statistics mean, as the values of these statistics directly affect the behavior of the algorithms.

representation has some effect on what the domain expert *does* or thinks about.

Having found a relationship between ice cream sales and the weather, the expert will feel elated, ambitious, or greedy, and this is a fourth, *affective* kind of meaning. Let us suppose, however, that the relationship is not real, it is entirely spurious (an artifact of a poor sampling procedure, say) and is contradicted by solid results in the literature. In this case the representation is meaningless in the sense that it does not *inform* anyone about how the world really works.

To which of these notions of meaning should a program that learns meanings be held responsible? The semantics of classical statistics and regression analysis in particular are sophisticated, and many humans perform adequate analyses without really understanding either. More to the point, what good is an agent that learns *formal* semantics in lieu of *domain* or *functional* semantics? The relationship between x and y can be learned (even without a statistician specifying the form of the relationship), but so long as it is a *formal* relationship between random variables, and the denotations of x and y are unknown to the learner, a more knowledgeable agent will be required to translate the formal relationship into a domain or functional one. The denotations of x and y might be learned, though generally one needs some knowledge to bootstrap the process; for example, when we say, "x denotes daily temperature," we call up considerable amounts of common-sense knowledge to assign this statement meaning. [7] As to affective meanings, we believe artificial agents will benefit from them, but we do not know how to provide them.

This leaves two notions of meaning, one based in the functional roles of representations, the other related to the informativeness of representations. The philosopher Fred Dretske wrestled these notions of meaning into a theory of how meaning can have causal effects on behavior [11, 12]. Dretske's criteria for a state being a meaningful representational state are: the state must *indicate* some condition, have the *function* of indicating that condition, and have this function assigned as the result of a *learning* process. The latter condition is contentious [10, 8], but it will not concern us here as this paper is about learning meaningful representations. The other conditions say that a reliable indicator relationship must exist and be exploited by an agent for some purpose. Thus, the relationship between mean daily temperature (the indicator) and ice-cream sales (the indicated) is apt to be meaningful to ice-cream companies, just as the relationship between sonar readings and imminent collisions is meaningful to mobile robots, because in each case an agent can do something with the relationship. Learning meaningful representations, then, is tantamount to learning reliable relationships between denoting tokens (e.g., random variables) and learning what to do when the tokens take on particular values.

The minimum required of a representation by Dretske's theory is an indicator relationship $\mathbf{s} \leftarrow I(\mathbf{S})$ between the external world state $\mathbf{S}$ and an internal state

[7] Projects such as Cyc emphasize the denotational meanings of representations [17, 16]. Terms in Cyc are associated with axioms that say what the terms mean. It took a collosal effort to get enough terms and axioms into Cyc to support the easy acquisition of new terms and axioms.

$\mathbf{s}$, and a function that exploits the indicator relationship through some kind of action a, presumably changing the world state: $f(\mathbf{s}, a) \rightarrow \mathbf{S}$. The problems are to learn representations $\mathbf{s} \sim \mathbf{S}$ and the functions f (the relationship $\sim$ is discussed below, but here means "abstraction").

These are familiar problems to researchers in the reinforcement learning community, and we think reinforcement learning is a way to learn meaningful representations (with the reservations we discuss in [30]). We want to up the ante, however, in two ways. First, the world is a dynamic place and we think it is necessary and advantageous for $\mathbf{s}$ to represent how the world changes. Indeed, most of our work is concerned with learning representations of dynamics.

Second, a *policy* of the form $f(\mathbf{s}, a) \rightarrow \mathbf{s}$ manifests an intimate relationship between representations $\mathbf{s}$ and the actions a conditioned on them: $\mathbf{s}$ contains the "right" information to condition a. The right information is almost always an abstraction of raw state information; indeed, two kinds of abstraction are immediately apparent. Not all state information is causally relevant to action, so one kind of abstraction involves selecting information in $\mathbf{S}$ to include in $\mathbf{s}$ (e.g., subsets or weighted combinations or projections of the information in $\mathbf{S}$). The other kind of abstraction involves the *structure* of states. Consider the sequence AABACAABACAABACAABADAABAC. Its structure can be described many ways, perhaps most simply by saying, "the sequence AABAx repeats five times, and $x=$C in all but the fourth replication, when $x=$D." This might be the abstraction an agent needs to act; for example, it might condition action on the distinction between AABAC and AABAD, in which case the "right" representation of the sequence above is something like this $p_1 s_1 p_1 s_1 p_1 s_1 p_1 s_2 p_1 s_1$, where p and s denote *structural* features of the original sequence, such as "prefix" and "suffix'. We call representations that include such structural features *structural abstractions*.

To recap, representations $\mathbf{s}$ are meaningful if they are related to action by a function $f(\mathbf{s}, a) \rightarrow \mathbf{S}$, but f can be stated more or less simply depending on the abstraction $\mathbf{s} \sim \mathbf{S}$. One kind of abstraction involves selecting from the information in $\mathbf{S}$, the other is structural abstraction. The remainder of this paper is concerned with learning structural abstractions.[8] Note that, in Dretske's terms, structural abstractions can be indicator functions but not all indicator functions are structural abstractions. Because the world is dynamic, we are particularly concerned with learning structural abstractions of time series.

2 Approach

In the spirit of dividing problems to conquer them we can view the problem of learning meaningful representations as having two parts:

1. Learn representations $\mathbf{s} \sim \mathbf{S}$

[8] Readers familiar with Artificial Intelligence will recognize the problem of learning structural abstractions as what we call "getting the representation right," a creative process that we reserve unto ourselves and to which, if we are honest, we must attribute most of the performance of our programs.

2. Learn functions $f(\mathbf{s}, a) \rightarrow \mathbf{S}$

This strategy is almost certainly wrong because, as we said, $\mathbf{s}$ is supposed to inform actions a, so should be learned while trying to act. Researchers do not always follow the best strategy (we don't, anyway) and the divide and conquer strategy we did follow has produced an unexpected dividend: We now think it is possible that some kinds of structural abstraction are *generally* useful, that is, they inform large classes of actions. (Had we developed algorithms to learn $\mathbf{s}$ to inform particular actions a we might have missed this possibility.) Also, some kinds of actions, particularly those involving deictic acts like pointing or saying the name of a referent, provide few constraints on representations.

Our approach, then, is to learn structural abstractions first and functions relating representations and actions to future representations, second.

3 Structural Abstractions of Time Series and Sequences

As a robot wanders around its environment, it generates a sequence of values of state variables. At each instant t we get a vector of values $\mathbf{x_t}$ (our robot samples its sensors at 10Hz, so we get ten such vectors each second). Suppose we have a long sequence of such vectors $X = \mathbf{x_0}, \mathbf{x_1}, \ldots$. Within X are subsequences x_{ij} that, when subjected to processes of structural abstraction, give rise to *episode structures* that are meaningful in the sense of informing action [9]. The trick is to find the subsequences x_{ij} and design the abstraction processes that produce episode structures. We have developed numerous methods of this sort and survey them briefly, here.

