Language Games in Wubble World: Final Report

Paul R. Cohen, Wesley Kerr, Daniel Hewlett, Shane Hoversten, Yu-Han Chang

June 4, 2008

Contents

1 Introduction

The goal of this project was to have machines learn language in the same way as young children do — from competent, facilitative speakers who strive to associate their language with what's going on in the child's environment. With the support of DARPA, we have spent a year creating Wubble World (`www.wubble-world.com`), an online game environment in which children interact with softbots — called wubbles —using natural language. We demonstrated that wubbles could learn some aspects of the meanings of some words, and that children can remain engaged with Wubble World for sufficiently long to teach their wubbles word meanings. We also discovered some limitations of associative learning of word meanings, and of our representations and algorithms.

Section 2 of this report describes our reasons for building Wubble World and what we expected to learn from the exercise. It also describes four Wubble World games. Section 3 describes a simple (indeed, simplistic) representation of word meanings and an algorithm for learning word meanings from children as they play games in Wubble World. Section 4 reports the results of several word-learning experiments. As discussed in Section 6, this work had two major limitations: Children would not remain engaged with simple Wubble World games for more than an hour or two, and our representation of word meanings was too poor to capture the meanings of important word classes. Section 7 discusses recent work on better representations and word-learning algorithms.

2 Wubble World

Wubble World attempts to reproduce the kinds of environments in which children learn language, not to model human language learning but to accelerate machine language learning. Human and machine language learning are distinctly different. Machines learn language models by extracting statistics from gigabytes of text. Children learn language from adults who point to objects, elaborate on their children's sentences, engage children in tasks and talk about them, and in many other ways relate language to what's going on in the immediate environment. When a child says, "More!" the parent says, "More milk?" and points to the empty milk glass. When a child says, "Milk!" the parent says, "Yes, more milk. I'll get you more milk. You drank all your milk. Would you like more milk?" Paul Bloom summarizes human word learning as follows:

> There is nothing else — not a computer simulation, and not a trained chimpanzee — that has close to the word learning abilities of a normal 2-year-old child.
>
> Here is how they do it: Young children can parse adult speech (or sign) into distinct words. They think of the world as containing entities, properties, events, and processes; most important, they see the world as containing objects. They know enough about the minds of others to figure out what they are intending to refer to when they use words. They can generalize; and so when they learn that an object is called "bottle" and an action is called "drinking", they can extend the words to different objects and actions. They can also make sense of pronouns and proper names, which refer to distinct individuals, not to categories; and so they understand that "Fido" refers to a particular dog, not to dogs in general.
>
> These capacities improve in the course of development. Children become better at parsing utterances into words, at dividing the world into candidate referents for these words, and at figuring out what other people are thinking about when they speak. Months after their first

> words, they possess enough understanding of the language to learn from linguistic context, exploiting the syntactic and semantic properties of the utterances that new words belong to. This enables the learning of many more words, including those that could only be acquired through this sort of linguistic scaffolding. (from http://www.bbsonline.org/Preprints/Bloom/Referees/).

Everything in the paragraph that begins, "Here is how they do it," can be built today. Indeed, many cognitive scientists have tackled various parts of Bloom's recipe for lexical acquisition [14, 13, 22, 15, 16, 20, 21, 19, 17, 6, 5]. However, we know of no attempt to tackle the whole problem — scaffolding by facilitative speakers, bootstrapping and gradual acquisition of word meanings, and the interaction of syntax and lexical acquisition — on a very large scale.

Why do interactions like this produce such rapid mastery of language, whereas gigabytes of text and enormously sophisticated learning algorithms produce at best shallow semantics? The reason is that, to a human language learner, the semantics of sentences are immediately accessible in scenes—in what is going on when the language is uttered. In contrast, a machine, given only text, with no access to scenes, can extract only poor, shallow semantics from distributional statistics, syntactic classes, and other structural features of the text.

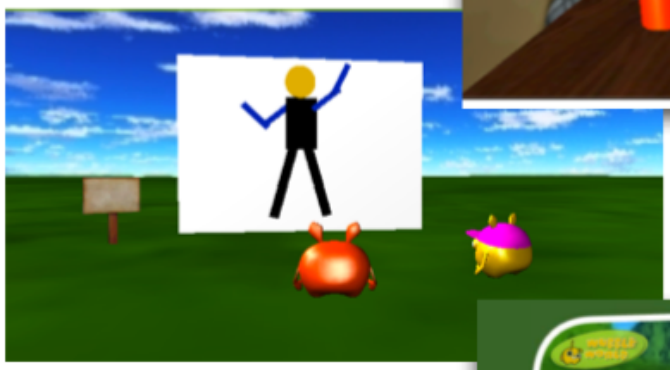
One way to recreate the parent-child language learning environment is to build a computer game in which players have the role of parents and softbots have the role of language learning children. The game would require "parents" to speak to "children" in natural language about what's going on in the game. A popular game could generate very large *sentence-scene corpora* — corpora of sentences spoken by players and softbots aligned with corpora of formal descriptions of what's happening in the game. Algorithms would then process these parallel corpora to learn the meanings of words and perhaps syntax and pragmatics, as well.

Wubble World has achieved some of this. It is a web site where children can adopt and communicate with softbots in natural language, while engaged in simple games (`www.wubble-world.com`).

We built four games, screen shots of which are shown in Figure 1. In Jean's Room, children type instructions to their Wubbles to get them to accomplish tasks, such as building towers of blocks. There are three versions of Jean's Room with slightly harder tasks in each. In the *Introduction Room* children familiarize their wubbles with different objects. After a short period of interaction in the Introduction Room, a larger set of varied objects is placed in the room. The wubble begins questioning the child about these objects, and in the process gathers additional data with which to refine recently learned concepts, such as "red" and "cube." The wubble gathers this information with assertions like: *Click on all of the red objects.* After this, the wubble and child are promoted to the *Apple Room*, which contains several objects differing in size, shape, and color. On a platform in one corner of the room is an apple. The child's goal is to get her wubble to retrieve the apple. The physics of Wubble World won't let wubbles jump as high as the platform, so the child must instruct her wubble to build a staircase from the objects in the room. The final room is the *Sorting Room* where the child works with her wubble to line up the objects in the room from tallest to shortest.

The Dragon Helper game is a more Warcraft-like version of Jean's Room in which the goal is to navigate caverns, fend off attackers, and collect gold coins. Wubble Pictionary is a kind of chatroom in which children draw pictures from shapes and converse about them. Initially we thought the wubbles might learn language from these interactions between children, but later decided the game

Jean's Room, in which children type commands to their wubbles to accomplish tasks such as building a staircase from blocks.



Wubble Pictionary, in which one child draws an image on the screen and other children try to identify what it depicts, and wubble "listen in"

The Sheep Game, in which wubbles under the manual control of children herd sheep into goals. In this team game children coordinate their wubbles' activities by typing messages or speaking to each other via voice over IP.

Dragon Helper: An edgier version of Jean's Room. The goal is to find all the gold coins in a dungeon. The wubbles can pick up coins, the dragons can protect the wubbles with crossbows. Children type English messages to their dragon companions.

Figure 1: Four Wubble World games. Most of our experimental work was done in Jean's Room.

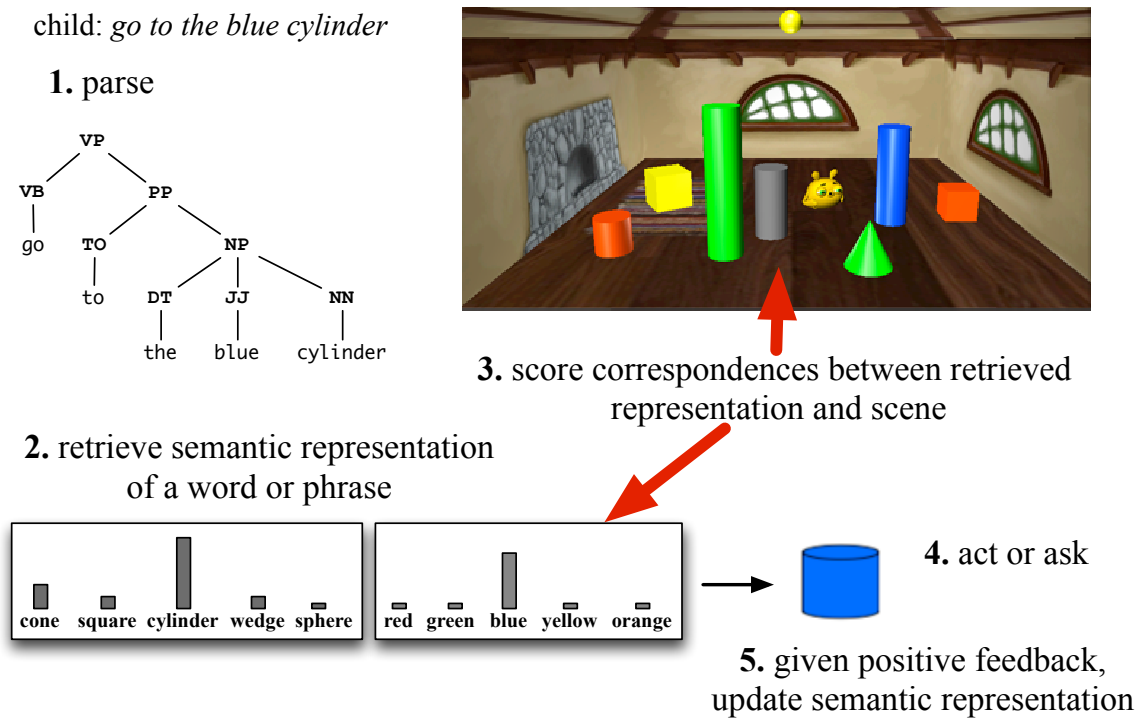


Figure 2: The steps involved in learning word meanings in Wubble Room.

