

# An Association-Based Approach to Semantics

Ido Milstein                      Magnus Sandberg  
ido@cs.stanford.edu    sandberg@cs.stanford.edu

## 1 Introduction

This report describes the design and implementation of a prototype system for collecting, structuring and using semantic information derived from sentences of natural language. The main feature of the system is a mind map like *semantic network* consisting of *symbols* and *associations*. Standard methods of part-of-speech tagging and parsing, together with some additional processing, is used to extract semantic structure from sentences, information which is used to build and update the network. A *generalization mechanism* is responsible for adjusting the strengths of, as well as adding and removing, associations. Information can be retrieved from the network by means of a *query mechanism*.

The rest of this paper describes each of these features in detail, presents some initial results, and provides a discussion of the pros and cons of this approach to semantics.

## 2 Problem statement

### 2.1 Background

The ultimate objective of natural language processing (NLP) is to construct systems that can understand, in a non-trivial sense, natural language (NL) as spoken by humans. In order to reach this goal a number of problems have to be solved. Current NLP research is to a large extent focused on the task of obtaining a correct and adequate *syntactic* representation of NL text. Significant advances have been made in this area during the last decade, based mainly on statistical techniques [2].

A harder problem is that of *semantics*: finding the *meaning* of a text. Essentially, the question is how to obtain a semantic representation of a text given a syntactic representation (in reality the task is not strictly sequential, since some syntactic ambiguities can only be resolved given semantic understanding [2]). If we could let a computer solve this problem, the implications would be far-reaching in areas such as human-computer interaction and information processing.

One could argue that the problem of semantics is hard because the meaning of a text is not an intra-textual property but relates to the actual world (or, more generally, to some *domain*). Indeed, one possible approach to semantics is to try to map NL text to propositions of logic, the truth or falsity of which can be determined in the context of a pre-existing *model* of the world (see [3] for an introductory discussion). On the other hand, some linguists and philosophers (including, notably, Wittgenstein) have argued that the meaning of words and phrases is nothing more than how they are used. This would suggest that semantics should be approached with statistical techniques rather than with logic.

It is not the purpose of this paper to discuss the validity of the "use" theory of meaning; suffice it to say that the authors do not believe that understanding, in any interesting sense, can be created in a vacuum. Philosophical questions aside, we note that one can adopt a weaker sense of "understanding", in which a system is said to understand natural language (or some subset thereof) if it can

- accept sentences of natural language
- extract and store some of the information in these sentences
- answer queries related to this information
- give answers that are correct and make sense.

Systems with some of these capabilities exist. For example, there is an abundance of "chatterbots" on the web<sup>1</sup>, programs that can carry on a conversation with a human user. However, the chatterbots are usually very limited in terms of semantic processing, their focus being on syntactic correctness and the production of human-like dialogue. As another example we may consider question-answering services such as NeuroMedia's *Virtual Service Representative*<sup>2</sup>. This system handles items 3-4 on our list quite well. It is, however, not designed (or intended) to accept new information presented to it in NL (except conversation-level information such as the name of the user), but rather relies on a database created by some other means.

Thus, chatterbots and question-answering systems have in common a very limited capability for *semantic learning*, i.e. the learning of facts, presented in natural language, about some real or imaginary domain. In contrast, our intention has been to design a system capable of exactly that. On the other hand we have been willing to sacrifice syntactic versatility and conversational talent, properties which we believe are secondary to semantic processing. We have focused in particular on making a system that can learn *incrementally*: start with little or no knowledge about the world and acquire more gradually.

It is the belief of the authors that the logic-based approach to semantics, as briefly described above, has fundamental shortcomings in the context of semantic learning, and incremental learning in particular. Some of these, which are also the problems we would like our system to address, are:

- Logic-based systems are *static* in the sense that they deal with a fixed and sharply delimited domain.
- A logic-based system requires a lot of *initial knowledge* about its domain.
- Logic-based systems tend to be *inflexible*, in that they either solve a problem (answer a query) correctly, or cannot deal with it at all.
- Logic lacks the *associative nature* of human language and memory.

Of course, it is not inconceivable that the kinds of logics used today could be augmented in one way or the other, so as to make a logic-based system capable of incremental semantic learning. We notice, however, that although such a system, the Advice Taker, was proposed by McCarthy already in 1957 [4], no one has yet been able to come up with a feasible implementation.

The next section describes an alternative, and in our view more promising, approach.

## 2.2 The association model

In order to do semantics one needs a *world model* and a *language model*. The problem, as we see it, with the systems mentioned in the previous section, is that while they may feature fairly elaborate language models, their world models are inadequate, being either too simplistic (chatterbots), too rigid (logic-based systems), or entirely fixed (question-answering systems).

A more dynamic approach is to model the world as a *semantic network*<sup>3</sup>. Generally, a semantic network is a graph consisting of nodes (symbols) connected by edges (relations). The most obvious type of symbol is the one representing real-world objects or abstractions: *car*, *square*, *rhododendron*. However, symbols can equally well denote properties of objects (*green*, *voluptuous*), activities (*to work*, *to procrastinate*), or just about anything else you'd care to think of.

The real-world relationships between symbols are captured by the relations between them in the network. Typical attributes of relations are type (*car isa vehicle*, *wheel part-of car*), direction (a car is a vehicle, but a vehicle need not be a car), and strength (Peter *really* likes rhododendrons).

It should be pointed out that language is also a real-world phenomenon, the nature of which can be captured (at least to some extent) by a semantic network. That is, there can be symbols and relations for e.g. syntactic categories (*noun*) and relationships (modal *modifies* verb). This, in combination with the fact that symbols such as the ones described above can have NL words assigned to them, opens for the possibility of simultaneous learning of a language model and a model of the world.