3.1 Structural Abstraction for Continuous Multivariate Series

State variables such as translational velocity and sonar readings take continuous values, in contrast to categorical variables such as "object present in the visual field." Figure 1 shows four seconds of data from a Pioneer 1 robot as it moves past an object. Prior to moving, the robot establishes a coordinate frame with an x axis perpendicular to its heading and a y axis parallel to its heading. As it begins to move, the robot measures its location in this coordinate frame. Note that the ROBOT-X line is almost constant. This means that the robot did not change its heading as it moved. In contrast, the ROBOT-Y line increases, indicating that the robot does increase its distance along a line parallel to its original heading. Note especially the VIS-A-X and VIS-A-Y lines, which represent the horizontal and vertical locations, respectively, of the centroid of a patch of light on the robots "retina," a CCD camera. VIS-A-X decreases, meaning that the object drifts to the left on the retina, while VIS-A-Y increases, meaning the object moves toward the top of the retina. Simultaneously, both series jump to

[9] Ideally, structural abstraction should be an on-line process that influences action continuously. In practice, most of our algorithms gather data in batches, form abstractions, and use these to inform action in later episodes.

constant values. These values are returned by the vision system when nothing is in the field of view.

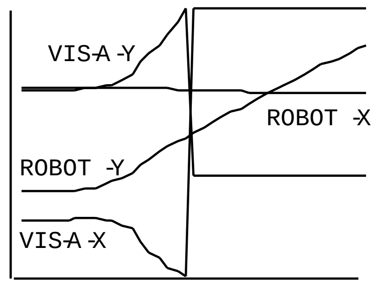


Fig. 1. As the robot moves, an object approaches the periphery of its field of view then passes out of sight.

Every time series that corresponds to moving past an object has qualitatively the same structure as the one in Figure 1. It follows that if we had a statistical technique to group the robots experiences by the characteristic patterns in multivariate time series (where the variables represent sensor readings), then this technique would in effect learn a taxonomy of the robots experiences. *Clustering by dynamics* [28] is such a technique:

1. A long multivariate time series is divided into segments, each of which represents an episode such as moving toward an object, avoiding an object, crashing into an object, and so on. (Humans divide the series into episodes by hand; more on this in section 5.) The episodes are not labeled in any way.
2. A dynamic time warping algorithm compares every pair of episodes and returns a number that represents the degree of similarity between the time series in the pair. Dynamic time warping is a technique for "morphing" one multivariate time series into another by stretching and compressing the horizontal (temporal) axis of one series relative to the other [13]. The algorithm returns a degree of mismatch (conversely, similarity) between the series after the best fit between them has been found.
3. Given similarity numbers for every pair of episodes, it is straightforward to cluster episodes by their similarity.
4. Another algorithm finds the "central member" of each cluster, which we call the *cluster prototype* following Rosch [24].

Clustering by dynamics produces structural abstractions (prototypes) of time series, the question is whether these abstractions can be meaningful in the sense of informing action. In his PhD dissertation, Matt Schmill shows how to use prototypes as planning operators. The first step is to learn rules of the form, "in state i, action a leads to state j with probability p." These rules are learned by a classical decision-tree induction algorithm, where features of states are decision variables. Given such rules, the robot can plan by means-ends analysis. It plans not to achieve world states specified by exogenous engineers, as in conventional generative planning, but to achieve world states which are preconditions for its actions. Schmill calls this "planning to act," and it has the effect of gradually increasing the size of the corpus of prototypes and things the robot can do. The neat thing about this approach is that every action produces new data, i.e., revises the set of prototypes. It follows that the robot should plan more frequently and more reliably as it gains more experiences, and in recent experiments, Schmill demonstrates this. Thus, Schmill's work shows that clustering by dynamics yields structural abstractions of time series that are meaningful in the sense of informing action.

There is also a strong possibility that prototypes of this kind are meaningful in the sense of informing communicative actions. Oates, Schmill and Cohen [29] report a very high degree of concordance between the clusters of episodes generated by the dynamic time warping method, above, and clusters generated by a human judge. The prototypes produced by dynamic time warping are not weird and unfamiliar to people, but seem to correspond to how humans themselves categorize episodes. Were this not the case, communication would be hard, because the robot would have an ontology of episodes unfamiliar to people. Oates, in particular, has been concerned with communication and language, and has developed several methods for learning structural abstractions of time series that correspond to words in speech and denotations of words in time series of other sensors, as we shall see, shortly.

3.2 Structural Abstraction for Categorical Sequences

Ramoni, Sebastiani and we have developed Bayesian algorithms for clustering activities by their dynamics [18]. In this work, dynamics are captured in first-order markov chains, and so the method is best-suited to clustering sequences of discrete symbols. The Bayesian Clustering by Dynamics (BCD) algorithm is easily sketched: Given time series of tokens that represent states, construct a transition probability table (i.e., a markov chain model) for each series, then measure the similarity between each pair of tables using the Kullback-Liebler (KL) distance, and finally group similar tables into clusters. The BCD algorithm is *agglomerative*, which means that initially, there is one cluster for each markov chain, then markov chains are merged, iteratively, until a stopping criterion is met. Merging two markov chains yields another markov chain. The stopping criterion in BCD is that the posterior probability of the clustering is maximum. Said in another way, BCD solves a Bayesian model selection problem where the model it seeks is the most probable *partition* of the original markov chains given the data and the priors (a partition is a division of a set into mutually exclusive and exhaustive subsets). As the space of partitions is exponential, BCD uses the KL distance as a heuristic to propose which markov chains to merge, but only merges them if doing so improves the marginal likelihood of the resulting partition. We have developed versions of BCD for series of a single state variable and for series of vectors of state variables [18, 23].

The clusters found by BCD have never been used by the robot to inform its actions, as in Schmill's experiments, so we cannot say they are meaningful to the robot. It is worth mentioning that they have high concordance with the clustering produced by dynamic time warping and with human clustering, when applied to the series in the Oates et al. experiment, cited above.

3.3 A Critique of Sensory Prototypes

Clustering by dynamics, whether by dynamic time warping or Bayesian model selection, takes a set of time series or sequences and returns a partition of the set. Prototypes, or "average members," may then be extracted from the resulting subsets. While the general idea of clustering by dynamics is attractive (it does produce meaningful structural abstractions), the methods described above have two limitations. First, they require someone (or some algorithm) to divide a time series into shorter series that contain instances of the structures we want to find. For example, if we want to find classes of interactions with objects, we must provide a set of series each of which contains one interaction with objects. Neither technique can accept time series of numerous undifferentiated activities (e.g., produced by a robot roaming the lab for an hour).

A more serious problem concerns the kinds of structural abstraction produced by the methods. The dynamic time warping method produces "average episodes," of which Figure 1 is an example, and BCD produces "average markov chains," which are just probability transition matrices. Suppose we examine an instance of each representation that corresponds to the robot rolling past a cup. Can we find anything in either representation that denotes the cup? We cannot. Consequently, these representations cannot inform actions that depend on individuating the cup; for example, the robot could not respond correctly to the directive, "Turn toward the cup." The abstractions produced by the algorithms contain sufficient structure to cluster the episodes, but still lack much of the structure of the episodes. This is particularly true of the markov chain models, in which the series is chopped up into one-step transitions and all global information about the "shape" of the series is lost, but even the representation in Fig. 1 does not individuate the objects in an episode.