does not provide sufficient scene information to support language learning. It remains on the Wubble World site for recreation, only. The Sheep Game is a real-time strategy game in which children team up to herd escaped sheep into their pens. Teams compete to catch the most sheep. Unlike the other games, in which messages are typed, the Sheep Game encourages children to speak to each other and the speech is transcribed by speech-to-text software. Jean's Room and Dragon Helper are supposed to mimic young children (wubbles) learning language from direct interaction with parents (the players). Wubble Pictionary and the Sheep Game mimic young children learning language by listening to conversation in a scene by competent speakers.

3 Word Learning

Wubble word learning is illustrated in Figure 2. Word learning is perhaps a misleading phrase because what's learned is a concept, the name of which is a word or a phrase. Wubbles maintain simple, feature-based representations of concepts. For objects, the features are shape, color and size; for prepositions, the features describe ways of dividing up space. Each feature has an associated weight or probability distribution, as shown in step 2 of Figure 2. For instance, the weights associated with the feature values $shape = cylinder$ and $color = blue$ are higher than the weights associated with $shape = sphere$ and $color = orange$.

The wubble learns incrementally, on-line, by updating the weights associated with each feature.

When a wubble encounters a sentence, it parses it (step 1, Fig. 2) and retrieves from memory the concepts that correspond to the words and phrases in the sentence (step 2, Fig. 2). If a word is new, the wubble creates a set of weight distributions, one for each feature, initialized with a uniform distribution. Next, the wubble *imagines* an object (or adjective, or preposition) by sampling from the retrieved distributions of each feature. Sampling is probabilistic, so there is a small probability that the wubble will imagine a blue cone, say, instead of a blue cylinder (the output of step 2, Fig. 2). It then compares the imagined object with the objects in the scene, and it calculates scores for the correspondences between the imagined object and those in the scene (step 3, Fig. 2). After that, if the wubble is certain enough it has identified the referent of the word or phrase, it acts. For instance, in the scene in Figure 2, the wubble would go to the blue cylinder. However, if it remains uncertain about the referent of a word or phrase, it asks the player to point to the referent (step 4, Fig. 2). Then the wubble updates the feature representation associated with the word or phrase to make it more like the object in the scene (step 5, Fig 2). In this way, it quickly learns the feature representations that should be associated with words. A more formal description of word learning in Wubble World may be found in [10].

4 Experiments and Results

This section presents the results of two experiments: One tests whether wubbles can learn word meanings from artificially generated sentences (Sec. 4.1), the other tests whether children are engaged by Wubble World and whether wubbles can learn word meanings from interactions with children (Sec. 4.3).

4.1 Experiment 1

To validate our word-learning algorithm, we performed experiments in a number of different environments. The first experiment measured how quickly and with what level of mastery a wubble can learn a vocabulary of nouns and adjectives. The second experiment measured the wubble's mastery of preposition words.

Adjective and Noun Vocabulary We tested a wubble's mastery of adjectives and nouns in two separate types of environments: a *facilitative* environment containing relatively few objects, and a *challenging* environment, which contains 100 objects. It is our hypothesis that the facilitative environment fosters learning since it contains fewer referents for the wubble to reason about.

We generated a set of testing sentences that was held constant across all trials. Each sentence in the testing set is similar in form to the sentence: "choose a small red column." The sentences contain two adjectives, one describing the object's size and the other describing its color, and a noun describing its shape. This set of sentences contained 14 words that the wubble was required to learn.

The next step was to populate the room with a set of objects. We ensured that at least one object for each sentence in the test set existed in the room. The rest of the objects in the room were randomly selected (with replacement) from the set of possible objects.

We generated a set of training sentences that provide guidance for a wubble to learn language. For each object in the environment, we generated the sentences that describe it, of the forms "choose a *type*," "choose a *color* object," "choose a *size-[x,y,z]* object," and all possible combinations. For instance, "choose a *color type*" combines the type sentence with the color sentence and results in a new set of referent objects. Each sentence generated by the process can refer to several objects within the scene.

We presented this collection of sentences as training sentences for the wubble, keeping the training and test sentences separate, so that while the wubble may have trained on the *individual* words, it has never experienced the specific combination of words in the testing sentence.

We defined one epoch as a sequence of 10 randomly selected training sentences, and after each epoch the wubble was presented the set of test sentences. Performance was measured by counting the number of guesses required to correctly identify the referent of the sentence. Perfect performance resulted in one guess. One trial is a collection of nine epochs with an initial testing phase prior to training.

The protocol explores our initial hypothesis that *facilitative* environments foster learning. We also ran several experiments to explore knowledge transfer for the wubble by augmenting the wubble with training experiences from a different environment. To simplify the setup of the transfer experiment each wubble started with the correct concept for each color word in the testing sentences. The task was to learn the rest of the vocabulary.

Preposition Vocabulary Testing a wubble's mastery of prepositions was relatively straightforward. Since the wubble automatically determines the landmark of the preposition we were able to run experiments in a simple environment that contained only one object. This focused the evaluation of performance to the preposition word, rather than identifying the correct landmark. We presented the wubble with one sentence similar to: "choose a point behind the wedge." The challenge was to identify a region of space that is *behind* the wedge.

We cannot combine prepositions in the current system, such as "in front of and to the left of." Therefore, when testing a wubble on prepositions, we needed only one sentence. Each time the sentence was presented, it acted as both a training instance as well as a testing instance. The wubble's performance is the number of different locations tried before identifying the correct location.

4.2 Experiment 1 Results

The results for adjectives and nouns are displayed in Fig. 3*(a)*. Each point is an average of 100 trials. The error bars are two standard deviations wide. These results show that in *facilitative* environments the wubble can learn the meanings of words in fewer training epochs. When we introduced color concepts in the transfer case, the wubble has an automatic leg up because it is able to reduce the space of correct objects and quickly identify referents. By combining both the *facilitative* environment and the transfer case, the wubble is able to correctly identify the referent almost immediately.

Fig. 3*(b)* shows the results for learning a single preposition over 100 trials. It takes three training sentences, on average, for a wubble to understand the meaning of a preposition.

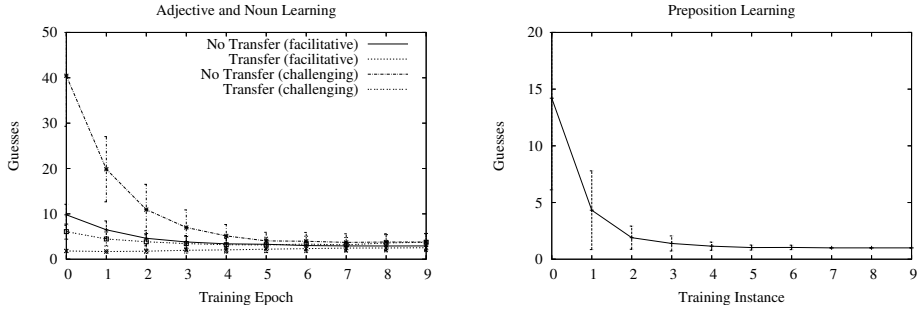


Figure 3: *(a)* Results for nouns and adjectives. *(b)* Results for prepositions.

We found several cases where our system is unable to learn the correct concept, all related to interactions between features. Since feature distributions are treated independently, words defined as a disjunction of values in multiple distributions cannot currently be learned. Examples include "large" (at least `big` in any dimension) and "far away" (`negative-far` or `far` on any axis).