---

<sup>1</sup>See for instance <http://www.student.toplinks.com/hp/sjlaven/index.htm>

<sup>2</sup><http://www.neuromedia.com>

<sup>3</sup>The term "semantic network" seems to be used in the literature to denote a wide variety of models. We proudly contribute to this confusion by adding our own definition.

An interesting application of semantic networks to the area of natural language processing is Microsoft's MindNet. MindNet is a "lexical knowledge base" structured as a semantic network, derived automatically from dictionary text [5]. Its most distinguishing feature is perhaps its set of no less than 24 types of relations, including *cause*, *time*, and *co-actor*. These types are deemed sufficient to express most, if not all, kinds of possible relationships between objects and concepts. Another important feature of MindNet is its use of a specialized "semantic parser" that extracts semantic structure from sentences, structure that is then used to build the network.

MindNet represents an impressive effort, and there is no doubt that a similar approach could be useful for many applications. Nonetheless, we have chosen to take a somewhat different path. The rich type approach to relations, while offering important advantages, also has its problems. First, even granting the possibility of creating a "canonical" set of relation types, it is not clear that it's really possible to extract enough semantic information from arbitrary sentences to be able to correctly instantiate these types in the network, unless you already have a system with full-fledged semantic understanding. Second, the large number of relations calls for a multitude of query mechanisms, each specialized to extract a particular kind of information from the network. Third, it seems difficult to create general procedures for generalizing and otherwise restructuring the knowledge encoded in the network.

In sharp contrast to the rich type set approach, we have chosen to work with type- and direction-less relations which we refer to as *associations*. An association is always a binary relation, i.e. it connects exactly two objects, and its only attribute is its *strength*, which is a number between 0 and 1. The main reason for this design is that it is the most general one: Any typed relation, regardless of arity, can be represented with associations by creating intermediate symbols<sup>4</sup>. This means that an association-based network imposes very little restrictions on what kind of data can be modeled. Essentially, we are taking a stance for variance and against bias. Given the extremely arbitrary nature of natural language (and the world), this seems like a reasonable choice.

The association model offers additional advantages. In particular, it addresses the shortcomings of MindNet mentioned above. Consider for instance the first problem, that of extracting enough information from a sentence to be able to update the network. If a relation between two symbols means nothing more than that they are associated in *some* way, this is obviously a much easier task (things mentioned in one and the same sentence are usually associated in one way or the other). As for the other two points, a surprising variety of information can be extracted from the network through the general procedure of *activating* a set of symbols, *propagating* the activation across associations, and *reading* off the resulting activation levels on another set of symbols. This procedure can be used to implement a query mechanism, and it can also be used to induce e.g. Hebbian learning and structural updates to the network. Section 3 will give the details of how all this can be done.

To summarize, then, the aim of this project has been to create a system that

- accepts reasonably simple NL sentences
- performs some syntactic and semantic extraction on these sentences
- builds an association-based semantic network using the extracted information
- answers NL queries related to the network content
- can generalize its knowledge to some degree.

It should perhaps be pointed out right away that our results so far (see section 5) have been fairly modest, and that we cannot claim to have accomplished all of the above objectives. Whether the association approach to semantics is feasible on a large scale must be regarded as a wide open question. Section 6 provides a discussion of the problems and promises of our model.

---

<sup>4</sup>Strictly speaking, this is not true for directed relations. As the report will show, we think there may be reasons for letting associations be directed.

## 3 Methods

### 3.1 Design overview

The system has been designed so as to consist of two well separated layers. Layer 1 is an interface layer – its job is to translate back and forth between natural language and network representation. The most important components of layer 1 are a *part-of-speech tagger*, a *sentence parser* with an associated *grammar*, and a *query mechanism*. Layer 2 provides the core functionality of the system. It is responsible for creating and maintaining the *network structure*, and for the mechanisms of learning, generalization, and inference, which we refer to as *network dynamics*.

### 3.2 Sentence parsing

For the system to be able to make sense out of a sentence of natural language (in effect English), the sentence needs to be translated into a form that gives some information about the syntactic and semantic relationships between its components. This is done in three steps.

#### 3.2.1 Part-of-speech tagging

Part-of-speech tagging is the task of assigning to each word in a sentence its proper part of speech (noun, verb, and so on). This is a first step towards obtaining a complete syntactic parse of the sentence. The system uses the following reduced set of tags:

AT	article	C	conjunction	EX	exist. "there"
J	adjective	N	noun	NS	plural noun
NG	genitive noun	O	gen. marker (of)	P	preposition
R	adverb	TO	inf. marker (to)	V	verb
VI	inf. form	VZ	s-form	VPP	past participle
VG	ing-form	VB	form of "be"	VH	form of "have"
VD	form of "do"	VM	modal	W	wh-adverb
<b>S</b>	sentence	<b>SP</b>	sub-sentence	<b>NP</b>	noun phrase
<b>VP</b>	noun phrase	<b>AP</b>	adv. phrase	<b>PP</b>	prep. phrase
<b>JP</b>	adj. phrase	<b>PROP</b>	start of propos.	<b>QUERY</b>	start of query

Table 1: Part-of-speech tags

A few remarks are in order here. First, the boldface tags are not used by the tagger per se, but are essential in the later stages of parsing. Second, the tag names are admittedly non-standard. To make matters worse we use the tags in a way that would probably make a linguist freeze in horror. For instance, the J tag is used to tag not just adjectives but many other things, such as determiners, that can modify a noun. Likewise, we have adopted a very broad definition of "adverb". More atrocities like these will follow in section 3.2.2.