If one is comfortable with a crude distinction between sensations and concepts, then the structural abstractions produced by the methods described above are entirely sensory [21]. They are abstractions of the dynamics of sensor values—of how an episode "feels"—they do not represent concepts such as objects, actors, actions, spatial relationships, and the like. Fig. 1 represents the *sensations* of moving past an object, so it is meaningful if the robot conditions actions on its sensations (as it does in Matt Schmill's work) but it is not a representation of an object, the act of moving, the distance to the object, or any other individuated entity.

Nor for that matter do these abstractions make explicit other structural features of episodes, such as the boundaries between sub-episodes or cycles among states.

Oates has developed methods for learning structural abstractions of time series that individuate words in speech and objects in a scene. Oates' methods are described in the following section. We also have implemented algorithms for finding the boundaries in episodes and the hierarchical structure of episodes; these are described in section 5.

4 Learning Word Meanings

Learning the meanings of words in speech clearly requires individuation of elements in episodes. Suppose we wanted to talk to the robot about cups: We would say, "there's a cup" when we see it looking at a cup; or, "a cup is on your left," when a cup is outside its field of view; or, "point to the cup," when there are several objects in view, and so on. To learn the meaning of the word "cup" the robot must first individuate the word in the speech signal, then individuate the object "cup" in other sensory series, associate the representations; and perhaps estimate some properties of the object corresponding to the cup, such as its color, or the fact that it participates as a target in a "turn toward" activity. In his PhD dissertation, Oates discusses an algorithm called PERUSE that does all these things [20].

To individuate objects in time series Oates relies on *deictic markers* — functions that map from raw sensory data to representations of objects [4, 3, 1]. A simple deictic marker might construct a representation whenever the area of colored region of the visual field exceeds a threshold. The representation might include attributes such as the color, shape, and area, of the object, as well as the intervals during which it is in view, and so on.

To individuate words in speech, Oates requires a corpus of speech in which words occur multiple times (e.g., multiple sentences contain the word "cup"). Spoken words produce similar (but certainly not identical) patterns in the speech signal, as one can see in Figure 2. (In fact, Oates' representation of the speech signal is multivariate but the univariate series in Fig. 2 will serve to describe his approach.) If one knew that a segment in Figure 2 corresponded to a word, then one could find other segments like it, and construct a prototype or average representation of these. For instance, if one knew that the segment labeled A in Figure 2 corresponds to a word, then one could search for similar segments in the other sentences, find A', and construct a prototype from them. These problems are by no means trivial, as the boundaries of words are not helpfully marked in speech. Oates treats the boundaries as hidden variables and invokes the Expectation Maximization algorithm to learn a model of each word that optimizes the placement of the boundaries. However, it is still necessary to begin with a segment that probably corresponds to a word. To solve this problem, Oates relies on versions of the boundary entropy heuristic and frequency heuristics, discussed below. In brief, the entropy of the distribution of the "next tick" spikes at episode (e.g., word) boundaries; and the patterns in windows that contain boundaries tend to be less frequent than patterns in windows that do not.

These heuristics, combined with some methods for growing hypothesized word segments, suffice to bootstrap the process of individuating words in speech.

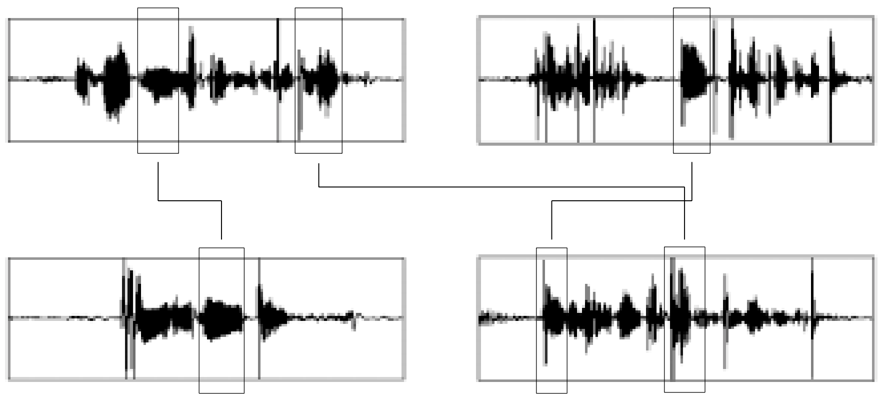


Fig. 2. Corresponding words in four sentences. Word boundaries are shown as boxes around segments of the speech signal. Segments that correspond to the same word are linked by connecting lines.

Given prototypical representations of words in speech, and representations of objects and relations, Oates' algorithm learns associatively the *denotations* of the words. Denotation is a common notion of meaning: The meaning of a symbol is what it points to, refers to, selects from a set, etc. However, naive implementations of denotation run into numerous difficulties, especially when one tries to learn denotations. One difficulty is that the denotations of many (perhaps most) words cannot be specified as boolean combinations of properties (this is sometimes called the problem of necessary and sufficient conditions). Consider the word "cup". With repeated exposure, one might learn that the word denotes prismatic objects less than five inches tall. This is wrong because it is a bad description of cups, and it is more seriously wrong because no such description of cups can reliably divide the world into cups and non-cups (see, e.g., [15, 5]).

Another difficulty with naive notions of denotation is referential ambiguity. Does the word "cup" refer to an object, the shape of the object, its color, the actions one performs on it, the spatial relationship between it and another object, or some other feature of the episode in which the word is uttered? How can an algorithm learn the denotation of a word when so many denotations are logically possible?

Let us illustrate Oates' approach to these problems with the word "square," which has a relatively easy denotation. Suppose one's representation of an object includes its apparent height and width, and the ratio of these. An object will appear square if the ratio is near 1.0. Said differently, the word "square" is more likely to be uttered when the ratio is around 1.0 than otherwise. Let ϕ be the group of sensors that measures height, width and their ratio, and let x be the value of the ratio. Let U be an utterance and W be a word in the utterance. Oates defines the denotation of W as follows:

$$denote(W, \phi, x) = Pr(contains(U, W)|about(U, \phi), x) \tag{1}$$

The denotation of the word "square" is the probability that it is uttered given that the utterance is about the ratio of height to width and the value of the ratio. More plainly, when we say "square" we are talking about the ratio of height to width and we are more likely to use the word when the value of the ratio is close to 1.0. This formulation of denotation effectively dismisses the problem of necessary and sufficient conditions, and it brings the problem of referential ambiguity into sharp focus, for when an algorithm tries to *learn* denotations it does not have access to the quantities on the right hand side of Eq. 1, it has access only to the words it hears:

$$hear(W, \phi, x) = Pr(contains(U, W)|x) \tag{2}$$

The problem (for which Oates provides an algorithm) is to get $denote(W, \phi, x)$ from $hear(W, \phi, x)$.

