A Memory Based Method for Inventing Features

Christer Johansson
Bergen University, Norway
{christer.johansson}@lili.uib.no

Abstract. Lexical features are usually not found in the surface forms alone. Such features are used in many practical applications, and theories. This article shows that it is fairly easy to automatically invent lexical features from syntagmatic relations, without calculating any statistics of co-occurrence. Apart from being a plausible model of lexical acquisition, this model can be implemented efficiently.

1 Introduction

This article will discuss a possible way of automatically inventing a lexical representation from exposure to sequences of lexical units. We will show how a simple instance based mechanism can easily be extended with a mechanism that allows generalization, and at the same time can exploit information from individual exemplars. Results will be shown for recognition, prediction, and separation of regular input from word salad (random sequences of words). Examples of the quality of representation will be given.

We will show, for a rather small test problem, that the acquired representations aid separation of grammatical sequences from word salad. The acquired representations additionally aid more accurate predictions of following words. Finally we will show that high quality representations arise in another popular test problem [13], but without training for any specific task. Machine learning algorithms have been extended with the capability of creating and changing representations before [13, 9, 20, 3, 8]. The first three were based on a hybrid neural network architecture, allowing for a symbolic lexicon in a connectionist model. The last two use instance based learning. Recently, instance based learners have become increasingly popular: in psychology of language [21, 6], theories of automatization [12, 10], computational linguistics [22, 2], and in modeling natural language processing [7, 18]. Similarity between exemplars plays a central role in all of these areas. The aim of feature creation is to go from representing the existence of each word to having a representation for each word. This makes it possible to compare each word to all other words. There are many applications where we would like to motivate a regularity with the fact that we have observed *similar* words that fit the pattern, even if those words are very different in form.

Defining a family resemblance, or similarity, is inherently difficult. The similarity only exists when we compare two items, but we are interested in finding the best matches of all the possible matches stored in the database of previous inputs (experience). This demands extensive search, which may be performed efficiently by a cluster of computers. We may think of similarity in many ways. It could be based on immediate measures of form (e.g., characters, or sound), it could be based on elaborate encoding of exemplars for syntax and semantics, and it could be based on co-occurrence of word forms [11, 8]. This paper presents a general task-independent mechanism for feature creation. The mechanism creates features by changing an initially random representation. The changes are motivated by how input is matched to previous experience.

Many computer models of natural language processing assume a rich representation of lexical features. It is rarely discussed where these features come from, or if it is possible to acquire such features automatically [15]. Recently there has been an increased research effort in this area, especially focussed around *Lexical Space* and that the meaning of a word depends on the company it keeps [11, 4, 17, 3, 16]. The approach in this article is different. This article suggests a method to automatically invent abstract representations. We may find the closeness of any two words by a geometric measure of distance (see section 2.1) between the representational vectors of the two words. This is done *without explicitly storing or computing* which words co-occur in the same contexts. A significant and novel feature of this article is that no statistic of co-occurence is calculated—representations are manipulated directly. The created features have binary values for further processing by, for example, neural networks. Every word sharing a feature value are similar along that feature dimension. If two words share more feature dimensions than expected they are said to be more similar than expected. All values are initialized randomly to either zero or one. This is a sufficient, as well as an efficient, way to represent the existence and the identity of the word. We may easily control the risk that two or more unrelated words will have the same representations [1, 5] simply by adding more features.

The general idea of this paper is related to a well known technique, which is used by most serious spell checking software: Bloom-filters [1, 5] can efficiently represent that a word exists. A number of hash codes, for example 10, are calculated for each word, each code points to a specific bit being set, in a very large vector initially filled with zeroes. This vector is called the (existential) dictionary. When half of the bits in the dictionary are on, we have a $(1/2)^{10}$ risk that a novel word, which has not been stored in the dictionary, has all its ten bits turned on in the dictionary as well. If we want to lower that risk, we can simply set more bits and have a larger dictionary size. The same idea is used in this work. Each word has a signature of 50 bits, of which initially half are on-bits. The risk that two different words will have the same code is initially $(1/2)^{50}$. Yet another use of the technique is *document signatures* [19]. We store each word in a five word context. Thus we have 250 $(5 * 50)$ features that can match. For any such instance we select the stored instance with the most matching features (i.e., bit values). If all features match, the initial risk that we have retrieved an instance different from the input is $(1/2)^{250}$ and finding an instance with 1 mismatch is 250 times more common. Two mismatches is $250 * 249$ times more common, but it is still a very small risk. The mechanism takes a representation of identity, and adjusts it in such a way that words with shared contexts will also share more features.

