WordCat

Sean McLennan
B651 - Dr. Michael Gasser
Indiana University
December 14, 1999

1. Introduction

WordCat is a word recognition utility whose architecture was inspired by CopyCat, a system for analogy-making (Mitchell, 1993). One of CopyCat's most unique features is that the text it is presented is a completely unanalyzed chunk. There are no control or delimiting characters — nothing has an explicit effect on processing other than those letters that are directly perceivable and manipulable by the system. I feel that this approach could be very valuable in a radically different domain — text parsing.

Since a complete text parser of this type would be a dramatic undertaking, a sub-task is required. I have chosen word recognition, what I consider to be the lowest level of text parsing of this type. By saying that WordCat "recognizes words" I mean that it is presented "noisy" text — words with typos or spelling mistakes — and WordCat supplies the closest match from its lexicon, hopefully the intended word. WordCat was implemented in C++ and has two gross functions: creating the network that is central to its functioning, and using said network for word recognition. At this point, WordCat only recognizes words in isolation, not in context.

WordCat's performance was tested by systematically altering arbitrary words that exist in its lexicon and comparing them with its output. In general, WordCat performed well, making few unreasonable errors and I believe the problem areas could be ameliorated by further implementing CopyCat-like features.

Section 2 below discusses CopyCat and its most salient features with special attention to those that have explicit analogs in WordCat. Section 3 details the design, implementation, and use of WordCat itself and describes how it relates to CopyCat. Section 4 summarizes the results of the tests performed (full results appear in Appendix D), and finally, section 5 discusses WordCat's advantages, limitations, promise, and applications.

2. CopyCat

CopyCat is an model of high-level perception and analogy making in a well defined microworld designed by Douglas Hofstadter and Melanie Mitchell (Mitchell, 1993). The analogy problems with which CopyCat works are strings of letters, for example:

abc → abd
xyz →?

CopyCat's task is to fill in the "?" by making an analogy of the transformation that occurs above. It is not the analogy making process that I feel has particular value for text parsing at this stage. It is that CopyCat builds all of its own structure based entirely on observations it makes concerning the individual letters and their immediate neighborhoods.

CopyCat's "knowledge" of its microworld consists of the successor / predecessor relationships between the 26 letters of the alphabet, the numbers 1 to 5 (and their predecessor, successor relationships), and a handful of "concepts" such as "left", "right", and "opposite". CopyCat is supplied a string of characters (implicitly providing information about which are the first and last elements in the string) and no other information. CopyCat searches for similarities and correspondences, eventually building a quite sophisticated, hierarchical structure that it uses to produce a solution to the problem. This task is very similar to that of a bottom up text-parser — given a string of characters, build a representation of the sentence.

There are four main aspects to CopyCat's architecture: the Slipnet, the Workspace, the Coderack, and the Temperature. All of these divisions rely integrally on stochastic, dynamical processes that make the system as a whole non-deterministic.

2.1. The Slipnet

CopyCat's Slipnet is where all of CopyCat's "knowledge" is represented. It is important to note that although it is similar to a neural network in that it contains nodes and links and employs spreading activation, it is not intended to be a neural network. Each node in the network represents a single concept and the links represent the relationship between nodes. Because there can be different types of relationships (successor, predecessor, opposite, etc.), the links themselves are connected to concept nodes that label the relationship and the "closeness" of the nodes (i.e. weight of the link) is dependent on the activation of the label. Additionally, each node has a "depth" that is used in helping determine the quality of the structure being built. Essentially, the more abstract the concept, the deeper the node; and CopyCat likes deeper concepts. There is no learning in the Slipnet, although, over the course of a problem, the dynamics of the network constantly change.

Activations in the Slipnet are supplied by "Codelets" (described below) when they perceive a similarity / structure / concepts in the inputted string. When the closeness of two nodes in the Slipnet increases (corresponding to activation of the appropriate concept node), "Slippages" of concept occur; that is, two structures or concepts are considered analogous. The result is a very flexible tool for directing the search for a solution to the analogy problem.

2.2. The Workspace, Coderack, and Temperature

The Workspace is where CopyCat builds structure on its letter strings, simultaneously entertaining several hypotheses that can be built up or destroyed until an acceptable solution is found. The Coderack is a waiting area for Codelets which are small, independent, task specific bits of code that act in the Workspace. Codelets are stochastically chosen from the Coderack to be run and do things like "notice" facts about the Workspace, "seek out" particular hypothesized structures, place other Codelets on the Coderack, supply activation to the Slipnet, and build / destroy structure in the Workspace. They perform all of the work in CopyCat.