At this juncture, however, we have said enough to make the case that word meanings can be learned from time series of sensor and speech data. We claim that Oates' PERUSE algorithm constructs representations and learns their meanings by itself. Although the deictic representations of objects are not learned, the representations of words *are* learned and so are the associations between features of the deictic representations and words. PERUSE learns "above" and learns to associate the word with a spatial relationship. At no point does an engineer implant representations in the system and provide them with interpretations. Although PERUSE is a suite of statistical methods, it is about as far from the data analysis paradigm with which we began this paper as one can imagine. In that example, an analyst and his client domain expert select and provide data to a linear regression algorithm because it means something to them, and the algorithm computes a regression model that (presumably) means something to them. Neither data nor model mean anything to the algorithm. In contrast, PERUSE selects and processes speech data in such a way that the resulting prototypes are likely to be individuated entities (more on this, below), and it assigns meaning to these entities by finding their denotations as described earlier. Structural abstraction of representations and assignment of meaning are all done by PERUSE.

The algorithm clearly learns meaningful representations, but are they meaningful in the sense of informing action? As it happens, PERUSE builds word representations sufficient for a robot to respond to spoken commands and to translate words between English, German and Mandarin Chinese. The denotational meanings of the representations are therefore sufficient to inform some communicative acts.

5 Elucidating Episode Structure

Earlier we described our problem as learning structural abstractions of state information, particularly abstractions of the dynamics of state variables, which inform action. Section 3 introduced clustering by dynamics and sensory abstractions which did not individuate objects or the structure of a robot's activities. The previous section showed how to individuate words and objects and learn denotations. This section is concerned with the structure of activities.

Time series data are sampled at some frequency (10 Hz for our robots) but the robot changes what it is doing more slowly. Of course, one can describe what a robot is doing at any time scale (up to 10 Hz), but some descriptions are better than others in ways we will describe shortly. First, an example. Figure 3 shows three variables from a multivariate time series, each running for 1500 ticks, or 150 seconds. These series together tell a story: the robot approaches an object and starts to orbit it; at two points (around tick 500 and again around tick 800) the object disappears from view, and the robot's movement pattern changes in response. At this relatively macroscopic scale, then, the robot changes its activity half a dozen times, although, as one can see, its sensor values change with much higher frequency.

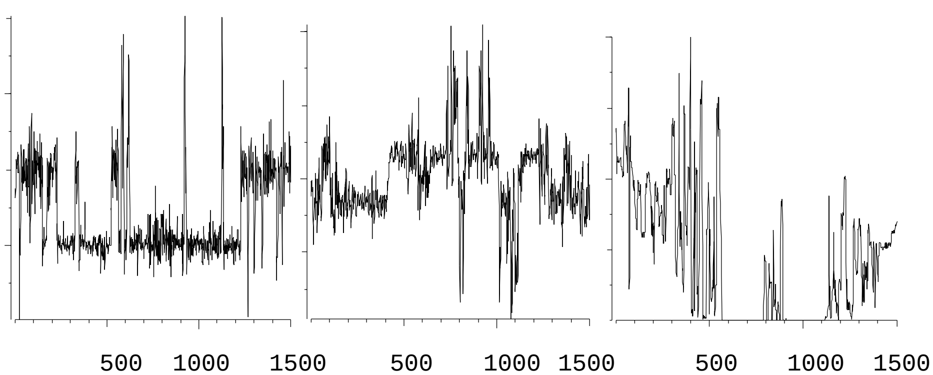


Fig. 3. Time series of translational velocity, rotational velocity, and area of a region in the visual field.

By *episodic structure* we mean relatively macroscopic patterns in time series that are meaningful in the sense of informing action. If Oates showed how to individuate things we might denote with nouns, prepositions, and adjectives, episodic structure individuates things we might denote with verbs.

As noted earlier, the right way to proceed might be to couple the problem of learning episodic structure with the problem of learning how to act, but we have approached the problems as if they were independent. This means we need a way to assess whether a segment of a time series is apt to be meaningful—capable of informing action—which is independent of the actions an agent might perform. Said differently, we are looking for a particular kind of marker in time series, one that says, "If you divide the series at these markers, then there is a good chance that the segments between the markers will be meaningful in the sense of informing action."

We have identified four kinds of markers of episode structures.

Coincidences Random coincidences are rare, so coincidences often mark the boundaries of episode structures. Figure 4 shows the same time series as in Figure 3, though the series have been smoothed and shifted vertically away from each other on the vertical axis. Vertical lines have been added at some of the points where two or more of the series change slope sharply. Now, if the series were unrelated, then such an inflection in one would be very unlikely to coincide with an inflection in another, for inflections are rare. If rare events in two or more series coincide, then the series are probably not unrelated. Moreover, the points of coincidence are good markers of episode structures, as they are points at which something causes changes in the series to coincide.

Boundary entropy Every unique subsequence in a series is characterized by the distribution of subsequences that follow it; for example, the subsequence "en" in this sentence repeats five times and is followed by tokens c, ", t and s. This distribution has an entropy value. In general, every subsequence of length has a *boundary entropy*, which is the entropy of the distribution of subsequences that follow it. If a subsequence S is an episode, then the boundary entropies of subsequences of S will have an interesting profile: They will start relatively high, then sometimes drop, then peak at the last element of S. The reasons for this are, first, that the predictability of elements within an episode increases as the episode extends over time; and, second, that the element that immediately follows an episode is relatively uncertain. Said differently, within episodes, we know roughly what will happen, but at episode boundaries we become uncertain.

Frequency Episode structures are meaningful if they inform action. Rare structures might be informative in the information theoretic sense, have few opportunities to inform action because they arise infrequently. Consequently, all human and animal learning places a premium on frequency (and, by the way, learning curves have their characteristic shape). In general, episode structures are common structures. However, not all common structures are episode structures. Very often, the most frequent structures in a domain are the smallest or shortest, while the meaningful structures—those that inform action—are longer. A useful example of this phenomenon comes from word morphology. The following subsequences are the 100 most frequently-occurring patterns in the first 10,000 characters of Orwell's book *1984*, but many are not morphemes, that is, mean-