The core engine of the mechanism is a memory based k-NN (k nearest neighbours) model [14], with two levels of similarity: the feature and the word level. The value of the lexical features is changed by comparing the input to the retrieved nearest neighbour for that input. We use a simple strategy: mismatching features are much more likely to change into a match, than matches are to change into a mismatch. Both creating matches and introducing variation by creating mismatches are important for successful feature creation. The process is reminiscent of genetic adaptation and mutation.

It is important that the sequences of words stay the same when features change. We do not want to forget patterns, as we learn which words are similar. Features and words must be able to change independently of each other. In other words, we want to separate *experience* from *generalization*.

The particular model is actually a 5,1-NN model (see figure 1). Each instance contains five temporal positions where a word can be found. One best context is retrieved for each of these positions. Using a simple window technique creates five instances for each word as input flows. A database is searched to find one closest match for each instance. Results for each word (i.e., five responses, see section 2.1) are integrated to find the best response to each word based on the preceding and succeeding word contexts. Each word in the input stream is treated as an index to a representation of lexical similarity stored in a separate lexicon, just as in previous works [13, inter al.]. Each new word is represented by random binary features. The best length of initial representations depends on the application. Fifty features were used in this paper.

First a glance at the model in more detail. The model will be described in terms of two memory systems: short and long term memory. The definition of the model rests on the architectural constraints we have between memory systems. Nothing can be stored in long term memory before having passed short term memory. The short term memory perform a task of categorizing input into instances. Access to long term memory is restricted by a lexicon lookup for features to compare.