4.3 Experimental 2

We designed an experiment to test the hypotheses that children will be engaged by Wubble World and that their interactions with wubbles are sufficient for the wubbles to learn word meanings.

We hired 10 children between the ages of 11 and 14 to participate in the experiment. Each child received $20 for participating. Each was paired with an experiment proctor who introduced the child to Wubble World tasks and answered any questions.[1] All of the children's interactions with Wubble World were confined to the Jean's Room area described earlier. Interactions were in English, which the children typed without difficulty on a standard computer keyboard. There was only one restriction on the children's utterances: They were told that the wubble could only understand half a dozen verbs: turn, walk, jump, pick up, put down, and choose. The wubble would not do anything if the children used other verbs. This ensured that the wubble could act on all of the children's utterances.

As described earlier, each child started with a wubble in the Introduction Room, then moved on to the Apple and Sorting rooms. We did not restrict the length of the experiment, and several children spent more than two hours working with their wubbles in the Apple and Sorting rooms. Although the tasks to be done in these rooms did not change, the wubbles' ability to understand English improved, and this kept some children engaged.

Here is a short snippet of 27 sentences typed by a child as she interacted with her wubble:

```
choose a giant red cylinder, pick it up, put it down, choose a tall red cube,
go to it,  pick it up, go to the right of the giant red cube, put it down,
```

[1]The USC Institutional Review Board has ruled that Wubble World does not constitute an experiment and does not require IRB approval. We do not solicit, extract or store any information about any participant in Wubble World.

```
choose a red cylinder, go to it, pick it up, choose a re dube, go in front of it,
no, put it down, choose a green cylinder, no, go to it, pick it up,
choose a red cylinder, no, go to the left of it, put it down, choose a red cylidner,
go to it, pick it up, choose a green cylinder
```

While it was gratifying to see children convinced that their individual wubbles were learning, we analyzed learning by pooling the interactions between children and their wubbles from all ten sessions — each session being a sequence of sentences typed by the child to his or her wubble. For each sentence, the word learning algorithm worked on learning the meaning of each constituent noun, adjective, and preposition. A word and its associated scene constitute a single training instance.

We selected 23 words from the children's transcripts and specified an "ideal" concept for each. Each concept is described by a set of features. A feature is *defining* iff every instance of the concept must have the same value of that feature. For example, every blue object must have the value "blue" for its color feature. An ideal feature vector representation of a concept is constructed by setting probability of each defining feature value to 1.0, and setting the probability distributions over values of non-defining features reflect the unconditional probabilities of these feature values in the environment. For example, if there are only ten cubes and five cylinders in the environment, then the ideal feature vector representation for blue objects would have $Pr(color = blue) = 1$, and $Pr(shape = cylinder) = .333$, $Pr(shape = cube) = .666$.

To show that the wubbles learned the concepts associated with words, we compared the learned concepts for the 23 words to the ideal feature-based representations using the Kullback-Leibler (KL) distance [11]. Smaller KL distances indicate that the feature representation of a concept is more similar to the ideal feature representation.

After the experiment we debriefed the children to get a qualitative analysis of the game — what the children liked, what worked and didn't work, how the children would improve the game, and so on.

4.4 Experiment 2 Results

The 23 target words. We tested whether it is possible to learn approximations to the ideal feature based representations of 23 words. Aggregating the training instances from all students yielded 957 sentences in which one or more of these words appeared. Figure 4, shows the *average* KL distance, over the 23 words, after successive training instances. Although each training instance affects the learned representation of just one word, we see that the average KL distance decreases with training, indicating that, for these 23 words, their feature based representations approach the ideal representations.

Not all word feature vectors followed an inexorable trajectory toward the ideal feature vector. Tracking the KL distance for the word "blue" we observe in Figure 5 that divergence decreases rapidly, then starts to grow, slowly. This is because the final eight training instances of a "blue" object were all the same size. The word learning algorithm learned that a "blue" object is not only *blue* but also a particular sized cube. More training instances, in more variable environments, will fix this problem.

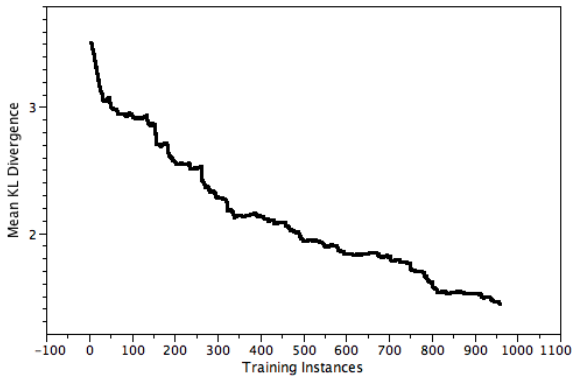


Figure 4: Average KL distance between learned word meanings and ideal meanings for 23 words, given children's interactions in English with their wubbles.

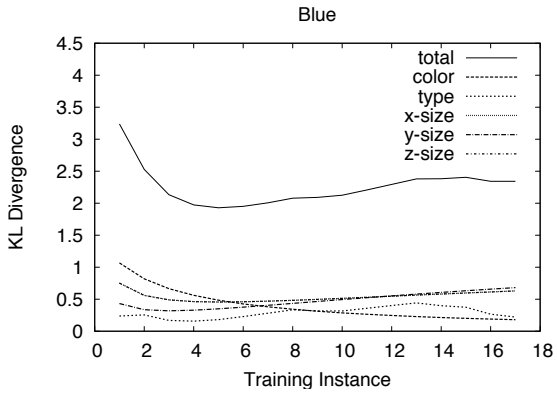


Figure 5: KL distance of "blue" over 17 training instances.

Other words. Of course, the children used many more than 23 words in their interactions with their wubbles. We did not define ideal feature based representations for all of these words, but we did attempt to characterize whether the word learning algorithm was learning *some* definition for these words, even if it wasn't what we considered the correct definition. One such assessment is given by the entropy of the feature vector representation of a word. High entropy means a uniform distribution of mass over all the features, or high uncertainty about which features define a word. So if the entropy for a word decreases as the number of training instances increases, it means that the wubble is becoming more certain that particular features define the word.

To illustrate, consider the word "end," used by just a handful of children. Wubbles cannot learn the common meaning of "end" because none of the word features describes the relationships between objects and regions of space that are denoted by the word.

One girl persisted in trying to teach her wubble "end" with sentences like, "move to the end of the orange cubes" (denoting a location at the end of a row of cubes). Figure 6 shows the entropy

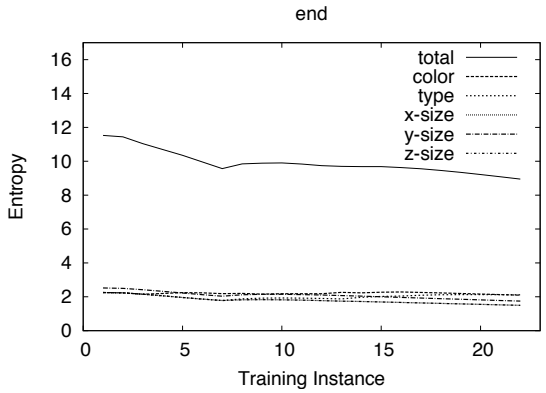


Figure 6: Final entropies of features of "end" after 23 sentences.

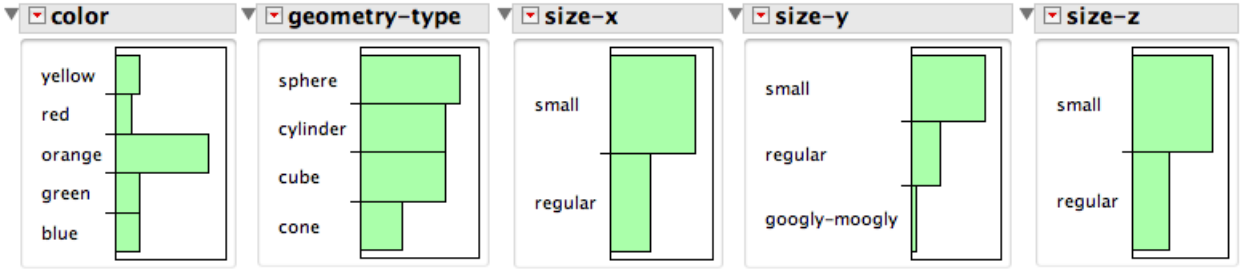


Figure 7: Final training distribution of "end" after 23 sentences.

of each of five word features – color, size-in-x, size-in-y, size-in-z, and geometry – over 23 sentences (end-size-x and end-size-y have identical entropies throughout). The total entropy, which is just the sum of the entropies of the five features, decreases, although there is a bump, caused by increasing entropy of the geometry feature, in the first ten sentences. It appears that "end" is becoming associated with a particular object (an orange one) at the end of a row objects. The interesting thing about this is that the speaker used the word "end" to refer to two locations in Wubble Room, one near the end of a clump of cylinders, the other near the end of a row of cubes. In both cases, the nearest object to the end was short and orange, and this is the meaning that the wubble learns for "end." Figure 7 shows the final values for the five word features. Orange is the dominant color, geometry is pretty uniform, and size-in-y is distinctly "small."