The initial representation of a sentence fed into the system is simply a string of characters. The tagger first strips the string of all punctuation, pulls out the individual words, and converts all characters to lowercase. If the sentence is pre-tagged each word is followed by its tag; in this case the tagger simply separates the tags from the words and reports them back. Otherwise, the sentence is tagged using a Hidden Markov Model (HMM). We have made a straightforward implementation of the Viterbi algorithm, the only interesting feature being that the transition and emission probabilities are actually obtained from layer 2. This is in the spirit of our goal to incorporate as much of the language model as possible into the network, a benefit of which is that learned semantic knowledge can influence syntactic interpretation.

To understand how the probabilities are derived, notice that the network will have symbols representing not only every word it has seen, but also every tag it has seen. The strength of the association between a tag  $T$  and a word  $W$  can be used to obtain an estimate of  $P(W|T)$ , by normalizing with the total strength of all associations  $T - W_i$ . Actually, the association strengths are not accessed directly, but the general procedure

of activate-propagate-read (see section 3.4) is employed. Furthermore, smoothing is necessary to take unseen words into account. A vocabulary size of 5000 was assumed, and Lidstone’s law applied using  $\lambda = 0.000001$ .

For transition probabilities the situation is a little more complicated. Because  $P(T_1|T_2)$  and  $P(T_2|T_1)$  can differ radically, directed associations are needed to store this information. For this reason we include a special type of relations, *order relations*, in the network. (Another reason is that statistics on the sequential order of words and tags in sentences are important for language generation.) Order relations are treated separately from associations and do not usually take part in the network dynamics. To calculate transition probabilities, however, activation is propagated across order relations only. Apart from this peculiarity, the process (including normalizing and smoothing) is analogous to that for emission probabilities.

Finally, the prior probabilities of the first tag are obtained by checking how strongly each tag is "order related" with a "start tag" that is inserted in the beginning of every sentence. Because of the different word order for questions and propositions in English, different start tags (QUERY and PROP) are used for the different types of sentences.

In practice the procedure of calculating probabilities could get quite expensive for large networks. However, the probabilities need not be recalculated after every update to the network.

### 3.2.2 Parsing and grammar

The next step is to build a parse tree for the tagged sentence. This essentially amounts to identifying syntactic structures above the part-of-speech level, such as noun phrases (*the big fat cat*) or prepositional phrases (*in the box*). For this purpose we use the mostly excellent, freely available LoPar parser (for binaries and a description, see [6]). Unfortunately, given the sparsity of training data we were faced with (see section 4), and the limited time at our disposal, it seemed out of the question to attempt probabilistic parsing. It turned out that we were able to use symbolic parsing without much difficulty, probably because the simple sentences we use for training do not allow for much ambiguity.

A parser requires a *grammar* to do its job, i.e. a list of rewriting rules of the form  $S \rightarrow NP VP$  (to be read as, a sentence may be constructed from a noun phrase followed by a verb phrase). Here, we were on our own, for several reasons. First, "real" grammars for English tend to operate on 100-200 tags instead of our mere 21. We had neither the time nor the linguistic know-how required to deal with such a large tag set (or the associated grammar), and there was also no need for it since we were going to use only simple sentences for training. Second, the parses produced by a general-purpose grammar would not necessarily be adequate for our purposes. For example, the sentence *The cat is in the box* would probably be divided into a noun phrase (*the cat*) and a verb phrase *is in the box*, the latter of which would in turn be subdivided into a verb (*is*) and a prepositional phrase (*in the box*) which would again be split. This hierarchy is questionable from a semantic point of view. The association we most of all would like to derive from the sentence is clearly *cat-box*, but the distance between these two words in the parse tree is quite large.

Thus, we were forced to write our own grammar. We did this using the age-old, ill-reputed technique of adding a new rule whenever a sentence didn't parse. To deal with the second problem we used sets of rules intended to flatten the parse tree by allowing branches of various arity. For instance, some rewriting rules for SP (which is essentially a sentence) are  $SP \rightarrow NP VP$ ,  $SP \rightarrow NP VP NP$ ,  $SP \rightarrow NP VP PP$ ,  $SP \rightarrow NP VP NP PP \dots$ . This is admittedly not a good idea in general, and would probably not work for full-sized grammars. However, given that the functionality of layer 1 was not the main focus of the project, it seems like an admissible hack. In the presence of more complex training data, more sophisticated methods would have to be used to obtain semantically relevant parses.

The final grammar, which is a horrible mess, is available in the submit directory as `final.gram`. It works for a fairly large class of basic propositions, the most obvious shortcomings perhaps being its inability to handle pronouns and compound sentences. It can also parse the most common types of question constructs.

### 3.2.3 Group extraction

The final step in layer 1's processing of a sentence is to extract *semantic groups* – and their interrelations – from the parse tree, to be given as input to the network. The current method for doing this is not terribly advanced – it consists of flattening the tree even further by removing all unary branches (e.g., if  $A \rightarrow B$  and  $B \rightarrow C D$ , substitute with  $A \rightarrow C D$ ), and then reporting each subtree as a group. For every group, one child is

reported as the *head node*; this information is given in the rules of the grammar. For instance, in the group  $JP \leftarrow NP \rightarrow cat$ , the head node would be *cat*.

The sentence is now finally ready to be presented to the network. Most importantly, layer 2 will receive the tree of semantic groups. In addition, it will be given the sentence as a plain word-tag sequence, which is necessary for order relations to be created. The next section describes what layer 2 does with this information.