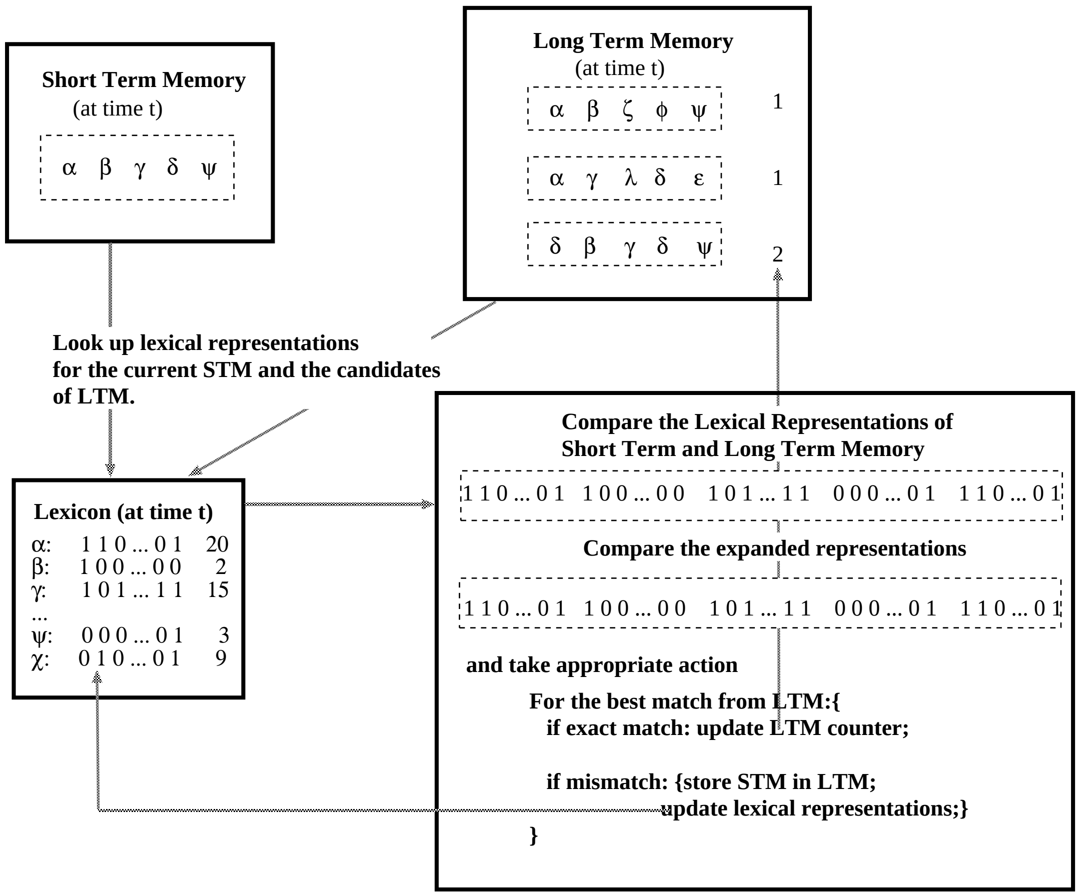

Figure 1: An outline of the mechanism

2 The Model

In this section the learning model will be explained in more detail. The reader can see an outline in figure 1. The core of the proposed mechanism is to utilize interaction between Short Term and Long Term Memory systems (STM and LTM). Sr and Lr are the short and long term memory responses, which are different from the system that creates those responses. The short term memory response, Sr, can be thought of as a key to retrieval of similar experience from LTM. An Lr is always a previous Sr. It is hypothesized that direct copying of an Sr into LTM is prohibited. Each Sr will be stored as a possible Lr in LTM first after lexical lookup (see figure 1). The conceptual difference between the two types of responses is that Sr contains recent inputs, whereas Lr contains *retrieved* information. Sr is intended as a result of perceptual categorization of sensory input. One may think of it simply as a sequence of distinct words. Windowing is used to get the five most recent words into an Sr, which will then trigger a response (a *recognition* from LTM). If a response is accurate it could be used to trigger a chain of other automatic responses.

We will investigate a) how such a mechanism may *construct* its own representation of lexical similarity, b) how it may be used to *recognize* new input in terms of old experience, and c) how it may *predict* appropriate parts-of-speech at a considerable distance. The model is not using any external feedback on how well the task was solved. All the mentioned tasks are accomplished without retraining the representation of similarity. Retraining is essentially to construct a new representation to fit a new task. In a sense, the solution of tasks b and c are just side effects of feature invention, and automatic responses from the LTM store.

2.1 Deciding the best recognition for a word

Recognition is delayed until the input word has been seen at all temporal positions in the Short Term Memory. At this time we have gathered the same number of responses from LTM as the capacity of an Sr. The computer program has a special memory structure for storing the five most recent Lr. This structure contains LTM responses for each word at different positions in time, which is integrated into a recognition of the first word in the STM response, when position 0 is reached. Assume an instance size of five words. When a word reaches the 0-position in Sr it has been retrieved at five temporal positions $x : Lr[t]...Lr[t-4]$.

```
   The store of previous LTM responses:
  Lr[t-4]   1  2  3  4  x
  Lr[t-3]   2  3  4  x  6
  Lr[t-2]   3  4  x  6  7
  Lr[t-1]   4  x  6  7  8
      (best match to the previous Sr)
  Lr[ t ]   x  6  7  8  9
      (best match to the current Sr)
  Sr        0  1  2  3  4
      (actual input)
Candidates are marked by x
```

If a majority (> 2) of the five positions contains the same word then we have found the word we are going to use for x. If there is no clear majority for any of the words given at the five x positions, then the candidate word which is closest to the prototype of the five candidates must be given. If all candidates are equally close then we will give the word from Lr[t]. In case we lack a clear majority for any of the retrieved words, we pick the word closest to the prototype (a construction of the average, expected, value of each feature). A simple example, word a is represented, in a lexicon of three words, by $[0, 1]$, word b by $[1, 0]$ and c by $[1, 1]$. The prototype of these three words is the average vector: [$2/3$, $2/3$]. Obviously word c is closest to this prototype, with a *'city block'* distance of $2|(1 - 2/3)| = 2/3$, compared to 1 for word a and b. In case the prototype does not decide, we use whatever word was most recently retrieved in position x at $Lr[t]$. Sometimes we retrieve the same word that was given, but often we will retrieve a stand-in if the word has never been seen in that particular (or a similar enough) context before.