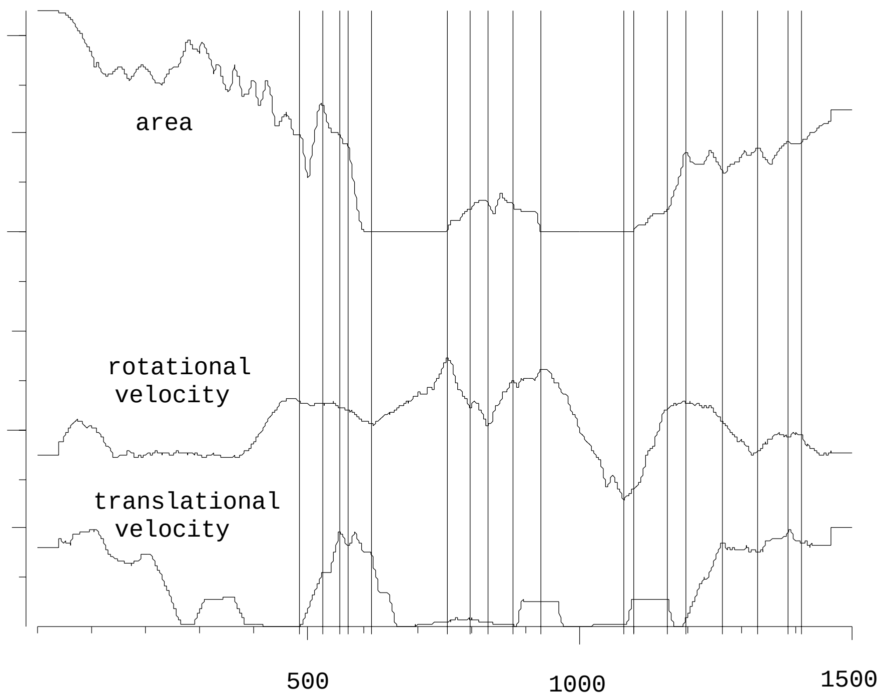


Fig. 4. The series from Figure 3, smoothed and shifted apart on the vertical axis. Vertical lines show points at which two or more of the series experience a significant change in slope.

ingful units:

> th in the re an en as ed to ou it er of at ing was or st on ar and es ic el al om ad ac is wh le ow ld ly ere he wi ab im ver be for had ent itwas with ir win gh po se id ch ot ton ap str his ro li all et fr andthe ould min il ay un ut ur ve whic dow which si pl am ul res that were ethe wins not winston sh oo up ack ter ough from ce ag pos bl by tel ain

Even so, frequency provides a good marker for the boundaries of episode structures. Suppose that the subsequences wx and yz are both both very common and subsequence xy is rare; where would you place a boundary in the sequence $wxyz$?

Changes in Probability Distributions Sequences can be viewed as the outputs of finite state machines, and many researchers are interested in inducing the machines that generate sequences. Our interest is slightly different; we want to know the boundaries of sequences. Another way to say this is we want to place boundaries in such a way that the probability distributions to the left and right of the boundaries are different.

5.1 Algorithms

This section presents two algorithms for learning episode structures. The first is based on the coincidence heuristic, above; the second relies on boundary entropy and frequency. These algorithms are described in detail in [6, 7], respectively, and some of the material in the following sections is excerpted from these papers. We are working on an online version of BCD that implements the "change in probability distribution" heuristic but the work is preliminary.

Fluents and Temporal Relationships The fluent learning algorithm induces episode structures from time series of binary vectors. A binary vector $\mathbf{b}_t$ is a simple representation of a set of logical propositions at time t: $\mathbf{b}[i]_t = 1$ means proposition p_i is true. If a proposition is true for all the discrete times in the range m, n (i.e., $\mathbf{b}[i]_{m,n} = 1$) then the proposition is called a *base fluent.* (States with persistence are called fluents by McCarthy [19].) In an experiment with a Pioneer 1 mobile robot, we collected a dataset of 22535 binary vectors of length 9. Sensor readings such as translational and rotational velocity, the output of a "blob vision" system, sonar values, and the states of gripper and bump sensors, were inputs to a simple perceptual system that produced the following nine propositions: STOP, ROTATE-RIGHT, ROTATE-LEFT, MOVE-FORWARD, NEAR-OBJECT, PUSH, TOUCH, MOVE-BACKWARD, STALL.

Allen [2] gave a logic for relationships between the beginnings and ends of fluents. We use a nearly identical set of relationships:

- SBEB X starts before Y, ends before Y; Allen's "overlap"
- SWEB Y starts with X, ends before X; Allen's "starts"
- SAEW Y starts after X, ends with X; Allen's "finishes"
- SAEB Y starts after X, ends before X; Allen's "during"
- SWEW Y starts with X, ends with X; Allen's "equal"
- SE Y starts after X ends; amalgamating Allen's "meets" and "before"

In Allen's calculus, "meets" means the end of X coincides exactly with the beginning of Y, while "before" means the former event precedes the latter by some interval. In our work, the truth of a predicate such as SE or SBEB depends on whether start and end events happen within a window of brief duration. Said differently, "starts with" means "starts within a few ticks of" and "starts before" means "starts more than a few ticks before" The reason for this window is that on a robot, it takes time for events to show up in sensor data and be processed perceptually into propositions, so coinciding events will not necessarily produce propositional representations at exactly the same time.

Let $\rho \in$[SBEB,SWEB,SAEW,SAEB,SWEW,SE], and let f be a proposition (e.g., MOVING-FORWARD). Composite fluents have the form:

$$F \leftarrow f \mid \rho(f, f)$$

$$CF \leftarrow \rho(F, F)$$

That is, a fluent F may be a proposition or a temporal relationship between propositions, and a composite fluent is a temporal relationship between fluents. A situation has many alternative fluent representations, we want a method for choosing some over others. The method will be statistical: We will only accept $\rho(F, F)$ as a representation if the constituent fluents are statistically associated, if they "go together."

Consider a composite fluent like SBEB(brake,clutch): When I approach a stop light in my standard transmission car, I start to brake, then depress the clutch to stop the car stalling; later I release the brake to start accelerating, and then I release the clutch. To see whether this fluent— SBEB(brake,clutch)—is statistically significant, we need two contingency tables, one for the relationship "start braking then start to depress the clutch" and one for "end braking and then end depressing the clutch":

	s(x=clutch)	*s(x!=clutch)*
s(x=brake)	*a1*	*b1*
s(x!=brake)	*c1*	*d1*

	e(x=clutch)	*e(x!=clutch)*
e(x=brake)	*a2*	*b2*
e(x!=brake)	*c2*	*d2*

Imagine some representative numbers in these tables: Only rarely do I start something other than braking and then depress the clutch, so $c1$ is small. Only rarely do I start braking and then start something other than depressing the clutch (otherwise the car would stall), so $b1$ is also small. Clearly, $a1$ is relatively large, and $d1$ bigger, still, so the first table has most of its frequencies on a diagonal, and will produce a significant χ^2 statistic. Similar arguments hold for the second table. When both tables are significant, we say SBEB(brake,clutch) is a significant composite fluent.

Fluent learning algorithm The fluent learning algorithm incrementally processes a time series of binary vectors. At each tick, a bit in the vector $\mathbf{b_t}$ is in one of four states:

Still off: $b_{t-1} = 0 \wedge b_t = 0$
Still on: $b_{t-1} = 1 \wedge b_t = 1$
Just off: $b_{t-1} = 1 \wedge b_t = 0$
Just on: $b_{t-1} = 0 \wedge b_t = 1$

The fourth case is called *opening*; the third case *closing*. It is easy to test when base fluents (those corresponding to propositions) open and close, slightly more complicated for composite fluents such as SBEB(f_1,f_2), because of the ambiguity about which fluent opened. Suppose we see open(f_1) and then open(f_2). It's unclear whether we have just observed open(SBEB(f_1,f_2)), open(SAEB(f_1,f_2)), or open(SAEW(f_1,f_2)). Only when we see whether f_2 closes after, before, or with

f_1 will we know which of the three composite fluents opened with the opening of f_2.