### 3.3 Network structure

In the heart of our project lies the problem of designing a good model, one that is able to learn new semantic information and display the information that it has already acquired. As hinted at earlier, the model we designed is a network-based model. Nodes in the network represent symbols – either single words, parts of speech, or parts of sentences. These nodes are sparsely related to each other by directed and undirected relations. We use the directed relations to learn the order of words and parts of speech in a sentence. The undirected relations are used for learning semantic associations between the above symbols (words, parts of speech, parts of sentences).

A vital property of an association is its *strength*. The basic idea we used for adjusting association strengths is similar to that of *Hebbian* learning, meaning that symbols that are activated at the same time are likely to be related to each other, and therefore the association between them should be strengthened. The association strengths, together with the network structure, form a long-term memory in the model. Another, short-term memory, consists of firing rates (activation strengths) of the nodes in the network. These are propagated throughout the network according to the strengths of the associations. Note that all relation strengths, and node activation rates, were limited to the range  $[0, 1]$ .

When new sentences are introduced to the network new nodes and new relations are created, and existing relations are strengthened. This is done in three steps: insertion, propagation, and update. In the insertion step nodes and relations are inserted and activated. Then, in the propagation step, nodes that fire cause their neighbor nodes to fire. Last, the relation strengths are updated according to the firing rates.

The exact way insertion is done depends on the structure of the parse tree of the sentence that is read. This parse tree is given to the system as input, together with the sentence. New nodes are created for each new word or part of speech that appear in the sentence and do not already exist in the network. In addition, every subtree of the parse tree that is not a single word – the semantic groups of section 3.2.3 – is assigned a new node. New associations are created between subtree nodes and their children nodes, and between words and their corresponding parts of speech as they appeared in the sentence. Directed relations are created only between parts of speech that follow each other, and words that follow each other.

For example, in the sentence *cs224N is a great course*, two new nodes will be created: one node for the entire sentence, and one node for the subtree *great course* (we remove all articles from the sentence for this part). The sentence node is then connected to three other nodes – the nodes of the single words *cs224N* and *is*, and the node that represents the subtree *great course*. This is illustrated in Figure 1.

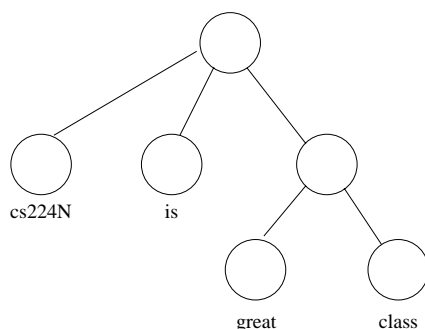


Figure 1: Network structure induced from simple sentence

The symbols are then activated in the following way: Every word in the sentence brings with it some basic constant *energy*. The sum of these energies constitutes the *basic sentence energy*, which is therefore

linear in the length of the sentence. This energy is then distributed to the nodes so that, in every semantic group, the group node gets the highest amount of energy. One of the children, the head node, is preselected to be the most important one in the subtree and it gets the highest share of the remaining energy. The remains are distributed evenly among the rest of the children.

The above insertion mechanism results in a network that grows linearly in the number of sentences (assuming that a sentence length is bounded). One may correctly point out that when a subtree appears many times in different sentences, we would end up adding new nodes each time it appears. This is true, and we made no effort to limit this, nor did we see it necessary to do so. In fact redundancy may many times be useful by averaging out random errors. Moreover, it is our feeling that a model that requires a lot of memory does not pose a serious problem, as long as this doesn't effect computation times. Our model lends itself well to parallelism, and so computation time in a suitable hardware system shouldn't be much affected by network size.

### 3.4 Network dynamics

The previous section dealt mainly with the static properties of the network, and the "learning by intervention" triggered by new sentences presented to the system. In this section we will consider how information is passed in the network, and how the network develops and changes.

#### 3.4.1 Propagation

Propagation of information is done in the network by simply *adding* the sum of the inputs to a node (association strength multiplied by the firing rate of the node on the other side of the association), to its current activation rate. Note that the reason we add and not replace is that the network is recurrent, and so replacement would propagate zeros into active areas. On every propagation step a firing rate *decay* is used, so that saturation will be avoided, and so that information that is no longer relevant will decay. This propagation is done according to a constant order (nodes that were inserted first are updated first). We did not investigate whether random order, synchronous mechanisms, or BFS-based orders would work better.

One problem that manifested itself quite early with this propagation model is that signals tend to fade away very quickly as they propagate through associations, because they are multiplied by the association weights which are in the range [0,1]. Boosting the propagated signals often resulted in a complete saturation of the network, and so this delicate point had to be finely tuned.

In addition to that, a basic trade-off lies in the question of how many times to propagate. On the one hand, propagating over large distances may help us make interesting connections between symbols. But, on the other hand, it may also bring a lot of information into irrelevant areas, and this will have the effect of adding a lot of noise to those areas. We are currently thinking of other methods that may substitute or improve this propagation model.

#### 3.4.2 Update

We used several opposing mechanisms to control the balance in the network. The first of them was a growth mechanism. We have already discussed how new symbols and associations are created when a new sentence is introduced to the network. Here we discuss the way association strengths are updated.

We have tried a few different ways to strengthen relations. Among these we tried using normal *Hebbian* learning:  $\Delta\omega = u*v$  where  $u$ , and  $v$  are the firing (activation) rates of the nodes that a relation with strength  $\omega$  connects. We also tried using an exponential growth variant:  $\Delta\omega = u * v * \omega$ . We tried using constant strengths, and lastly we tried using the following learning rule:

$$\Delta\omega = \begin{cases} 0 & \text{for a newly created relation} \\ 9 * \omega & \text{on the second update} \\ u * v & \text{after the second update} \end{cases}$$

The intuition behind this will be understood after the description of the counter-mechanism.