2.2 The Lexicon

The lexicon is an important memory structure, which is thought to deliver a representation of similarity for the symbolic currency of all positions in an Sr or Lr. Since the lexicon is separate from both LTM and STM, its contents may change independently of what is stored by LTM or STM. Each word is an index key to the lexicon, which delivers the features used for comparisons. A previously experienced Sr can always be retrieved accurately without the lexicon, but novel patterns will be less accurately recognized without a trained lexicon (see section 3).

2.3 Inventing a Lexical Representation

Some generalization is accomplished using only word identities. A good representation of similarity will enable the process to accurately recognize low frequency words by proxy. The algorithm starts with an array of binary features with initially random values (a random vector). First get a word from the input stream (punctuation counts as a word). Update the short term response Sr. Then select the best match (Lr) to the Sr (which keeps an instance of the last 5 words) from the long term storage LTM (which keeps all unique and previously processed instances). Update a memory of the last five retrieved instances. If Sr turns out to be a new pattern (i.e. something else was retrieved for it) then store it in LTM, and adjust the feature representation of *all* of the words that were retrieved as some other word. A feature is only adjusted if its word has been seen less than a threshold value (e.g., 25 times). Adjustment is done so that mismatching features are flipped with 50% chance. Two in a hundred matching features are flipped to keep close words apart. Note that a word which occurs once in the input has five positions in which its representation may be changed.

This procedure creates a drift towards more shared features for words that are often retrieved in similar lexical contexts. For example, when *"heard that the cat lives"* is retrieved for *"saw that the dog died"*, it will cause a random change of about half of the mismatching features of $\{$*saw, dog, died*$\}$ (in relation to their corresponding, retrieved words $\{$*heard, cat, lives*$\}$). Additionally, one in fifty *matching* features is changed to keep some variation in the representations. Low frequency words will more often be retrieved by proxy, which will make those words drift towards their high frequency family members. We expect about half of the features to match between any two lexical identities initially. This gives an expected penalty of 25 missed features (assuming a total of 50 features) for choosing a non-identical match, plus an extra penalty score for missing the identity of the word. When two words consistently match despite the penalty, it is likely that they share a surrounding context, which in turn is an indication they may be syntactically interchangeable.

One problem is to keep variation of feature values until the knowledge base has expanded. Two factors in the model accomplish this. First, feature values are initially random. Second, we will change matching features at an initially high rate, but rapidly dropping to a rate of two in a hundred. Fast feature creation is made possible by selecting the most similar instance, and only then change the match of features. A second reason for success is that similarity between Sr and Lr are stored in part by exemplars, and in part by lexical representations.

Let us see how this works for an imaginary word (at index 99), which has been retrieved as another word (at index 66). Say word 99 had the representation 0110. The retrieved word (66) has the representation 1110. The interesting representation for word 99, in relation to classification of the word in this context, is 1110 (not 0110). This could be useful. For example, we could train a generalizing mechanism (e.g. a backpropagation network) on this stand-in input. Such input provides a clearer signal as the retrieved items are likely more typical in that lexical environment. The candidates for training is always chosen from actual input, which avoids the problem of absurd compromising in neural networks. A well known example: A network is trained to turn left as many times as right. Normally, a network trained like this would compromise and drive straight through a fork in the road. With instance based pre-processing, the mechanism would first choose an appropriate instance (which could be either turn left or turn right), and thus avoid catastrophic behavior.

2.4 Generalizing the output

We have so far discussed the input as a sequence of words. However, it is often required that the output is a *reflection* on the input rather than a correct recognition of it. It may therefore be useful to supply each word in the input with a tag that will be displayed in place of that word. Depending on the task this tag could be directly derived from the word, or a tag that is relevant in that particular context. We will not allow that the tag is used for training. The retrieval of a tag will be kept incidental to finding the best instances. The benefit is that recognition, and prediction, can be achieved simultaneously without retraining representations. This is just a question of which tag is stored: If the tag is the word which is 5 words further in the input, then we have prediction of the fifth word to follow. It is likely that performance on a task could be better if we allowed feedback in training, but that would make the results depend on the task, and thus less general.

```
The store of LTM responses for tags:
    LTM_TAG[t-4]   1  2  3  4  x
    LTM_TAG[t-3]   2  3  4  x  6
    LTM_TAG[t-2]   3  4  x  6  7
    LTM_TAG[t-1]   4  x  6  7  8
    LTM_TAG[ t ]   x  6  7  8  9
```

We are interested in what was retrieved in the positions marked by x. Each of these positions have an associated tag (which could be the same word). If more than two x are the same we have a winner by simple majority, and the associated tag could be given directly. Otherwise, a winning word is selected from the five candidates, and its tag is given. The winning word is the most recent word that is also closest to the prototype (as in section 2.1).