The decision process that allows CopyCat to stop working and produce a solution is also a probabilistic process and is dependent on two factors: the Temperature and the amount of structure built. The Temperature, in turn, is dependent on the amount of structure and the quality of the structure (i.e. the system's confidence that the structure is correct). If the Temperature is low, a great deal of high quality structure has been found. Thus the Temperature is an assessment of the structure built.

3. WordCat

The primary architectural feature of CopyCat that was focused on in WordCat was the Slipnet. The network that WordCat uses similarly has Nodes of varying depth and links of different types that reflect different relationships. However, unlike CopyCat where both link weights and activations change over time, WordCat's links remain static while the program is running. In fact, WordCat uses no stochastic processes at all. Instead, it builds its network based on the observed letter transitions in a supplied training text. Thus, the links in the network reflect the statistics of the language and some degree of non-determinism is exhibited given that the inclusion or exclusion of a single word

in the training text can sometimes have a dramatic effect across runs.

WordCat has an assessment of its current hypothesis — its "certainty" — that is analogous to CopyCat's Temperature and determines how long the program will work on a particular word. WordCat's Workspace is simply a list in which it makes and tests predictions. Finally, WordCat does not implement anything like the Coderack or Codelets. It's search, prediction, and decision paths are completely static.

3.1. WordCat's Network

Design:

WordCat's Network is implemented in C++ lists of three types: Node, SuccConn (successor connections / links), and FamConn (family connections / links). I chose lists primarily for the flexibility they offer with regards to network size. There are three "depths" of nodes in WordCat: D1 - single letters, D2 - letter pairs, and D3 - full words. As the number of depth 1 nodes increase, so to do the letter pairs and links — exponentially. It is thus advantageous to include *only* those nodes and links that are actually present in the training text, greatly reducing the number of nodes and links, and helping to reflect the graphotactics of the language. By using lists, I was able to constantly vary the text without having to determine new array sizes.

The tradeoff for this flexibility, however, is speed. WordCat does not employ sophisticated searching or sorting algorithms and so creating and updating the network is very slow. For a network of ~550 nodes and ~3000 links that was trained on a text of 300 words, it usually takes 30sec to a minute or two to recognize a word. It can take anywhere from a few minutes to an hour to create the network. This is a major drawback to WordCat, but it is primarily a flaw in the implementation, not necessarily with the approach.

The Node class contains 3 pieces of information: the node's name, the node's depth, and the node's activation. There are two class of links, SuccConn and FamConn, that both contain the predecessor of the link, the successor of the link, and the link's weight. SuccConn links reflect the transitions observed in the training text from single letters to the following letter and letter pair. Thus SuccConn links exist between D1 nodes and (unidirectionally) between D1 and D2 nodes. FamConn links on the other hand reflect "familial" relationships between the nodes. They exist unidirectionally

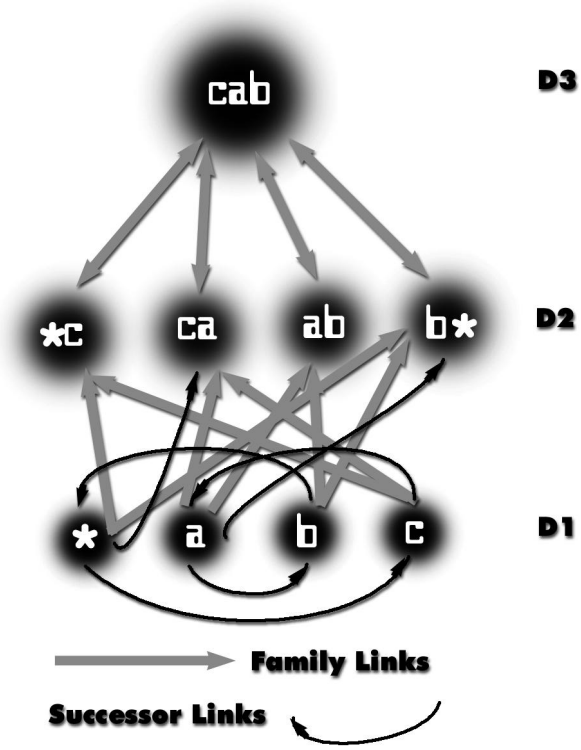

Fig. 1 WordCat's Network for the word "cab". "*" represents a space.

from D1 to D2 nodes (for example **a** → **ca**) and bidirectionally between D2 and D3 nodes (**ca** ® **can**). D1→D2 and D3→D2 family links have fixed weights and D2→D3 links are normalized for word length. A diagram of WordCat's Network appears in Fig. 1.