The counter-mechanism we used to oppose this growing mechanism is a linear *decay* of weights (which should not be confused with the firing rate decay described above). This slow decay caused weak associations

to die. On the one hand this “forgetting” mechanism makes us lose information, but on the other hand, associations that die are almost only such that were updated only once, and so are probably not important and are possibly even “confusing” associations. This is the case because our update rule strengthens associations that were called on twice a lot, and so associations that are updated twice live long.

### 3.4.3 Generalization

We have tried to make the network generalize and make new inferences based on its semantic information. We tried this by randomly creating new associations between symbols that simultaneously had high activation rates. The required threshold, that selected how high the activation rates should be, was manually adjusted. As will be discussed later very small perturbations of this threshold caused great changes in the network.

One form of generalization that we did not attempt is the creation of new symbols unrelated to any particular sentence. For instance, one could try to identify “cliques” of highly interconnected nodes and split these up by adding a new “hub node” to which the previous nodes would all connect, replacing their previous inter-relations. A clique could perhaps be defined as a set of nodes that consistently show a synchronized activation pattern.

In the best of all worlds, this mechanism could interact nicely with the afore-mentioned one. As an example of how this could work, consider a data set in which many sentences mention kinds of foods. Presumably, the symbols created to represent different foods would have some associations to other symbols (*eat*, *table* in common. For this reason they will tend to be activated simultaneously, which will enable the association-adding mechanism to kick in. Finally, when enough associations have been added between the foods symbols, a new symbol would be created too which the foods symbols would connect. The system would then have discovered the abstract concept of “food” all by itself.

We have not found a good way to generalize on the information about the order of words. Our hope was that we could use the acquired semantic information to do this. We had some ideas as to how to do this for the purpose of language generation, but since we did not implement a language generation part for the system, this was not explored.

## 3.5 Training

As seen in the previous section, learning in the network is a delicate process. From an outside point of view, however, training of the system is straightforward – it consists simply of feeding sentences to the system. There are two ways of doing this: batch mode, which is the most appropriate for training, and interactive mode, which is more intended for query processing. In either case, each sentence will be handled individually exactly as described in the previous sections.

Initially, the sentences given to the system must be pre-tagged, so that the network can start learning the associations necessary for tagging. Fairly soon, however, untagged sentences can be interleaved with tagged ones. Sentences that fail to parse will simply be ignored; a nice feature would be if they were automatically saved and tried again later, when the tagger will perhaps do a better job.

## 3.6 Queries

Of course, a system for information extraction and storage is not of much use unless it comes with a mechanism for accepting and responding to *queries* regarding its state of knowledge. For an NL system, one goal must be the ability to answer questions posed in natural language. We therefore wanted to try to implement such a query mechanism for our system.

In principle, many types of queries should be possible to answer using the basic activate-propagate-read procedure described in previous sections. The key is the ability to recast queries as queries about associations. For instance, the question *Where is the cat?* is the question of what location is most strongly associated with *cat*. The question *Is Peter tall?* asks about the strength of the association between *Peter* and *tall*, whereas *Is Peter the tallest?* asks about this strength in relation to that of associations between *tall* and other symbols representing people. (Here the term association strength is used in a dynamic sense, allowing for the fact that activation can be propagated from *tall* to *Peter* along many different paths.)

Thus, the basic idea behind our proposed query mechanism is as follows. Layer 1 is used to parse query sentences just as well as propositions; however, the information extracted from a query sentence is not used to update the network, but rather to select certain nodes in the network to be activated (the *trigger nodes* and another one (the *query node*) to be read. The activate-propagate-read procedure can be repeated several times with different sets of trigger nodes for comparison purposes. In some cases it may be more appropriate to look for the nodes with the highest overall firing rates after propagation instead of designating a specific query node. Because of the probabilistic nature of the system, it probably makes sense to generate a list of the responses deemed to be most likely, or to give the probability of a particular answer.

The query mechanism that we actually implemented was able to process queries such as *Is the cat green?* and *Does Peter like to run in the mornings?*. However, the quality of the answers was usually not that impressive. We believe that the main reason for this is that our methods for automatically selecting trigger nodes and their corresponding activation was too simplistic, and did not manage to make full use of the semantic information given in the queries. In the end, while the results obtained using the query mechanism described here provided some feedback about the functionality of the system, we had to use more specialized query types to assess the soundness of the network content (see section 5).

### 3.7 Division of labor

For grading purposes, here is a summary of how the work has been divided between the two authors. The overall design of the project, including the principle algorithms and organization of data used in layer 1 and layer 2, has been a joint effort. During the implementation phase, Ido worked on layer 2 (network structure and dynamics) whereas Magnus focused on layer 1 (tagging, parsing, and query mechanism). In the evaluation phase, Ido did some more testing (including the creation of training data) than report writing, whereas Magnus did the opposite.

## 4 Training data

### 4.1 Available data

A problem throughout the project has been that we have been forced to create our own training data. The main reason is the apparent lack of simple text corpuses for purposes of computational linguistics. Another reason is that the training data and grammar must be made to work together, and since our grammar is custom-made (see section 3.2.2) this calls for custom-made data.

A potentially good source of simple text, syntactically simplified and semantically focused, are children books (the main problem perhaps being that children books tend to be rather short). It appears possible to create a good training corpus based on this kind of text, supplemented with common-sense type propositions. However, since the system is not really ready for large-scale testing yet, we feel that a large effort on corpus creation would have been unmotivated.