2.5 Processing new cases

The automatically invented representations are loaded into LTM together with the exemplar database, and we are ready to test the mechanism on novel data. The ability to create new representations is now turned off.

The procedure is: Load the lexicon, which contains the invented representations. Load the training exemplars into memory. Read a word from the input stream. Form an instance (Sr) which can be used to retrieve the best matching instance (Lr) from memory. Update the memory for the five last retrieved instances. Select a tag as described previously, and display it. Repeat this procedure until all input is processed. For each new Sr, which may include unknown words as well as unknown word sequences, determine the Lr. The last five responses determine the output for a word in the input stream. Testing works as the previously described training algorithm, except that the lexical representations are not changed, and an output is displayed.

2.6 Reducing redundant features

The automatically created features tend to have redundant features that mostly overlap in values across the lexicon. Redundant features are automatically removed, before processing new cases, in order to create clearer representations. An additional benefit is faster execution of the program. The plan is to find the lowest number of features that separate exactly as many words as was originally separated. The rest of the features are redundant.

First sort all features on their discrimination value (which is highest if the feature splits the lexicon in two equal parts, and lowest if the feature is only associated with one word. The general principle is known as *information gain*). Keep picking more features in order of their discrimination, until the picked features discriminate just as many distinct words as the original set of all features. This is a fast approximation of Minimal Description Length. Print out the lexicon using the reduced number of features.

3 Results: Proof of Concept

We will present results on two tasks. The first task is to recognize grammatical input, while failing to recognize word salad. The data sets are based on a small artificial language that models various types of embedding [7, 8]. The training set consists of 5212 running words, with either 20 (*baseline, partial lexicon*) or 68 (*extended, full lexicon*) lexical units. The ’content’ words had 5 copies in the extended set (full lexicon), which were introduced with falling frequency of occurrence. The test set, consisting of unseen ’sentences,’ was copied 48 times, each time with one template content word replaced by a spawn. The test set thus contains 48 novel words out of a total of 68 words. The total size of the test set was slightly more than a quarter of a million words. The test word salad consisted of 660 randomly chosen baseline words. The training and test sets contained approximately equal proportions of simple sentences, as well as subject, object, and complex embedding [7]. In a typical newspaper corpus we would see much less of the embedded structures, and we would probably have a better coverage of the test material than 30% known words (20/68).

The second task is case role assignment using the available data sets of the fgrep-task [13]. The training set consists of 7190 presentations, and the test sets are 190 presentations in two conditions: familiar or unfamiliar sentences. The performance and ability to create features of our model is comparable to the original fgrep-task, but without explicit computation. One difference is that our invented representations consist of crisp binary features.

3.1 Word recognition and prediction

The first thing to notice in figure 2 is how well the grammatical sequences are separated from the word salad (*wordlist*), when we measure the rate of lexical classification, i.e. how often a word was recognized, or predicted, with the correct lexical class (e.g. plural noun).

The x-axis shows the distance of prediction. A distance of 0 equals pure recognition. At this position the word has been retrieved at five different temporal positions, and an integration of those results has eliminated some uncertainty. A distance of 1 is a prediction of the next word in a sequence. Words at a distance up to 4 have been presented in at least one temporal position in short term memory. Distances larger than 4 are true predictions of unseen words, i.e. these words was not part of the input.

Orthogonal representations were handcrafted to model lexical classes of words. The created representations are clearly close in performance to the handcrafted representation, and sometimes better. The lines marked with ’partial lexicon’ were not trained on all words, and all the novel words had different random representations. The lines marked with *w/o representation* shows the performance using only word forms. The important measure is the separation of regular input from word salad, which is best for the created representations.