We discuss the limitations of wubbles' word features in Section 6, yet despite these limitations, children had the impression that their wubbles were learning word meanings. In the previous example, the girl who trained her wubble to understand "end" thought that the training was successful. It was not until later, when we debriefed her, that she came to understand that the wubble had associated "end" with particular objects. She apparently *wanted to believe* that her wubble had learned the common meaning of "end." This illusion has been documented in other contexts, as well, notably Dennett's article about his experiences judging the Loebner Prize competition [7].

Apparently, we cannot help ascribing intelligence to and intentionality to artifacts.

This probably explains why Wubble Room, which is really quite dull as computer games go, captivated seven of the ten children who played it. (It was difficult to get reasons from the other three, but our impression is that they just did not think the game was interesting or engaging.) Those who did all mentioned how cool it was to *teach* the wubbles, and they were impressed that the wubbles learned. As demonstrated by Wubble Room, the *Black & White* series, and the *Creatures* series, we suspect that having game characters learn, especially with players teaching them, will continue to be an opportunity for game developers.

5 Other Games

It surprised us that this version of Wubble World engaged children, but even the most interested child can play it for only an hour or so before running out of things to do. So we have developed some other games along the same lines games where players must use language to refer to the scene. The most popular, so far, is called the Sheep Game (Fig. 1). It is a multiplayer game with teams of wubbles competing to herd sheep into goals.

Players can communicate over a text-based chat link, but it is difficult to control the wubbles and type at the same time, so we added the ability to speak to ones team members, and hoped that speech-to-text software would transcribe the speech, and wed get our sentence-scene corpus that way. Unfortunately, untrained speech-to-text software doesnt work well with eleven-year-old girls chirruping about wubbbles. We tried IBM, Dragon and Microsoft speech-to-text products with no real success. Training helped, but it is not a viable approach for Wubble World, as children don't want to recite canned text for 30 minutes, they want to play.

In an attempt to learn whether Wubble World would be engaging to children if wubbles were more capable, physically, cognitively, and linguistically, we ran a Wizard-of-Oz study with two of the children from Experiment 2. They were told that we had developed new, more powerful wubble technology, and they would be able to say anything they wanted to their wubbles. In addition, the study was conducted in the Dragon Room (Fig. 1), which provided a wider range of activities. The wubbles were actually controlled by a graduate student in an adjacent room, and all the wubble's dialog was made up and typed by the student. The children enjoyed this interaction very much, and one adopted some Goth affectations in dealing with her wubble, naming it Minion and insisting that it address her as Dark Mistress:

```
(1187631287421 child "call me dark mistress")
(1187631327114 wubble "yes dark mistress")
(1187631363882 child "pick up the barrel minion")
(1187631397051 child "put the barrel down by the table")
(1187631425483 child "fetch the red crate minion")
(1187631458411 child "put the red crate down by the barrel")
(1187631491052 child "jump on the red crate")
(1187631508867 child "jump on the barrel minion")
(1187631529005 child "fetch the coin minion")
(1187631552028 child "good job minion")
(1187631557846 wubble "thank you dark mistress")
```

```
(1187631583516 child "fetch the green crate")
(1187631603580 child "jump on the red crate")
(1187631615309 child "jump on the table")
(1187631628924 child "put fdown the green crate")
(1187631665422 child "put down the green crate on the table")
(1187631696574 wubble "ta da")
(1187631701966 child "good job")
(1187631707343 wubble "thank you dark mistress")
(1187631710317 child "jump on the green crate")
(1187631731677 child "jump from the green crate onto the chair")
(1187631744285 child "pick up the coin")
(1187631779046 wubble "we have 2 coins")
(1187631780941 child "good job minion")
(1187631788735 wubble "thank you dark mistress")
(1187631833153 child "keep the coin minion")
(1187631842649 wubble "what coin")
(1187631872047 wubble "I have that coin")
(1187631882001 child "come to me minion")
(1187631936768 child "use the crates and barrels to come to me")
(1187631973281 child "good job")
(1187631979334 wubble "thank you dark mistress")
(1187632005414 child "fetch the coin")
```

This sort of dialog went on for more than two hours, giving us some confidence that children will play longer with more capable wubbles. Perhaps the biggest surprise was that neither child realized that the entire set-up was fake, even though the graduate student would sometimes appear in the children's lab to fix bugs and then disappear moments before the wubble started talking again. When we debriefed the children they complimented the new wubble and said it communicated just like a human (they particularly liked that it said "thank you"). To this day they have not seen through the ruse.

5.1 If we build it, will they come?

We debriefed every child after each session with Jean's Room, Sheep, and Dragon Helper. Overall, the children were very enthusiastic, though it is difficult to know whether this was because of Wubble World or because they got to play with their friends in a "cool"" laboratory setting. Eight of the ten participants were girls, and the giggle factor was high. Even so, they all had plenty of suggestions about how to improve the games. The following comments, sent to us by email, are typical:

1. I enjoyed teaching my wubble about objects in a room.
2. The wubbles were very cute and I liked the format of the website.
3. I appreciated the fact that there was no personal information required to play.
4. The wubble drawing game was very fun and so was talking to the other wubbles.

5. In the sheep game, it was difficult to work as a team. The gardens did not impact the game that much. I understand they were designed to make the game slower but it did not affect the speed that much and the gardens were awkwardly placed so removing sheep from the garden was a waste of time.

6. It was fun to be able to chat amongst your teammates.

7. I liked the fact that on the sheep game you could write your thoughts or commentary and both teams could see it but on the earphones only your team could hear you.

8. Typing in the box in the sheep game was too time-consuming because you had to switch between the game and the box.

9. The rooms in which you taught the wubbles were the best part. I enjoyed the challenges of teaching my wubble.

10. The bubbles that you could acquire were not as effective as they could be. For instance, the sticky bubble did not collect sheep very well.

11. A gravity bubble would be very cool and it would send all of the sheep to your wubble. All of the sheep would form a ring around your wubble and then you could walk them to the end zone.

12. As the website continues to grow and more rooms are added, I think that it will become very popular and prosper.

13. It was a great idea and I am thankful that I was privileged to experiment with the wubbles. Thank you.

6 Discussion

We demonstrated that children will play Wubble World for an hour or two, and that more interesting games will keep them engaged for longer. Our word learning algorithm quickly learned the meanings of some nouns, adjectives, and prepositions from positive instances, only. We demonstrated a kind of syntactic bootstrapping, where new words were assigned semantic representations based on their hypothesized syntactic classes. Wubbles can ask very simple questions (e.g., "Is this the red block?", and "Am I behind it?", and "Point to the green cube.") to help it learn the referents of words.

The speed with which wubbles learn word meanings from positive instances is due to a process we called "imagining" in Section 3 and illustrated in steps 2–5 in Figure 2. When the wubble hears a word or phrase, such as "blue cube," it imagines a possible referent by sampling from the probability distributions over features associated with the word or phrase, then asks whether anything in the scene matches this imagined referent, and if so, it modifies the distributions over features to make them better match the referent in the scene. We have not yet analyzed this learning method in a formal way; in particular, we can make no guarantees about the accuracy of word meanings learned from positive instances only. We think such an analysis will be worthwhile

and might shed some light on the ability of children to learn word meanings from mostly positive instances.

By the official end of the project (around the end of 2007) we had identified several critical limitations of the Wubble World project, and we have spent the last few months working to fix them (Sec. 7 describes some of this work). The most serious limitations are these:

- The feature-based representation of word meanings is poor in the sense that it can only represent color, shape, size, and some spatial relationships. Meanings expressed in terms of other semantic attributes (e.g., verbs, adverbs, intentional terms, etc.) cannot be learned.

- We were able to learn three classes of words — nouns, prepositions and adjectives — each of which had a characteristic set of features. Other classes of words would require not only additional semantic features but also some modifications to the parser. Here's why: Syntactic bootstrapping works by parsing a sentence and guessing at the syntactic class of each unknown word, and creating an "empty" feature vector of that class for that word. This means wubbles cannot learn any word meanings that are not associated with syntactic classes. Suppose we wanted wubbles to learn words denoting composites of objects, such as "tower" (of several blocks). One semantic attribute of composites is that they have components, another is the physical relationships between the components. Even if we created a feature-based representation of the meaning of words that denote composite objects, we would still have to modify the parser to recognize that an unknown word is of this type, so it could create the appropriate feature vector representation of the meaning of an unknown word. This clearly isn't feasible, as syntax generally underconstrains semantics. One cannot tell by looking at the structure of "put a block on the tower" whether "tower" is a composite object or not.