### 4.2 Custom-made data

Instead, the data that we have for the most part used for training consists of two small sets of sentences, painfully hand-crafted by the authors. Both of the data sets evolve around the afore-mentioned children book domain, including proper gender stereotypes and all<sup>56</sup>. Here are some typical sentences:

Peter's<sub>NG</sub> neighbor<sub>N</sub> has<sub>V</sub> a<sub>AT</sub> mean<sub>J</sub> dog<sub>N</sub>. The<sub>AT</sub> dog's<sub>NG</sub> name<sub>N</sub> is<sub>V</sub> Nigel<sub>N</sub>.  
Nigel<sub>N</sub> is<sub>V</sub> a<sub>AT</sub> very<sub>R</sub> bad<sub>J</sub> dog<sub>N</sub>. Nigel<sub>N</sub> likes<sub>V</sub> chasing<sub>VG</sub> children<sub>N</sub>. Many<sub>J</sub> dogs<sub>N</sub>  
like<sub>V</sub> chasing<sub>VG</sub> children<sub>N</sub>.

The sentences were intentionally written to form coherent stories, so that short-term memory properties of the network can be tested. The data set consist of 94 and 30 pre-tagged sentences, respectively. While this

---

<sup>5</sup>At some points during the course of the project, the authors pondered whether they in fact did not show more talent for writing children stories than for doing research.

<sup>6</sup>The data files are available in the submit directory as `toys.txt` and `kids.txt`, respectively.

may seem ridiculously small, it turned out to be sufficient for initial testing and debugging, as well as for obtaining some preliminary results. Also, because the system is meant to be able to learn "from scratch", its performance on very small training sets is interesting in itself.

## 5 Results

### 5.1 Parsing

Since the functionality of layer 1 is not really central to our project, and the parser not even of our own design, we have not done any systematic testing of this part of the system. However, it should be noted that the part-of-speech tagger seems to do quite well; this gives a preliminary indication that the data stored in the network is not complete nonsense. For example, after training once on the larger of the data sets, the tagger manages to re-tag all of the same sentences correctly, and 40% of the sentences in the smaller set. While this is nice, it is of course impossible to make a real evaluation of the tagger unless larger data sets are provided.

### 5.2 Semantic processing

#### 5.2.1 Hand-crafted queries

Testing a system like ours is not a simple task. There is no clear mapping of inputs to outputs that can be checked for. It is not even clear what the outputs should be, as those may vary for different usages of the system. The way we decided to test the system is as follows.

We created a list of relatively simple memory- and text comprehension questions about our training data. For each one of these questions, we hand-picked the order of importance of words in the sentence, and made the corresponding symbols in the network fire at different levels (this corresponds to the trigger node selection described in section 3.6). One word in every sentence was the expected answer (even though some of the questions had more than one right answer we always picked only one); this word was not activated, but it was given a part of speech, the symbol of which *was* activated.

In order to use these questions we silenced the network, activated the selected symbols, propagated, and looked at the resulting list of active symbols (sorted by firing rate). We obtained a second list of symbols by silencing the network again and activating, this time, only the requested part of speech. We then took from the first list only words that appeared in the second list, and marked the rank in the order of firing rates of the correct answer. We used this process to compare how different settings of network parameters affected this rank in the firing rate order.

For example: In the text appeared several sentences about hair. One of them was *roger has a light brown hair*. A question that we formed about this was *?roger(Noun) has(MID) a(LOW) light(HIGH) brown(HIGH) hair(MID)?* where the bracketed values are the firing rates that we inserted. We designed them to emphasize the parts of the sentence that normal human intonation would have emphasized (considering that the text included also the sentence *lucy has a beautiful long hair*). We then silenced the network, fired the "noun" symbol, and got a second list of symbols. After removing all the words that did not appear in the second list, and those that appeared in the question, the remaining words that appeared in the first list were (by order of firing rate): *alice, roger, lucy, house, bob, bear, tail, . . .*. The reason that *alice* (and not *roger*) came up first is that she had many sentences in the text that included the words *has, hair*. So, even though *roger* came up at a nice place, this has shown us a few of the problems of our model. First, it showed how difficult it is to hand-craft queries to the system. Second, it showed that it's hard to direct the flow of activation in the network in the correct directions.

Out of 25 questions that we gave the system, 22 scored an average of about 2 (meaning that the correct answer was in the second place, like *roger* in the above example), and the other three had significantly worse results (places 11, 12 and 22). There were some apparent reasons for these bad answers, having to do with questions for which the answers were rare words, and questions that involved highly connected words that lit up the whole network. All in all, it seems to us that on average these are pretty good results.

### 5.2.2 Adding noise

In order to further examine the network we tried changing it in several ways. The first experiment was designed to check how adding some minor noise to the network, instead of silencing it before queries, affected the results. This change resulted in some improvement in some cases, and worse results in other cases. Not too surprisingly, improvement was made in cases where difficult questions had simple answers.

For example, the question that got the 22nd place on the previous test got in first on this one. The question was *?alice's room is the room of \_\_\_?*. This is a difficult question in the sense that *alice* and *alice's* never appeared together in the text, and our model doesn't separate words into morphemes, so the similarity in sound and spelling between *alice* and *alice's* could not be used by the system. That is probably one of the reasons it was placed only in the 22nd place on the first test. However, when we added some noise, the correct and very frequent word *alice* rose up enough to get to the first place. As a contrary example, the answer to the question *?alice will \_\_\_ tomorrow?*, where the expected answer was the rare word *cry* (appeared in the text only once), got worse results than when checked for without noise.