The results are fairly clear. Representation increases both prediction and recognition in the grammatical condition, and decreases performance on word salad (i.e, word lists). The handcrafted representation maintains a good recognition of word salad, which should have been avoided. The horizontal lines in the figure are baselines for randomly picking a category based on the frequency of types or tokens. It is not only that the invented representation improves performance on grammatical regular input compared to the random baselines, but it also decreases recognition of ungrammatical input compared to the same baselines. Thus the invented representation enhances the contrast between regular and non-regular input, and this is done with unsupervised learning. It is encouraging to see that these results are valid for true predictions at quite large distances from the decision point. The mechanism does show some level of expectation on the input to come.

3.2 Performance on the FGREP-task

The performance on case assignment agrees with what is reported by Miikkulainen and Dyer. The important issue is if the invented representations hold a good quality. Figure 3 shows that the ambiguous words *bat* and *chicken* are similar to *breakers*, and *food* respectively, but both are similar to other *animals* as well—just as in fgrep. The word *girl* ends up being different from both *boy* and *woman*, also in accordance with the original version of fgrep.

The representations have a fairly apparent structure, where *living* entities, and *things* are clearly separated. The function words have fairly distinct signatures from the rest. We can see high levels of similarity within some semantic groups present in the data set: *human* (man, woman, boy, (girl)), *animal* (wolf, lion, sheep, dog, bat, (chicken)), *food* (chicken, cheese, pasta, carrot), *fragile* (plate, window, vase), *hitter* (bat, ball, hatchet, hammer, vase, paperweight, rock), *breaker* (bat, ball, hatchet, hammer, vase, paperweight, rock), *possession* (bat,