The fluent learning algorithm maintains contingency tables that count co-occurrences of open and close events. We restrict the number of ticks, m, by which one opening must happen after another: m must be bigger than a few ticks, otherwise we treat the openings as simultaneous; and it must be smaller than the length of a short-term memory.[10] At each tick, the algorithm first decides which simple and composite fluents have closed. With this information, it can disambiguate which composite fluents opened at an earlier time (within the bounds of short term memory). Then, it finds out which simple and composite fluents have just opened, or might have opened. This done, it updates the open and close contingency tables for all fluents that have just closed. Next, it updates the χ_2 statistic for each table and it adds the newly significant composite fluents to the list of accepted fluents.

Two fluents learned by the algorithm are shown in Figure 5 (others are discussed in [6]). These fluents were never used by the robot for anything (besides learning other fluents) so they are not meaningful representations in the sense of Section 1, but they illustrate the kind of structural abstractions produced by the fluent learning process. The first captures a strong regularity in how the robot approaches an obstacle. Once the robot detects an object visually, it moves toward it quite quickly, until the sonars detect the object. At that point, the robot immediately stops, and then moves forward more slowly. Thus, we expect to see SAEB(NEAR-OBJECT,STOP), and we expect this fluent to start before MOVE-FORWARD, as shown in the first fluent. The second fluent shows that the robot stops when it touches an object but remains touching the object after the STOP fluent closes (SWEB(TOUCH,STOP)) and this composite fluent starts before and ends before another composite fluent in which the robot is simultaneously moving forward and pushing the object. This fluent describes exactly how the robot pushes an object.

Fluent learning works for multivariate time series in which all the variables are binary. It does not attend to the durations of fluents, only the temporal relationships between open and close events. This is an advantage in domains where the same episode can take different amounts of time, and a disadvantage in domains where duration matters. Because it is a statistical technique, fluent learning finds common patterns, not all patterns; it is easily biased to find more or fewer patterns by adjusting the threshold value of the statistic and varying the size of the fluent short term memory. Fluent learning elucidates the hierarchical structure of episodes (i.e., episodes contain episodes) because fluents are themselves nested. We are not aware of any other algorithm that is unsupervised, incremental, multivariate, and elucidates the hierarchical structure of episodes.

---

[10] The short term memory has two kinds of justification. First, animals do not learn associations between events that occur far apart in time. Second, if every open event could be paired with every other (and every close event) over a long duration, then the fluent learning system would have to maintain an enormous number of contingency tables.

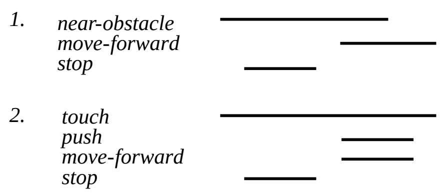
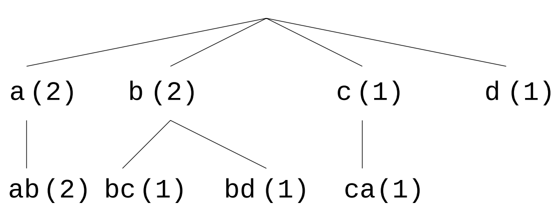


Fig. 5. Two composite fluents. These were learned without supervision from a time series of 22535 binary vectors of robot data.

The Voting Experts Algorithm The voting experts algorithm is designed to find boundaries of substructures within episodes, places where one macroscopic part of an episode gives way to another. It incorporates "experts" that attend to boundary entropy and frequency, and is easily extensible to include experts that attend to other characteristics of episode structures. Currently the algorithm works with univariate sequences of categorical data. The algorithm simply moves a window across a time series and asks, for each location in the window, whether to "cut" the series at that location. Each expert casts a vote. Each location takes n steps to traverse a window of size n, and is seen by the experts in n different contexts, and may accrue up to n votes from each expert. Given the results of voting, it is a simple matter to cut the series at locations with high vote counts. The algorithm has been tested extensively with sequences of letters in text: Spaces, punctuation and capitalization are removed, and the algorithm is able to recover word boundaries. It also performs adequately, though not brilliantly, on sequences of robot states. Research in that domain continues.

Here are the steps of the algorithm:
Build a prefix tree of depth $n+1$. Nodes at level i of an prefix tree represent ngrams of length i. The children of a node are the extensions of the ngram represented by the node. For example, $a\ b\ c\ a\ b\ d$ produces the following prefix tree of depth 2:



Every ngram of length 2 or less in the sequence $a\ b\ c\ a\ b\ d$ is represented by a node in this tree. The numbers in parentheses represent the frequencies of the subsequences. For example, the subsequence a b occurs twice, and every occurrence of a is followed by b.

For the first 10,000 characters in George Orwell's book *1984*, a prefix tree of depth 7 includes 33774 nodes, of which 9109 are leaf nodes. That is, there are over nine thousand unique subsequences of length 7 in this sample of text, although the average frequency of these subsequences is 1.1; most occur exactly once. The average frequencies of subsequences of length 1 to 7 are 384.4, 23.1, 3.9, 1.8, 1.3, 1.2, and 1.1.

Calculate boundary entropy. The boundary entropy of an ngram is the entropy of the distribution of tokens that can extend the ngram. The entropy of a distribution of a random variable x is just $-\sum Pr(x)\log Pr(x)$. Boundary entropy is easily calculated from the prefix tree. For example, the node a has entropy equal to zero because it has only one child whereas the entropy of node b is 1.0 because it has two equiprobable children.

Standardize frequencies and boundary entropies. In most domains, there is a systematic relationship between the length and frequency of patterns; in general, short patterns are more common than long ones (e.g., on average, for subsets of 10,000 characters from Orwell's text, 64 of the 100 most frequent patterns are of length 2; 23 are of length 3, and so on). Our algorithm will compare the frequencies and boundary entropies of ngrams of different lengths, but in all cases we will be comparing how *unusual* these frequencies and entropies are, relative to other ngrams of the same length. To illustrate, consider the words "a" and "an". In the first 10000 characters of Orwell's text, "a" occurs 743 times, "an" 124 times, but "a" occurs only a little more frequently than other one-letter ngrams, whereas "an" occurs much more often than other two-letter ngrams. In this sense, "a" is ordinary, "an" is unusual. Although "a" is much more common than "an" it is much less unusual relative to other ngrams of the same length. To capture this notion, we standardize the frequencies and boundary entropies of the ngrams. Standardized, the frequency of "a" is 1.1, whereas the frequency of "an" is 20.4. We standardize boundary entropies in the same way, and for the same reason.

Score potential segment boundaries. In a sequence of length k there are $k-1$ places to draw boundaries between segments, and, thus, there are 2^{k-1} ways to divide the sequence into segments. Our algorithm is greedy in the sense that it considers just $k-1$, not 2^{k-1}, ways to divide the sequence. It considers each possible boundary in order, starting at the beginning of the sequence. The algorithm passes a window of length n over the sequence, halting at each possible boundary. All of the locations within the window are considered, and each garners zero or one vote from each expert. Because we have two experts, for boundary-entropy and frequency, respectively, each possible boundary may garner up to $2n$ votes. This is illustrated in Figure 6. A window of length 3 is passed along the sequence `itwasacold`.