### 5.2.3 Changing the propagation distance

We tried to change the number of propagation iterations in two ways. First, in queries, recall that the process involves lighting up some symbols, propagating, and then looking at the firing rate rank of the expected answer. We tried to propagate 2, 4, 8, and 12 iterations (propagation in training was 4 iterations in these cases). This test was supposed to examine the propagation distance trade-off (better inference vs. irrelevant information) that was discussed before.

Results here showed that propagating only 2 steps produces significantly worse results than propagating more, probably because the activation from the trigger words did not reach important symbols. Propagating 4 iterations was slightly worse than 8 and 12 iterations, and 8 and 12 iterations produced the same results (the base results that were discussed above were taken from the 8 step propagation). The reason these got so similar results is that every propagation weakens the signals, and so after about 5 steps the signals practically die out. Raising the strengths of associations, or boosting the propagating signals in order to counteract this, quickly results in network saturation and gives bad results. This saturation requires re-tuning of all network parameters, which is not always successful. The necessity to hand-tune the parameters of the network in this case and others is one of the major faults of our model.

The second case we tried was to change the number of propagation iterations in training (recall that training was formed from insertion, propagation and update). We propagated 0, 2, 4, 8, and 20 iterations, and the results we got were very similar to the results in the first case. Zero propagations was a little bit like using random noise (as described in 5.2.2), giving good results mainly to questions for which the answers are common words. Note, though, that when this gave bad answers they were not as bad as when the random noise gave bad answers, probably because some local information is still used here more efficiently.

Propagating only 2 steps produced slightly worse results than 4, 8, or 20 steps, which all gave the same results. Again, the similarity between the 4, 8, and 20 results is probably a result of the weakening effect on signals that propagate over long distances.

### 5.2.4 Changing the threshold for creation of new associations

As hinted at before, the very sensitive firing rate threshold for creation of new associations changes the behavior of the network a lot when perturbed a little. This happens because positive feedback loops are created – more associations mean stronger signals, which in turn cause the creation of new associations. This meant that changing this threshold required tuning the other network parameters again, which is a painful and exhausting task, to restore stability to the network.

Creating more associations has other important implications. Creating new associations causes the initial sentence based structure of the network to become less and less significant, because other none structured associations are created. We went on and checked what the effects of distorting the basic structure in this way were, and the major result that we got on this part was that even after re-tuning the network parameters we got worse results than normal, and in general the less additional associations we created the better were the results. Looking at the results we saw earlier, that words common in the training data tend to appear too

frequently as answers to queries – this is a direct result of the hyper-connectivity. We should note however, that different tuning may have made it work better.

### 5.2.5 Association strength learning

As discussed before we tried several association strength adjustments methods. Out of those, the exponential growth variant  $\Delta\omega = u * v * \omega$  resulted in an expected saturation of the network and catastrophic results. The two-step function:

$$\Delta\omega = \begin{cases} 0 & \text{for a newly created association} \\ 9 * \omega & \text{on the second update} \\ u * v & \text{after the second update} \end{cases}$$

resulted, for some questions, in very good results (again about 1 average), but where it failed the right answer was usually not even among the first 100 answers. The reason for this is that associations relating things that appeared only once in the data got removed from the network. The benefit of this function was, however, that the size of the resulting network was much smaller (about 2/3 the size of the final network). This means that in some cases an update rule like this may be a good thing to use, e.g. for systems that are run over very large periods of time. However, since our test case was more of a short memory task (because we used a small text and immediate questions), performance suffered.

We therefore ended up using the normal *Hebbian* update rule  $\Delta\omega = u * v$  with our addition of decaying weights. We checked, for this case, whether the decay was useful or not, and it turned out that results were hardly changed by adding the decay, but the size of the network was 20% smaller; so we kept it.

Another relation strength learning method that we tried was to simply keep the weights constant (with/without additional decay). We were a little surprised to find that even though the results were slightly changed, on average they were almost as good as with the *Hebbian* learning rule. This probably means that we didn't manage to tune the network in a way that would take advantage of learning.

### 5.2.6 Contrasting with results on test data

One may wonder if we didn't get the nice results by simply over-fitting the network's parameters to our training text. In order to test this we wrote another text that had some of the same topics as the first one, and some topics of its own. We then trained the network on the new text and asked it questions from a new set of queries. The test file was smaller (about a third of the size of the training text) and contained less words, so naturally the list of possible answers for each question was smaller. With that in mind, the new average rank of answers was 2.5, and the two worst results out of 13 questions were both located at rank 5 in their corresponding orders. These good results suggest that over-fitting did not occur here (though more tests may be required to confirm this).

We went on to examine the effects of joining the texts. We first fed the network with the training data, then with the test data, and then tried all (25+13=) 38 questions. We then did the same thing in the opposite order. Our results were that questions whose answers appeared high in the order in the initial tests did not suffer at all, or suffered only very slightly (like moving one step back in the order). Questions that had bad initial results got even worse results. This happened because those bad answers had low firing rates in the initial case (that's why they were so low on the list), and so competing with them was relatively easy for the new words. Moreover, it seems that the queries that suffered the most were queries in which words that appeared in both documents appeared also in the question, and so caused confusion as to which text the question was related to.

One last intuitive result on this one was that texts had better results if they were presented last. So for example when the training text was presented last, it got better results than when it was presented first. This is due to the decay mechanism that was discussed above, causing early material to be less important than new material.

## 6 Discussion

When approaching a project as large as this, in an area that has not been much explored, and when dealing with highly dynamic systems, it is often very hard to predict what will work and what will fail, where major problems will arise, and where nice surprises may come. In our case, failure would have been to have absolutely no results (a stage we were in many times during this project), and success would have been a system that could learn massive amounts of information and speak about it in natural language. Our results were more somewhere in-between: after plenty of hard work we got fairly good results, and learned some lessons along the way.