- The Wubble World games, themselves, must be more engaging and varied. Unfortunately, the tools we used to develop the games originally — Flash and Director — are not suitable for sites that serve and collect data from many users. It was necessary to rebuild Wubble World in an open-source, Java-based environment. This took months, but now we at least have an environment that can handle many users, if the project grows.

The original conception of Wubble World was that it would produce parallel corpora: a linguistic corpus of representations of sentences and perhaps gestures, and a corpus of representations of what's happening in the wubble's world, perhaps including the wubble's intentions and its inferences about the speaker's intentions. Over time we have come to think of these as linguistic (sentence) and semantic (scene) corpora, and much of the last six months has been spent developing representations and learning algorithms for the semantic, or scene, corpus. At first we thought it would be easy enough to borrow a formal representation of scenes from, say, the knowledge representation or common sense reasoning or computer vision communities. No such luck! We found nothing sufficient off-the-shelf to represent the state of the wubble's environment. So we have been developing a semantic representation for Wubble World, ourselves. The following section describes two efforts, one to represent the semantics of verbs, the other for more static aspects of scenes.

7 Better Semantic Representations

7.1 Relational Representations

The first requirement we set for semantic representations is that they be *graphs* in which nodes represent things, relations, or states and arcs represent binary relations. (n-ary relations would be handled in the usual way, by creating nodes for relations, but we haven't needed them, so far.) Figure 8 represents a snapshot — a moment in time — during the Sheep game. One can see that this graph represents several sheep, and the fact that each is an instance of the type **sheep**. It represents other objects such as a wubble called **demon**, and some spatial relationships between this wubble and the sheep. It represents objects called wrenches and powerups, and the fact that all of these "collide" with the floor. It shows that some of the objects are in motion, and through relations such as **SwiftAway(x,y)** it shows that some objects are moving away from each other swiftly.

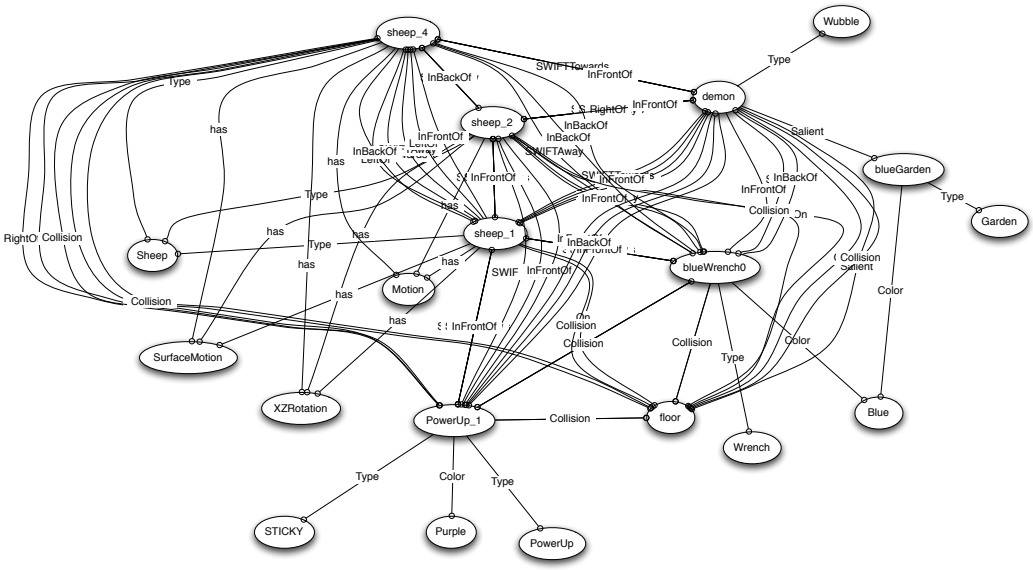


Figure 8: Semantic graph for a snapshot of a scene during the Sheep game. At approximately this time the sentence, "The wubble is in front of the sheep" was uttered.

Many other predicates are not shown in Figure 8; in all, we use roughly 200 predicates to characterize the state of a Wubble World at each instant.

We built a "scene parser" to generate graphs like Figure 8 automatically. We also developed an algorithm to *learn* representations of the dynamics of Wubble World by finding patterns in sequences of graphs. This algorithm is discussed in the following section.

Around the time this graph was captured, the player uttered the sentence "The wubble is in front of the sheep." A dependency parse of this sentences is shown in Figure 9.

Our sentence-scene corpus comprises dependency parses of sentences and snapshots of Wubble

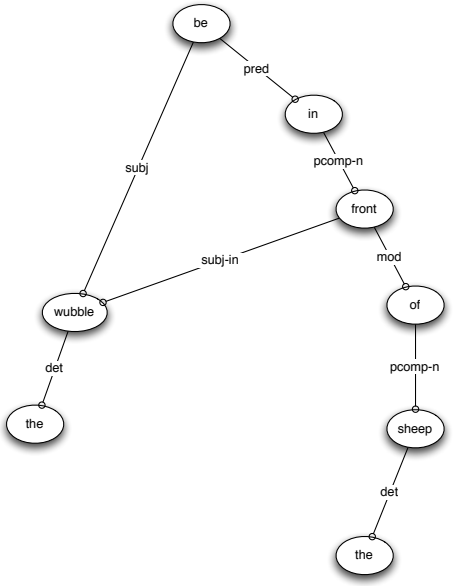


Figure 9: Dependency parse of sentence "The wubble is in front of the sheep".

World, represented as graphs, that are roughly aligned temporally with the sentences.

If word meanings are to be learned associatively, subgraphs of semantic graphs must be associated with words or phrases, that is, with subgraphs of dependency parse trees. Several factors make associative learning hard: The temporal alignment between semantic and linguistic graphs is imperfect. A sentence generally refers to only a small part of a scene. And sometimes sentences refer to nothing observable, such as mental states. Although we envision solutions to all these problems, none is likely to work unless we can first tame the combinatorics of associative learning. Semantic and linguistic graphs like Figures 8 and 9 each have many subgraphs and these may be paired in many ways. There are just too many hypotheses about what a word or phrase might denote.

Some progress has been made toward reducing the combinatorics of associative learning. The basic idea is to learn abstractions of the elements of semantic and linguistic graphs, then learn associations between these abstractions. For instance, *animate object* is an abstraction of **sheep1**, **sheep2**, and **demon**, just as *noun* is an abstraction of the words "sheep" and "wubble." A simple form of distributional clustering on graphs seems sufficient to generate these abstractions: Each node in a graph has a distribution of relations that connect it to other nodes. By looking at all occurrences of a node in many graphs, one gets a probability distribution over the relations it can have to other nodes. These probability distributions can be compared with the Kullback-Liebler distance, which tells us how similar two nodes are in terms of the relations they enter into. Given a similarity metric, it is trivial to cluster nodes; we use k-means clustering.

Clusters of nodes from semantic and linguistic graphs are shown in Figure 10. Clusters of nodes from semantic graphs are shown on the right side of the figure, whereas clusters of nodes from lin-

guistic graphs (i.e., dependency parse trees) are shown on the left. The arrows connecting these clusters were drawn by hand. Note, however, that the combinatorics of the associative learning problem has been reduced by this clustering heuristic: Instead of trying to associate the word "blue" with every entity in Wubble World, we can consider associating the cluster of words $\{blue, red, purple, of\}$ with clusters of entities, one of which is the colors $\{Blue, Green, Red, Purple, Yellow, Orange\}$. There is still a combinatorial problem to be solved, but it is much less severe.

The combinatorics can be reduced further by replacing the nodes in either a semantic graph or a linguistic one by the names of the clusters to which those semantic entities or words belong. This idea is shown in Figures 11 and 12, respectively. The numbers in the nodes are cluster identifiers. One can see that the number of ways of associating subgraphs in the semantic and linguistic graphs, while still large, is considerably smaller.

Our plan is to develop an associative learning algorithm for graphs like these.

7.2 Fluent-based representations

Verbs generally refer to activities that extend over time, and often refer to composites of activities. Thus, "retrieve x" has a plan-like semantics comprising activities such as "move to x," "grasp x," and "return with x." Although the semantic graphs discussed in the previous section contain predicates that represent activity (e.g., **SwiftAway(demon,sheep1)**) it is the extended truth or falsity of these predicates over time — over a sequence of semantic graphs — and particularly the temporal relationships between when these predicates become true or false, that represents the meaning of verbs.