Training Text:

The training text was constrained to the 26 lowercase letters, the apostrophe and asterix. This was for size considerations; to reduce the number of nodes and links. All characters are capable of being used by WordCat - the only real constraint being that the text to be recognized only contain characters that appeared in the training text. The apostrophe was included to deal with contractions effectively and the asterix represents the space character. A space character was included to provide information about the beginnings and endings of words; the asterix was chosen for visibility and to simplify manipulation since C++ treats the space as a default delimiter.

The text itself is a children's story called the "The Mole and the Owl". It appears in Appendix A. A total of 800 words were formatted and networks were built from training texts of varying size.

Algorithms:

The algorithms used in building and using the network are not sophisticated. The link weights were increased each time the same transition was observed in the training text simply by increasing the value of the weight using the following formula:

```
myweight += (1-myweight)/20;
```

Thus the weight asymptotes at 1; the constant **20** was derived empirically through trial and error to obtain a relatively broad range of weights.

Two types of related but subtly different statistics are stored in the network: transition

occurrence statistics (whether a particular transition appears or does not appear), and transition frequency statistics (how often a particular transition occurs). The former can be subsumed by the latter, however, in terms of how WordCat uses the information, it is useful to draw a distinction.

I tried various methods of spreading activation through the network at times. Attempts included varying decay rates for each depth, inhibition, recurrence, and both bottom-up and top-down feedback on varying combinations of depths and links. However, in the few configurations that were vaguely stable, the network showed a marked preference for the most frequent word, "a", regardless of the input. The only effective method was the simplest — input activation to the appropriate node(s) and cycle once. This, contrary to my original vision, limits the effect that the frequency statistics have on the output of the network. However, occurrence statistics play a critical role and frequency statistics are still used in making repair predictions (discussed below).

Activation is spread through the network by resetting the activations of the nodes, copying the node list to a temporary buffer, inputting the activations to the buffer, and systematically going through each link, finding the appropriate predecessor in the buffer and summing the affects in the original list of nodes. Again, this process is very slow but could easily be sped up by using a more efficient methods.

3.2. Recognizing Words

The process of recognition in WordCat is divided into two primary areas illustrated in Fig. 2. The left-hand box in the figure represents the initial checking of the letter string. The first two steps (encircled in light grey) are repeated for each letter in the word being processed (including initial and final spaces).

To determine "recognition predictions" WordCat inputs the current letter to the network, cycles, and reads off the subsequently active nodes. These represent the previously trained transitions, both single letters and letter pairs. WordCat then does the same for the following letter and compares the two lists with what is actually in the input. If it finds a letter to letter pair transition that is consistent with the input, it adds the letter pair to the workspace. If not, it adds nothing. This portion of the processes does not rely on the relative activations of possible transition nodes. It is an all or none prospect.