First of all, our nice initial results show that the path we took may after all be a good path to follow. When trying to evaluate more specifically what helped produce the good results, it seems to us that since the actual weight values did not have much effect it must have been the sentence-induced network structure that was basically good, and the idea of using associations to propagate activation between words. And, while our attempts to refine the network using various learning and generalization methods produced rather disappointing results, this does not mean that no such methods exist. Alternative approaches have been suggested here and there throughout the text, and some more comments will follow below.

There are, however, some severe problems inherent in the model. First of these is probably the problem of having to hand-tune the many parameters of a dynamic system, especially the problem of maintaining stability. It seems that our mechanism lacks a basic gradient towards stability. A very basic improvement in this respect would be to design a mechanism that would control the network parameters dynamically. It may very well be, for example, that as a network grows the rate of adding new associations and symbols should change too.

Second, one of the reasons that a system like this is very hard to control is that we have no way to look at the actual dynamics of the system, but are instead constrained to examine things locally. Hence, an important step for future development in this direction would be to find good ways to visualize the network structure and activation flows.

Third, our system used only positive sentences, and positive associations. This turned out to be more limiting than we expected it to be, not only because negatives are so common in language (aren't they?), but also because the nature of associations between words is many times negative (like in *bob likes . . .* and *bob hates . . .*). Moreover, when we want to ask the system questions we usually want to suppress all unrelated stuff, so in the example *\_\_ has a light brown hair* we would like to suppress all things whose hair is NOT light brown, and not only activate the things that do have a light brown hair.

Fourth, we tried to take advantage of on-going conversations by not silencing the network when training, so that topics from the previous sentences will remain activated. This may also make new associations of the type that are created in the generalization process more useful. We tried to implement this, but were unable to tune the network so that it didn't saturate. Nevertheless this seems like an important thing that should be used and thought of for future versions.

Fifth, as it seems like we haven't been able to make much use of the weight learning, it would be interesting to think about other methods that may be more successful. One problem with the way we did it is that there is no clear interpretation for the association strengths' values, or a good definition for what the weights try to maximize, or minimize. It would be very interesting to develop a similar setting where the weights represent probability distributions in some more precise way.

And, lastly, many problems occur because we are only dealing with words. It seems highly unlikely to us that in a more complex and developed system, the system will be able to "understand" the difference, for instance, between "in" and "on" and be able to generalize about them. Reason for that being that these words appear in similar contexts, but their meaning is taking from another world – the "true" world. So, a fully functional system, in our view, would have to use semantics in their proper way, meaning as functions from language to real objects, actions, states etc.

### 6.1 Future improvements

As has been stated several times the core of this project is the semantic network, and so a continuation of the project ought to focus, following the above discussion, on achieving better interpretation, learning, and generalization of network content along. However, there are also some features, not directly related to

the network, that would definitely belong in a complete implementation of our system, and that have so far been left out entirely. The most prominent one is of course language generation – the ability to express the semantic knowledge encoded in the network as natural language, in particular in response to queries. Implementing this seems like a major project in itself; we have not given it much thought, and it will not be discussed further here.

Another obvious shortcoming of the current implementation is its complete lack of morphological processing. Layer 1 has no way of telling that *cat* and *cats* are closely related syntactically as well as semantically. In theory it would be possible for the network to deduce this by itself, given enough data – but that seems rather far-fetched. Therefore, it would probably improve the system a good deal if layer 1 knew enough about morphology to be able to present words to layer 2 only in their common form. As it is, there will be many symbols in the network referring to essentially the same thing, something which makes generalization and inference much harder.

Yet another thing for which layer 1 must assume responsibility is the handling of cross-references in and in-between sentences. Good short-term memory properties of the network can be of help, but some pre-processing in layer 1 seems unavoidable if this kind of information is to be handled correctly.

## 6.2 Conclusion

One might wonder whether some of the alternative approaches to semantics mentioned in section 2 would not in fact have produced much better results than our semantic network on the training data and test queries we used. The answer is almost certainly yes. However, it would be a mistake to use this as an argument against the association model. In fact, it is precisely on these kinds of limited domains, with demands for highly precise answers, that e.g. logic-based approaches can be expected to fare well. We believe that an associative semantic network, properly used, could have a much wider applicability.

That said, we would not in any way want to claim that we have managed to present a compelling argument *for* the usefulness of the association model. Careful revision of the model, together with much more extensive testing, would be required before any such conclusion could be drawn. Nevertheless we hope that we have managed to present some interesting ideas that can be of use in future research on semantic modeling.

## References

- [1] Hutchens, J. L., and Alder, M.D.: *Introducing MegaHal*. Online, <http://ciips.ee.uwa.edu.au/hutch/hal/Talk.html>, 1998.
- [2] Manning, C. D., and Schütze, H.: *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
- [3] Manning, C. D.: *An Informal but Respectable Approach to Computational Semantics*. Online, <http://www.stanford.edu/class/cs224n/materials.html>, 2000.
- [4] McCarthy, J.: "Programs with Common Sense". In *Mechanisation of Thought Processes. Proceedings of the Symposium of the National Physics Laboratory*, pp. 77-84. London, 1959
- [5] Richardson, S. D, Dolan, W. B, and Vanderwende, L.: *MindNet: acquiring and structuring semantic information from text*. Online, <http://research.microsoft.com/nlp/nlppubs.asp>, 1999.
- [6] Schmid, H.: *LoPar: Design and Implementation*. Online, <http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/LoPar-en.html>, 2000.