One can organize sequences of propositional data in an array in which each row represents a logical proposition, each column represents a moment in time, and each cell contains 1 or 0 depending on whether the corresponding proposition is true or false at that moment. For example, input data such as

```
move(W)      1111111111111
push(W,D)    0000111100000
```

represents an agent, W, moving for an interval during which W pushes an object D. Following McCarthy [12], we call the interval during which a proposition is true a *fluent*. The input data above contains two fluents in an *Allen relation* called *during* [1]. The Allen relations describe all the ways that two fluents in relatively close temporal proximity can co-occur (see Table 7.2).

We developed an algorithm that discovers patterns made of Allen relations between possibly very large numbers of fluents. Following Allen, we might describe the above data as an *Allen sentence*

$$(\texttt{move(W)}\ \textit{during}\ \texttt{push(W,D)})$$

Alternatively, we might represent it with a *bit pattern* (BP), obtained from the data by removing all but one column in each run of identical, consecutive columns (see Fig. 13):

```
move(W)      111
push(W,D)    010
```

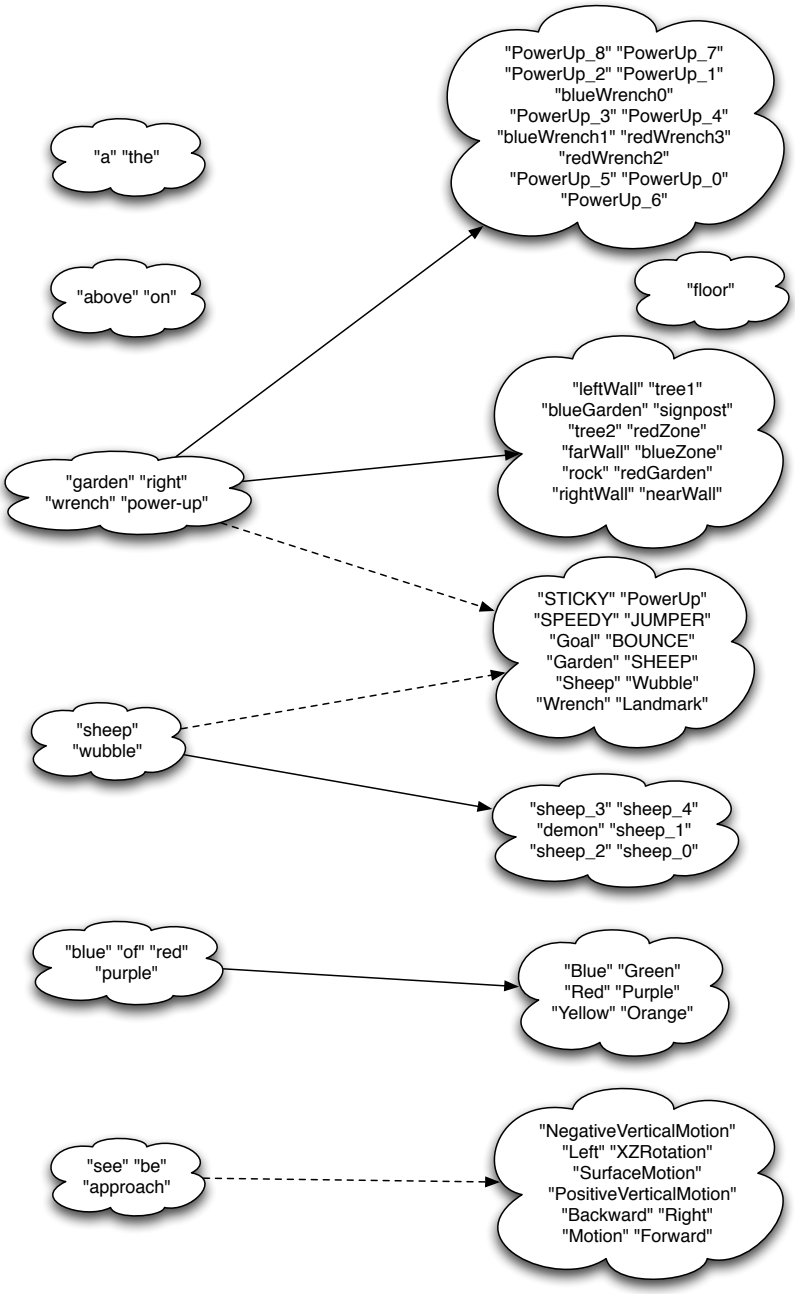


Figure 10: Sentence-scene cluster mapping (by hand).

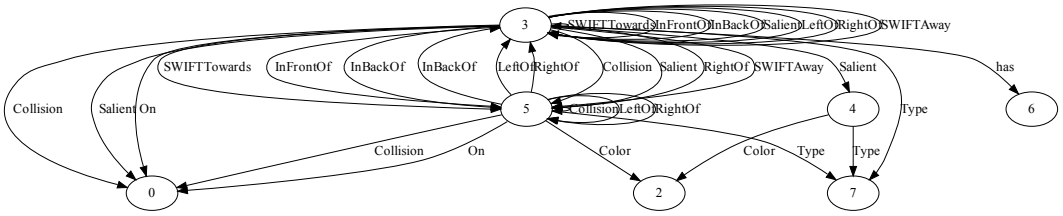


Figure 11: Semantic cluster graph for a snapshot of a scene during the Sheep game. All the nodes in the graph in Figure 8 have been replaced by identifiers associated with the entity clusters in Figure 10.

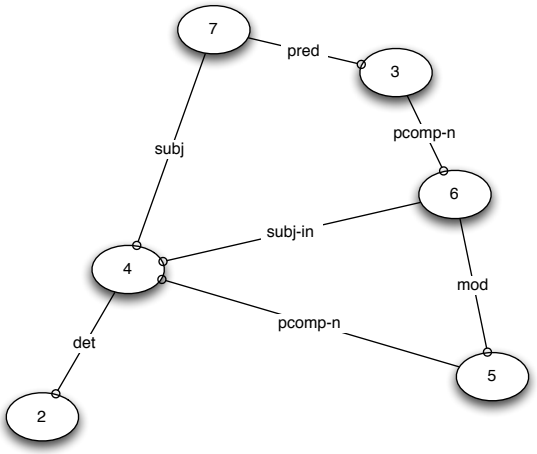


Figure 12: Linguistic cluster graph of sentence "The wubble is in front of the sheep". All the nodes in the graph in Figure 9 have been replaced by identifiers associated with the word clusters in Figure 10.

When only two fluents are involved, BPs and Allen sentences contain identical information. Both say that `move(W)` starts before and ends after `push(W,D)`. Both elide information about durations, and for this reason, the mapping from the input data to either BPs or Allen sentences is many-to-one.

However, when three or more fluents are involved, the mapping of input data to Allen sentences is many-to-many. To illustrate, imagine the agent, W, making contact with an immovable object such as a wall at the same time that `move(W)` ends. This pattern might be described with the Allen sentence:

$$((\texttt{move(W)}\ \textit{during}\ \texttt{push(W,D)})\ \textit{meets}\ \texttt{contact(W,Wall)}) \tag{1}$$

Alternatively, we might describe the same pattern as

$$((\texttt{move(W)}\ \textit{meets}\ \texttt{contact(W,Wall)})\ \textit{during}\ \texttt{push(W,D)}) \tag{2}$$

However, the pattern has only one BP representation:

```
move(W)            1110
push(W,D)          0100
contact(W,Wall)    0001
```

Generally, one can construct several Allen sentences to describe a set of input data, whereas any set of input data can only be represented by a single unique BP. Moreover, some Allen sentences lose information that is available in BPs. For example, Sentence 2, above, admits the possibility that `push(W,D)` happens during `contact(W,Wall)`, which is ruled out by the corresponding BP. (Many authors have noted that a composition of Allen relations — an Allen sentence — does not pin down the relations that hold between all its constituents [1, 2, 18, 8].)

Both of these observations create difficulties for algorithms that *learn* patterns composed of Allen relations, such as the Fluent Learning algorithm [4, 3] and algorithm proposed in [9]. When a single set of input data gives rise to more than one Allen sentence, the statistical evidence for the bit pattern contained in that data will be distributed over several sentences and diluted. One can try to canonicalize the Allen sentences so that exactly one sentence represents each pattern, but if the learning algorithm is iterative, this is hard to do. For example, if the algorithm first learns ($\texttt{move(W)}$ *during* $\texttt{push(W,D)}$), it might later learn Sentence 1; and if it first learns ($\texttt{move(W)}$ *meets* $\texttt{contact(W,Wall)}$) it might later learn Sentence 2. These sentences are *not* identical in their entailments and thus should not be reconciled to one canonical form.