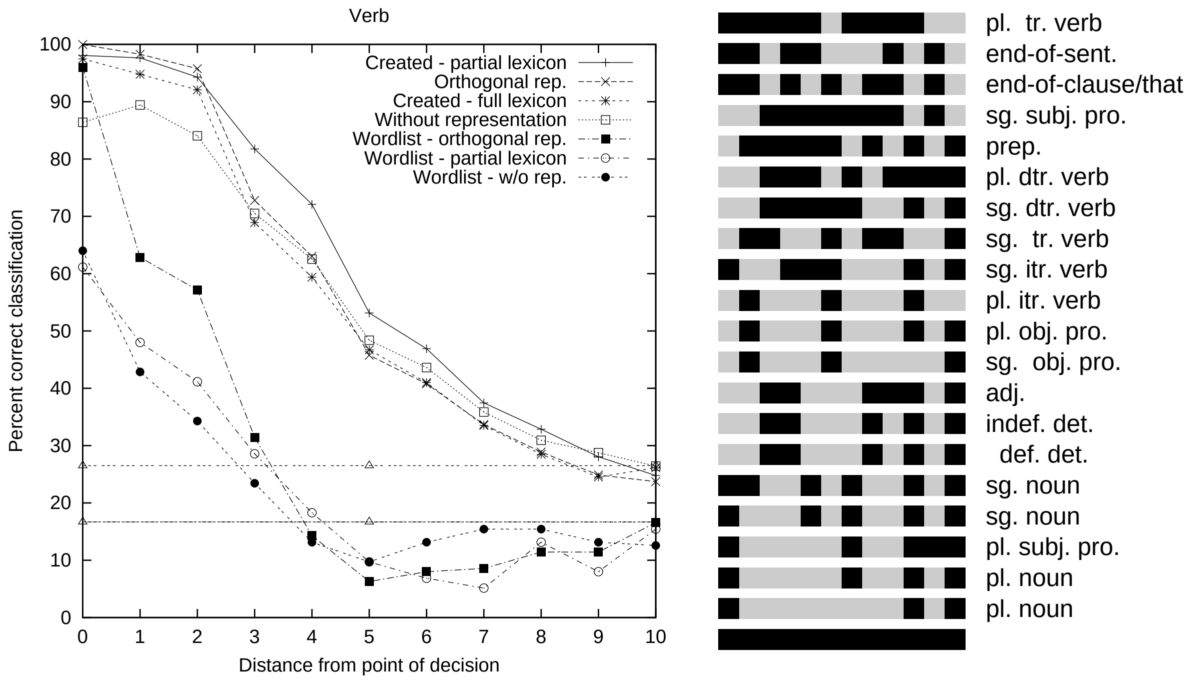

Figure 2: Recognition, and Prediction

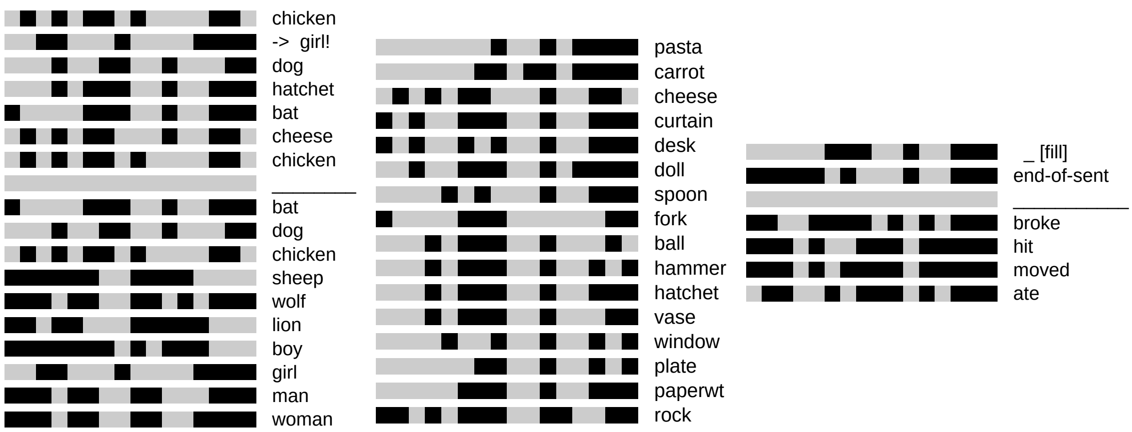

Figure 3: Automatically invented representations.

ball, hatchet, hammer, vase, dog, doll). There are some representations that are a little strange, for example *lion* and *boy* are more similar than *lion* and *wolf*. However, the individual features that are invented do not seem to have a clearly defined semantic meaning. The possible usefulness of such features is in relation to the retrieval mechanism, as the features together seem to implement fuzzy semantic boundaries. Different mechanism could weigh the features differently, which in essence would create different views into the knowledge base. If we want to implement a specific known task, we could accomplish a more optimal combination of features by means of supervised learning.

As Takahashi [20] noted, we would have liked to create a lexical entry for each version of ambiguous words, for example *chicken1* for the food, and *chicken2* for the living animal, or *bat1* for the animal, and *bat2* for the hitter. Automatic detection of polysemy and vagueness from sequences of words, without access to pre-classified exemplars, is a hard problem. The context helps to disambiguate, but this implies a model of the lexicon in which there are multiple entries for each word, depending on the context or situation in which the word appears. There are very common cases of true ambiguity where it is not possible to decide one exact meaning of an expression. In sentences like ”wolf ate chicken” we assume that chicken refers to the animal, but it is also possible, to some degree, that the wolf ate a dish, which might have been prepared and cooked by someone else. The database is not elaborate enough to separate these cases, but it should in principle be possible to evaluate the range of situations if more information was available. This necessitates a mechanism for evaluating each situation as it is stored in the knowledge base. For living beings such an evaluation system must be crucial for their perception of the world.

4 Discussion and conclusion

We have shown that the created features offer better performance on some tasks, and that, in an exemplar based model, performance goes down with the use of invented features for recognizing some types of ungrammatical input (word salad). Furthermore, we have shown that the created features allow for family resemblance between semantic classes of words in a well controlled task (fgrep).

It was *not* crucial for our mechanism to be trained on a task in order to create features. This is in sharp contrast to the fgrep-mechanism [13]. In fgrep it is crucial that there is a task to solve since the error feedback is used to create the features. Another contrast to our approach is that neural network mechanisms often begin with a fixed number of features to learn. In our approach we begin with a too large number of features, and eliminate redundant features at the end of the procedure. A reduction of the connections in neural networks are often compared to a developmental phase in brain development where neurons and neural connectivity are reduced. This is mirrored by a loss of features in our model.