Initially, the window covers `itw`. The entropy and frequency experts each decide where they could best insert a boundary within the window. The boundary entropy expert votes for the location that produces the ngram with the highest standardized boundary entropy, and the frequency expert places a boundary so as to maximize the sum of the standardized frequencies of the ngrams to the left and the right of the boundary. In this example, the entropy expert favors the boundary between `t` and `w`, while the frequency expert favors the boundary between `w` and whatever comes next. Then the window moves one location to the right and the process repeats. This time, both experts decide to place the boundary between `t` and `w`. The window moves again and both experts decide to place the boundary after `s`, the last token in the window. Note that each potential boundary location (e.g., between `t` and `w`) is seen n times for a window of size n, but it is considered in a slightly different context each time the window moves. The first time the experts consider the boundary between `w` and `a`, they are looking at the window `itw`, and the last time, they are looking at `was`.

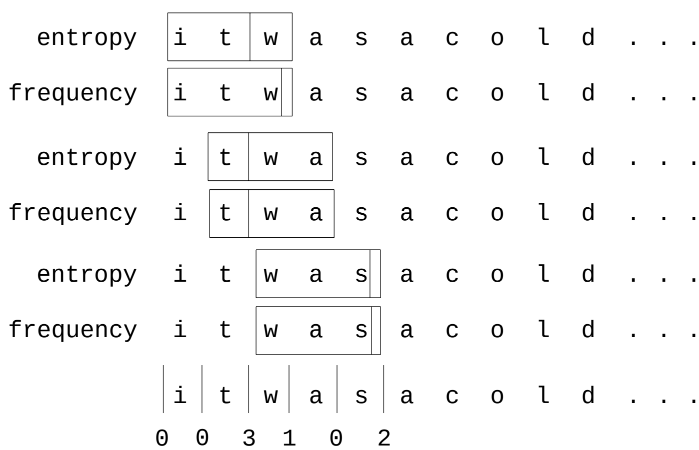


Fig. 6. The operation of the voting experts algorithm.

In this way, each boundary gets up to $2n$ votes, or $n = 3$ votes from each of two experts. The `wa` boundary gets one vote, the `tw` boundary, three votes, and the `sa` boundary, two votes.

Segment the sequence. Each potential boundary in a sequence accrues votes, as described above, and now we must evaluate the boundaries in terms of the votes and decide where to segment the sequence. Our method is a familiar "zero crossing" rule: If a potential boundary has a locally maximum number of votes, split the sequence at that boundary. In the example above, this rule causes the sequence `itwasacold` to be split after `it` and `was`. We confess to one embellishment on the rule: The number of votes for a boundary must exceed a threshold, as well as be a local maximum. We found that the algorithm splits too often without this qualification. In the experiments reported below, the threshold was always set to n, the window size. This means that a location must garner half the available votes (for two voting experts) and be a local maximum to qualify for splitting the sequence.

The algorithm performs well at a challenging task, illustrated below. In this block of text—the first 200 characters in Orwell's *1984*—all spaces and punctuation have been excised, and all letters made capital; and to foil your ability to recognize words, the letters have been recoded in a simple way (each letter is replaced by its neighbor to the right in the alphabet, and Z by A):

H S V Z R Z A Q H F G S B N K C C Z X H M Z O Q H K Z M C S G D B K N B J R V D Q D R S Q
H J H M F S G H Q S D D M V H M R S N M R L H S G G H R B G H M M T Y Y K D C H M S N G
H R A Q D Z R S H M Z M D E E N Q S S N D R B Z O D S G D U H K D V H M C R K H O O D C
P T H B J K X S G Q N T F G S G D F K Z R R C N N Q R N E U H B S N Q X L Z M R H N M R S
G N T F G M N S P T H B J K X D M N T F G S

Suppose you had a block of text several thousand characters long to study at leisure. Could you place boundaries where they should go, that is, in locations that correspond to words in the original text? (You will agree that the original text is no more meaningful to the voting experts algorithm than the text above is to you, so the problem we pose to you is no different than the one solved by the algorithm.)

To evaluate the algorithm we designed several performance measures. The *hit rate* is the number of boundaries in the text that were indicated by the algorithm, and the *false positive rate* is the number of boundaries indicated by the algorithm that were not boundaries in the text. The *exact word* rate is the proportion of words for which the algorithm found both boundaries; the *dangling* and *lost* rates are the proportions of words for which the algorithm identifies only one, or neither, boundary, respectively. We ran the algorithm on corpora of Roma-ji text and a segment of Franz Kafka's *The Castle* in the original German. Roma-ji is a transliteration of Japanese into roman characters. The corpus was a set of Anime lyrics, comprising 19163 roman characters. For comparison purposes we selected the first 19163 characters of Kafka's text and the same number of characters from Orwell's text. We stripped away spaces, puncuation and capitalization and the algorithm induced word boundaries. Here are the results:

	Hit rate	F. P. rate	Exact	Dangling	Lost
English	.71	.28	.49	.44	.07
German	.79	.31	.61	.35	.04
Roma-ji	.64	.34	.37	.53	.10

Clearly, the algorithm is not biased to do well on English, in particular, as it actually performs best on Kafka's text, losing only 4% of the words and identifying 61% exactly. The algorithm performs less well with the Roma-ji text; it identifies fewer boundaries accurately (i.e., places 34% of its boundaries within words) and identifies fewer words exactly. The explanation for these results has to do with the lengths of words in the corpora. We know that the algorithm loses disproportionately many short words. Words of length 2 make up 32% of the Roma-ji corpus, 17% of the Orwell corpus, and 10% of the Kafka corpus, so it is not surprising that the algorithm performs worst on the Roma-ji corpus and best on the Kafka corpus.

As noted earlier, the algorithm performs less well with time series of robot states. The problem seems to be that episode substructures are quite long (over six seconds or 60 discrete ticks of data, on average, compared with Orwell's average word length, around 5.) The voting experts algorithm can find episode structures that are longer than the depth of its prefix tree, but recall that the frequency of ngrams drops with their length, so most long ngrams occur only once. This means the frequency and boundary entropy experts have no distributions to work with, and even if they did, they would have difficulty estimating the distributions with any accuracy from such small numbers.

Still, it is remarkable that two very general heuristic methods can segment text into words with such accuracy. Our results lead us to speculate that frequency and boundary entropy are general markers of episode substructures, a claim we are in the process of testing in other domains. Recall that Oates used these heuristics to bootstrap the process of finding words in the speech signal.