This paper describes an algorithm for learning BPs that canonicalize Allen sentences.

7.2.1 Definitions

Propositional data may be organized in a bit array, as described earlier. The runs of "1" in row f of the array are called *instances of fluent* f. Instances of two fluents, f_1 and f_2, might be in an Allen relation (Table 7.2); for example, some instances of f_2 might *overlap* instances of f_1. All instances of *overlap*, irrespective of their durations, have a common form: There is an interval during which the proposition represented by fluent f_1 is true (f_1 is "on") and the proposition represented by

Table 1: Temporal relations between fluents as defined by [1].

BPP	Allen	Meaning
(1) (1)	equal	x starts and ends with y
(10) (01)	meets	x ends and y starts
(11) (01)	finishes	y starts after and ends with x
(11) (10)	starts	y starts with x and finishes before x finishes
(111) (010)	during	y starts after and finishes before x
(110) (011)	overlap	y starts after x starts and finishes after x finishes
(100) (001)	before	y starts after x finishes with small delay

fluent f_2 is false (f_2 is "off"), followed by an interval where both are true, followed by an interval when only f_2 is true. The bit pattern prototype (BPP) for *overlap*, shown in the first column of Table 7.2, represents this pattern of events.

A BPP that specifies the fluents it relates is called a bit pattern (BP). Instances of fluent f_1 and f_2 that are in an Allen relation are easily mapped to a BP by discarding all but the first of a run of identical consecutive columns in the bit array, as shown in Figure 13. This procedure yields bit patterns of any number of fluents. The *order* of a BP is the size of the set of fluents described by the BP, and we'll often refer to this set of fluents as being *in* the BP.

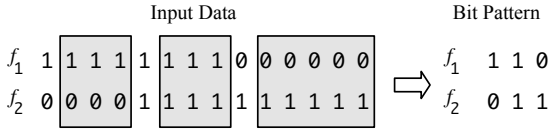


Figure 13: Discarding all but one of identical columns (shown shaded) in a bit array produces a bit pattern (BP).

7.2.2 Significance Testing

Our algorithm learns BPs when their elements co-occur significantly more often than would be expected by chance. Consider an instance of the BP called *meets* in Table 7.2:

```
coffee   10
jitters  01
```
(3)

Table 2: A contingency table for significance testing of a BP.

10 01	$jitters$	$\neg jitters$	
$coffee$	a	b	$(a+b)$
$\neg coffee$	c	d	$(c+d)$
	$(a+c)$	$(b+d)$	n

That is, jitters begin after the coffee is finished. Tests of significance of BPs rely on contingency tables like the one in Table 2. Here n is the total number of instances of the BP *meets*, and a is the number of instances of the BP $meets(\texttt{coffee}, \texttt{jitters})$. The row margin, $(a+b)$, is the number of grounded instances of the form $meets(\texttt{coffee}, y)$, and the column margin, $(a+c)$, is the number of grounded instances of the form $meets(x, \texttt{jitters})$, where x and y are other fluents.

For a BP to be worth learning, the quantity a should be bigger than b, that is, one should get the jitters more often than not after drinking coffee. Suppose this is true, so $a = kb$ for some $k > 1$. If the relative frequency of jitters is no different after one drinks, say, water, or rides one's bike down a busy street in Los Angeles (e.g., if $c \approx kd$) then clearly there's no special relationship between coffee and jitters. For a BP to be worth learning, we'd want $a = kb$ and $c = md$, and $k \gg m$.

This notion of "worth learning" can be tested with the χ^2 statistic for a 2 x 2 contingency table:

$$\chi^2 = \frac{n(ad - bc)^2}{(a+b)(c+d)(b+d)(a+c)} \tag{4}$$

This statistic has a chi-squared distribution, and χ^2 is more likely to be statistically significant as k becomes greater than m, where $k = a/b$ and $m = c/d$. (One can also get a significant χ^2 statistic when $k \ll m$, but we disregard these cases, which indicate significant relationships between *non*-occurrences of fluents.)

7.2.3 Algorithm

The number of potentially significant bit patterns in a dataset increases combinatorially with the number of propositions in the bit array, so no exhaustive generate-and-test procedure will scale up. This section describes a BP-learning algorithm and a greedy heuristic that directs the algorithm's efforts to learning larger BPs.

The algorithm operates iteratively over a set of training instances $\{T\}$, identifying successively larger BPs by finding statistically significant Allen relations (represented as BPPs) among smaller, previously learned BPs. Each training instance, T_i, is an interval during which a set of instances of fluents is observed. BPs represent Allen relations that hold between the fluents in T_i. We will refer to individual fluents as trivial or unary BPs. The *status* of a BP may be *focal* or *candidate*: Focal BPs are statistically significant, as described in Section 7.2.2. The algorithm will consider only these and unary BPs (i.e., individual fluents) for forming new, larger BPs. Candidate BPs

are compositions of two focal BPs that have been observed in the training data but are not yet statistically significant.

At each iteration i, a list of focal BPs G_i and a set of contingency tables (one for each candidate BP) are maintained. G_1 is the set of all unary BPs (all fluents). Given training instance T_i, the algorithm considers each case in the data where two focal BPs occur in close temporal proximity. For T_1, this corresponds to all cases where pairs of fluents occur near each other. In each of these cases, a BP is generated from the data, using all the fluents in the two contributing focal BPs, as described in Section 7.2.1. This generated BP may be either a candidate BP or a focal BP. Its contingency table is then updated, or if it has not been observed before, the table is created. Significance tests are then applied to all the contingency tables, and G_{i+1} is updated to include any new BPs that are found to be significant. Conversely, any focal BP in G_i that is no longer significant is removed.

Updating contingency tables Consider the case where the algorithm has just observed a candidate BP $g*$ composed of two focal BPs g_i and g_j of order m and n respectively. The algorithm must then update all contingency tables that are affected by this observation. If g_i and g_j are fluents, then the updates are simple: Rather than storing and updating full contingency tables for the various possible pairs of fluents that can occur in the BPP relationship specified in this candidate BP, the algorithm updates the four numbers corresponding to the *contingency table counts* a, $a+b$, $a+c$, and n in Table 2.

This scheme is sufficient when $|G_i|$ contains only degenerate BPs. However, with higher-order BPs, the algorithm must also consider the different ways of composing the candidate BP $g*$ of order $m+n$ from lower-order focal BPs. For example, there may exist two focal BPs of order $m-1$ and $n+1$ that can also combine to form $g*$. The number of different ways of combining BPs to create $g*$ equals the number of binary partitions of the set of fluents in $g*$, where either partition is empty. This results in a worst case $2^{m+n-1}-1$ updates, where each update corresponds to incrementing the four contingency table counts, as described above.

A Greedy Heuristic Although the number of updates increases dramatically as the order of the significant BPs in G_i increases, two factors help to keep the algorithm tractable. First, the number of significant higher-order BPs is typically small. Second, a heuristic limits the number of BPs used to perform updates during a training instance: Rather than performing updates for all BPs $g_i \in G_i$ the heuristic focuses attention on occurrences of the *highest order* BPPs, that is, the bit pattern prototypes that describe temporal relationships among the largest number of fluents. In a sense, these are the BPs most "worth learning" since we expect complex relationships among large numbers of fluents to be much more rare than relationships among a small number of fluents. If the algorithm's goal is to find the largest BP that is evident in the data, this is a reasonable heuristic since it focuses on finding statistically significant relationships between the larger BPs in G_i before trying to find relationships between small BPs.

Each training instance T_i is treated as a set of fluent intervals $f(a,b)$, where (a,b) is the interval in which the fluent is on. The heuristic works by greedily tiling the focal BPs in G_i onto this set of fluents, using the largest BPs first. A BP "covers" a set of fluents if that set of fluents is described

by the BP. Once a fluent is covered by a BP, the heuristic prevents the fluent from being covered again by a smaller BP. Thus, instead of considering a combinatorially large number of BPs given the set of fluents in T_i, the algorithm considers a maximum of $|T_i|$ *tiled* BPs. These tiled BPs are the only ones that can participate in combining to form larger candidate BPs. The maximum of $|T_i|$ tiled BPs only occurs if all the focal BPs are unary (i.e., individual fluents). As the algorithm adds larger BPs to the set of focal BPs G_i, this number of tiled BPs rapidly decreases.

7.2.4 Experiments with BP Learning

We ran an experiment to learn whether the BP learning algorithm could learn complex patterns corresponding to activities in Wubble World. We focused on jumping over blocks and jumping on blocks.We specified *target* BPs for each activity. The target for jump-over is shown in Figure 15.