Finally, it should be noted that our model could perform different tasks, without having its invented features retrained. It accomplished recognition by retrieving the most similar instances that could be retrieved from the memory store. It accomplished prediction by looking ahead in the database from the instances that were retrieved. We could say that the tasks were solved as a side effect of the feature training (and exemplar retrieval) rather than training for a specific task. It is possible to use other modes of machine learning to discover the weight of the invented features (and the combination of such features) for some specified task. For example, we could train another mechanism (e.g., a decision tree) on parts-of-speech tagging, using the invented features when we get access to data with the correct parts-of-speech tags. What our mechanism accomplishes for us is a percolation of information from co-occurrence into lexical representations, and we end up with richer representations of words. There is generous space for enrichment of the representations based on other sources of information, for example morphological form.

Acknowledgment This work has been conducted with financial support from the Norwegian Research Council (NFR) to the BREDT project. I thank Anders Nøklestad at Language, Logic, and Information (SLI) Oslo University, for valuable comments and discussion, Lars G. Johnsen for discussion and encouragement, and Jill Walker for many suggestions that helped me make the text readable. I also thank two reviewers for all their efforts, and kind advice.

References

[1] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[2] A. van den Bosch and W. Daelemans. Do not forget: Full memory in memory-based learning of word pronunciation. In D.M.W. Powers, editor, *proceedings of NeMLap3/CoNLL98*, pages 195–204, Sydney, Australia, 1998.

[3] A. van den Bosch. Expanding k-nn analogy with instance families. In *Skousen et. al. (02)*, pages 209–223, 2002.

[4] C. Burgess and K. Lund. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28:203– 208, 1996.

[5] E.T. Floyd. An Existential Dictionary. *Dr. Dobb's Journal of Software Tools*, 15(11):20, 22, 24, 28, 30, 32, 110–112, 1990.

[6] S.D. Goldinger. Echoes of echoes? An episodic theory of lexical access. *Psychological Review*, 105(2):251–279, 1998.

[7] C. Johansson. Noise resistance in processing center-embedded clauses: A question of representation? In *Proc. of ICCS'99*, pages 253–258, Tokyo, 1999. Waseda University.

[8] C. Johansson. The Hope for Analogical Categories. In *Skousen et. al. (02)*, pages 301–316, 2002.

[9] P. Kleiweg and J. Nerbonne. An FGREP investigation into phonotactics. In F. van Eynde, Schuurman. I., and N. Schelkens, editors, *CLIN'98*, volume 29 of *Language and Computers: Studies in Practical Linguistics*, pages 37–50, Amsterdam, the Netherlands, 1999. Rodopi.

[10] K. Lamberts. Flexible tuning in exemplar-based categorization. *J. of Experimental Psychology: Learning, Memory, and Cognition*, 20(5):1003–1021, 1994.

[11] T.K. Landauer, P.W. Foltz, and D. Laham. An introduction to Latent Semantic Analysis. *Discourse Processes*, 25:259–284, 1998.

[12] G. Logan. Towards an instance theory of automatization. *Psychological Review*, 95(4):492–527, 1988.

[13] R. Miikkulainen and M.G. Dyer. Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, 15(3):343–399, 1991.

[14] T.M. Mitchell. *Machine Learning*. McGraw-Hill Int. ed., Singapore, 1997.

[15] T.J. Palmeri. Formal models and feature creation. *Behavioral and Brain Sciences*, 21:33–34, 1998.

[16] M. Sahlgren. Vector-based Semantic Analysis: Representing word meanings based on random labels. In *ESSLI Workshop on Semantic Knowledge Acquistion and Categorization*, 2001.

[17] H. Schütze. Word Space. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann Publ., 1993.

[18] R. Skousen, D. Lonsdale, and D.B. Parkinson. *Analogical Modeling: An exemplar-based approach to language*, volume 10 of *Human Cognitive Processing*. John Benjamins, Amsterdam, the Netherlands, 2002.

[19] G.W Smith. *Computers and Human Language*. Oxford University Press, New York, 1991.

[20] N. Takahashi. Polysemous words and the FGREP algorithm. In Hitoshi Isahara and Qing Ma, editors, *Proc. of the 2nd Workshop on Natural Language Processing and Neural Networks*, pages 101–106, Tokyo, Japan, 2001.

[21] P.L. Tenpenny. Abstractionist versus episodic theories of repetition priming and word identification. *Psychonomic Bulletin & Review*, 2:339–363, 1995.

[22] J. Zavrel and W. Daelemans. Memory based learning: using similarity for smoothing. In *Proc. of ACL'97*, pages 436–443, Madrid, Spain, 1997.