6 Conclusion

The central claim of this paper is that programs can learn representations and their meanings. We adopted Dretske's definition that a representation is meaningful if it reliably indicates something about the external world and the indicator relationship is exploited to inform action. These criteria place few constraints on what is represented, how it is represented, and how representations inform action, yet these questions lie at the heart of AI engineering design, and answering them well requires considerable engineering skill. Moreover, these criteria admit representations that are meaningful in the given sense *to an engineer* but not to a program. This is one reason Dretske [12] required that the function of the indicator relationship be *learned*, to ensure that meaning is endogenous in the learning agent. Dretske's requirement leads to some philosophical problems [10] and we do not think it can survive as a *criterion* for contentful mental states [8]. However, we want programs to learn the meanings of representations not as a condition in a philosophical account of representation, meaning and belief, but as a practical move beyond current AI engineering practice, in which all meanings are exogenous; and as a demonstration of how symbolic representations might develop in infant humans.

What is the status of our claim that programs can learn representations and their meanings? As our adopted notion of meaning does not constrain what is represented, how it is represented, and how representations inform action, we have considerable freedom in how we gather evidence relevant to the claim. In fact, we imposed additional constraints on learned representations in our empirical work: They should be grounded in sensor data from a robot; the data should have a temporal aspect and time or the ordinal sequence of things should be an explicit part of the learned representations; and the representations should not merely inform action, but should inform two essentially human intellectual accomplishments, language and planning. We have demonstrated that a robot can learn the meanings of words, and construct simple plans, and that both these abilities depend on representations and meanings learned by the robot. In general, we have specified *how* things are to be represented (e.g., as transition probability matrices, sequences of means and variances, multivariate time series, fluents, etc.) but the contents of the representations (i.e., what is represented) and the relationship between the contents and actions have been learned.

Acknowledgments

The authors wish to thank the many graduate students who have worked on the Robot Baby project, particularly Marc Atkin, Brendan Burns, Anyuan Guo, Brent Heeringa, Gary King, Laura Firiou, and Matt Schmill. We thank Mary Litch, Clayton Morrison, Marco Ramoni and Paola Sebastiani for their lively contributions to the ideas herein. David Westbrook deserves special thanks from all the members of the Experimental Knowledge Systems Laboratory for his many contributions.

This work was supported by DARPA under contract(s) DARPA / USASMDC-DASG60 - 99 - C -0074 and DARPA / AFRLF30602-00-1-0529, and by a Visiting Fellowship Research Grant number GR/N24193 to support Paul Cohen from the Engineering and Physical Sciences Research Council (UK).

References

1. Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Seattle, Washington, 1987. American Association for Artificial Intelligence.
2. James F. Allen. An interval based representation of temporal knowledge. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 221–226, San Mateo, CA, 1981. Morgan Kaufmann Publishers, Inc.
3. Dana H. Ballard, Mary M. Hayhoe, and Polly K. Pook. Deictic codes for the embodiment of cognition. Computer Science Department, University of Rochester.
4. D.H. Ballard. Reference frames for animate vision. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligenc*. Morgan Kaufmann Publishers, Inc., 1989.
5. Paul Bloom. *How Children Learn the Meanings of Words*. MIT Press, 2000.
6. Paul Cohen. Fluent learning: Elucidating the structure of episodes. In *Proceedings of Fourth Symposium on Intelligent Data Analysis*. Springer, 2001.
7. Paul Cohen and Niall Adams. An algorithm for segmenting categorical time series into meaningful episodes. In *Proceedings of Fourth Symposium on Intelligent Data Analysis*. Springer, 2001.
8. Paul R. Cohen and Mary Litch. What are contentful mental states? dretske's theory of mental content viewed in the light of robot learning and planning algorithms. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.

9. Bndicte de Boysson-Bardies. *How Language Comes to Children.* MIT Press, 2001.
10. Daniel Dennett. Do it yourself understanding. In Daniel Dennett, editor, *Brainchildren, Essays on Designing Minds.* MIT Press and Penguin, 1998.
11. Fred Dretske. *Knowledge and the Flow of Information.* Cambridge University Press, 1981. Reprinted by CSLI Publications, Stanford University.
12. Fred Dretske. *Explaining Behavior: Reasons in a World of Causes.* MIT Press, 1988.
13. Joseph B. Kruskall and Mark Liberman. The symmetric time warping problem: From continuous to discrete. In *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, 1983.
14. B. Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, 2000.
15. George Lakoff. *Women, Fire, and Dangerous Things.* University of Chicago Press, 1984.
16. D. B. Lenat. Cyc: Towards programs with common sense. *Communications of the ACM*, 33(8), 1990.
17. D. B. Lenat and R. V. Guha. *Building large knowledge-based systems: Representation and inference in the Cyc project.* Addison Wesley, 1990.
18. Paola Sebastiani Marco Ramoni and Paul Cohen. Bayesian clustering by dynamics. *Machine Learning*, to appear(to appear):to appear, 2001.
19. John McCarthy. Situations, actions and causal laws. Stanford Artificial Intelligence Project: Memo 2, also, http://wwwformal.stanford.edu/jmc/mcchay69/ mcchay69.htm, 1963.
20. J. T. Oates. *Grounding Knowledge in Sensors: Unsupervised Learning for Language and Planning.* PhD thesis, Department of Computer Science, University of Massachusetts, 2001.
21. Tim Oates Paul R. Cohen, Marc S. Atkin and Carole R. Beal. NEO: Learning conceptual knowledge by sensorimotor interaction with an environment. In *Proceedings of the First International Conference on Autonomous Agents*, pages 170–177, 1997.
22. David Pierce and Benjamin Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence Journal*, 92:169–229, 1997.
23. Marco Ramoni, Paola Sebastiani, and Paul Cohen. Multivariate clustering by dynamics. In *Proceedings of the Seventeenth National Conference on AI*, pages 633–638. AAAI Press/The MIT Press, 2000.
24. E. Rosch and C. B. Mervis. Family resemblances: Studies in the internal structure of categories. *Cognitive Psychology*, 7:573–605, 1975.
25. Robert St. Amant and Paul R. Cohen. Intelligent Support for Exploratory Data Analysis. *The Journal of Computational and Graphical Statistics*, 1998.
26. Robert St. Amant and Paul R. Cohen. Interaction With a Mixed-Initiative System for Exploratory Data Analysis. *Knowledge-Based Systems*, 10(5):265–273, 1998.
27. Luc Steels. *The Talking Heads Experiment: Volume I. Words and Meanings.* Laboratorium, Antwerpen, 1999. This is a museum catalog but is in preparation as a book.
28. Paul Cohen Tim Oates, Matthew Schmill. Identifying qualitatively different outcomes of actions: Gaining autonomy through learning. In *Proceedings Fourth International Conference on Autonomous Agents, pp 110-111.* ACM, 2000.
29. Paul Cohen Tim Oates, Matthew Schmill. A method for clustering the experience of a mobile robot that accords with human judgments. In *Proceedings of Seventeenth National Conference, pp. 846-851.* AAAI Press/The MIT Press, 2000.

30. Paul Utgoff and Paul R. Cohen. Applicability of reinforcement learning. In *The Methodology of Applying Machine leraning Problem Definition, Task Decompostion and Technique Selection Workshop, ICML-98*, pages 37–43, 1998.