One challenge for the algorithm is that most of the fluents in Wubble World should not enter into the learned representations of these activities. We recorded 171 distinct propositions (fluents) at each moment in this experiment, roughly 10% of which are part of the meaning of jump over or jump on. It is quite possible that some of these irrelevant propositions will be added to the patterns we want the algorithm to learn.

By manually controlling the wubbles, a sequence of 59 unique instances of jumping over blocks was generated in one long session. Then a session comprising 59 jump-on activities was generated. Before and after each jump-over (or jump-on) instance, we marked instance boundaries in the sequence, then moved the wubble somewhere else in its environment, and then jumped over (or on) another object. The intervals between the instance boundaries became our training instances. A sequence of 236 training instances for each activity was generated by appending the sequence of 59 instances to itself three times. We used the first 200 items in each sequence to train the learning algorithm.

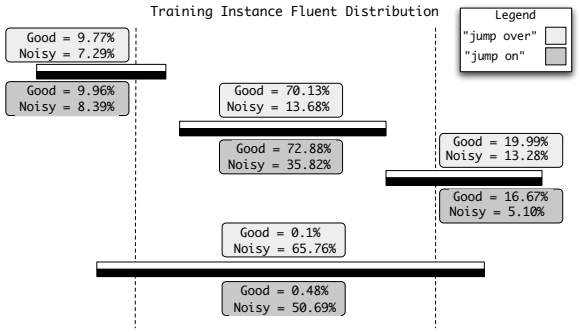


Figure 14: The distribution of fluents participating in the training datasets. The dotted lines represent training instance boundaries and the boxes represent the fluent intervals interactions with the boundaries.

The distribution of irrelevant fluents with respect to training instance boundaries in the jump-over and jump-on datasets is shown in Figure 14. Because the wubble moved around its environment in a continuous session, while training instances were particular intervals of the session, some fluents

were already on at the beginning of a training instance, some turned on and off during a training instance, some turned on during and remained on after the end of the training instance, and some turned on before the training instance and remained on throughout. The latter fluents were removed from the training instances because fluents that do not turn on or off during training instances do not provide any information to the learning algorithm. One can see in Figure 14 that 65.76% of the fluents that do not belong in the jump-over target BP were of this latter type. Conversely, 34% of the 153 fluents that do not belong in jump-over — or 52 fluents — turned on or off during training instances, and so could be incorrectly incorporated into learned BPs.

The algorithm examined 150,394 BPs and selected 231 of them as signficant BPs. Among these were the target BPs for jump-over and jump-on. It resisted including noise fluents into higher-order BPs. Figure 16 shows the composition of learned BPs of different orders. One sees that low-order learned BPs contain noise fluents, but higher-order fluents do not. Learned BPs of order 8 or lower comprise correct and noise fluents, but higher order BPs comprise only correct fluents.

The largest significant BP the algorithm learned matches the target in Figure 15. First W (the wubble) approaches D (a block) on the floor, next it jumps, moves to a position that is above D, and finally W ends up back on the ground with D behind it.

jump over	
InBackOf(W,D)	----------11111111
NegativeVerticalMotion(W)	----------1111-----
Away(W,ceiling)	---------11111-----
Towards(W,floor)	---------11111-----
Away(W,D)	------11----1111111
Below(D,W)	------11111--------
Above(W,D)	------11111--------
PositiveVerticalMotion(W)	----111-------11---
Away(W,floor)	----1111-------11--
Towards(W,ceiling)	----1111------111--
Jump(W)	---11--------------
Towards(W,D)	--1111---111-------
SurfaceMotion(W)	--1111111111111111
Motion(W)	--1111111111111111
Forward(W)	-11111111111111111
Collision(W,floor)	11111--------111--1
On(W,floor)	11111--------111--1
InFrontOf(W,D)	111111-------------

Figure 15: Target *jump over* grounded BPP.

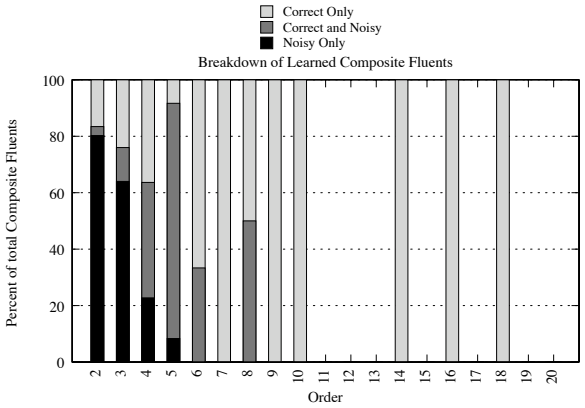


Figure 16: The distribution of BPs for the *jump over* dataset. All significant grounded BPPs are composed of fluents from the correct significant grounded BPP, a mixture of correct and noisy fluents, or all fluents are actual noisy fluents.

7.2.5 Discussion of BP Learning

We assert that the bit pattern in Figure 15 is a semantic representation of the wubble (denoted W) jumping over a block (denoted D). It contains much of the information one would like to have to answer questions about jumping: "When you jump, do you move toward the ceiling or the floor?" (Trick question: You move toward the ceiling and *then* you move toward the floor) "If you are below something before you jump, can you be above it later?" And so on. One can play the bit pattern in Figure 15 like a movie to get a kind of mental simulation of what it means to jump over something.

Note that the bit patterns for jump over and jump on were *learned* by the BP learning algorithm. This is important because, if the Wubble World project is to thrive, we cannot provide semantic representations of all possible words *a priori*, so the ability to learn them is essential.

References

[1] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] Thomas A Alspaugh. Software support for calculations in allen's interval algebra. *UCI-ISR-05-02*, 2005.

[3] Paul Cohen. Fluent learning: Elucidating the structure of episodes. *Advances in Intelligent Data Analysis*, 2189:268–277, 2001.

[4] Paul Cohen, Charles Sutton, and Brendan Burns. Learning effects of robot actions using temporal associations. *The 2nd International Conference on Development and Learning*, pages 96–101, 2002.

[5] Paul R. Cohen and Tim Oates. A dynamical basis for the semantic content of verbs. In *Working Notes of the AAAI-98 Workshop on The Grounding of Word Meaning: Data and Models*, pages 5–8, 1998.

[6] E. D. DeJong. The development of a lexicon based on behavior. In *Proceedings of the Tenth Dutch Conference on Artificial Intelligence*, 1998.

[7] D Dennett. Can machines think? *Alan Turing: Life and Legacy of a Great Thinker*, Jan 2004.

[8] Rosella Gennari. Temporal reasoning and constraint programming a survey. *CWI Quarterly*, 11(2,3):163–214, 1998.

[9] P. Kam and A. Fu. Discovering temporal patterns for interval-based events. *Data Warehousing and Knowledge Discovery*, 1874:317–326, 2000.

[10] Wesley Kerr, Shane Hoversten, Daniel Hewlett, Paul Cohen, and Yu-Han Chang. Learning in wubble world. *International Conference on Development and Learning*, 2007.

[11] S Kullback and R Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, Jan 1951.

[12] J McCarthy. Situations, actions, and causal laws. *Stanford University*, 1963.

[13] Tim Oates. PERUSE: An unsupervised algorithm for finding recurring patterns in time series. In *Proceedings of the IEEE International Conference on Data Mining*, pages 330 – 337, 2002.

[14] Tim Oates. Grounding word meanings in sensor data: Dealing with referential uncertainty. In *Proceedings of the HLT-NAACL-2003 workshop on Learning Word Meaning From Non-Linguistic Data*, 2003.

[15] T. Regier. A model of the human capacity for categorizing spatial relationships. *Cognitive Linguistics*, 6(1):63–88, 1995.

[16] Terry Regier. *The Human Semantic Potential*. The MIT Press, 1996.

[17] Deb Roy. *Learning Words from Sights and Sounds: a Computational Model*. PhD thesis, MIT, 1999.

[18] Timothy K Shih, Anthony Y Chang, Hwei-Jen Lin, Shwu-Huey Yen, and Chuan-Feng Chiu. Interval algebra for spatio-temporal composition of distributedmultimedia objects. *International Conference on Parallel and Distributed Systems*, pages 308–315, 1998.

[19] J. M. Siskind. A computational study of cross-situational techniques for learning word-to-meaning mappings. *Cognition*, 61(1–2):39–91, 1996.

[20] Luc Steels. Emergent adaptive lexicons. In *Proceedings of the Fourth International Conference on Simulating Adaptive Behavior*, 1996.

[21] Luc Steels. Perceptually grounded meaning creation. In *Proceedings of the International Conference on Multi-Agent Systems*, 1996.

[22] C. A. Thompson. *Automatic Construction of Semantic Lexicons for Learning Natural Language Interfaces*. PhD thesis, University of Texas, Austin, 1998.