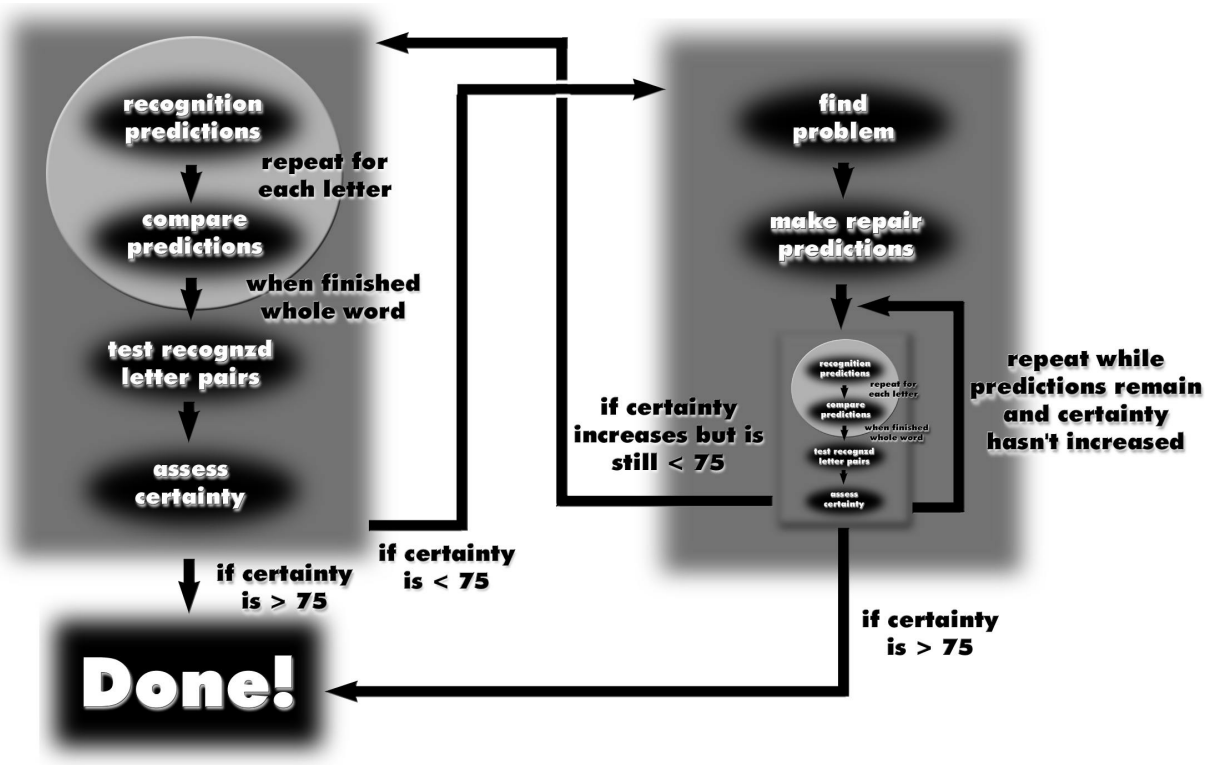

Fig. 2. WordCat's recognition decision flow chart

When WordCat finishes the entire word, the workspace has a collection of identified letter pairs that are consistent with what was observed in the training text. Converting the input text into letter pairs as individuals aids the process of recognition by emphasizing differences. For example, compare the workspace representations of `mountain` and `mouhtain` that WordCat would have after checking for letter pairs:

```
*m * mo m ou o un u nt n ta t ai a in i n* n *
*m * mo m ou o u h t ai a in i n* n *
```

In this case, a single letter difference in the input caused there to be a 3 letter pair discrepancy in the identified forms.

The letter pairs are input to the network and cycled, resulting in activations in the D3 word nodes. These words represent WordCat's "best guesses" ordered by their relative activations. At this point, if the word to be recognized is one that existed in the training text and it has not been too badly damaged, it is usually WordCat's first, sometimes second pick.

WordCat assesses the certainty of its number one best guess — a function of the activation

of the best guess, the difference in activations of the first and second best guesses, and the word length of the first and second best guesses relative to the originally input word:

```
top1fit = top1.activation - |top1.size - input.size|*10;
top2fit = top2.activation - |top2.size - input.size|*10;
certainty = top1fit + (top1fit - top2fit);
```

Certainties range from about -20 to 175. If the certainty is greater than 75 (a value that I empirically found to give efficient results) then WordCat commits itself to that answer.

However, if the certainty is below 75, WordCat tries to repair the word (the right-hand box of Fig. 2). First, WordCat tries to find the first problem area indicated by one of two things. It determines the component letter pairs of the top two letter pairs (or the best guesses with the top two activations in the case of a tie) and then finds the first identified letter pair that is not a component of the best guesses. Alternatively, WordCat finds the first single letter node that does not have an identified letter pair following it.

Once a problem spot has been found, WordCat re-determines possible letter transitions and compares the resultant letter pairs with the original word, and the component letter pairs the best guesses. Based on these comparisons, WordCat attempts to produce predictions for improving the word. If it finds no predictions, it commits to its original best guess.

Usually, a list of 1-4 predictions is found and they are ordered according to activation — i.e. by frequency of occurrence. WordCat continues to employ and assess the predictions until it runs out or until the certainty increases. This process of checking the repaired word is the same as the initial recognition (hence the left-hand box of Fig. 2. is repeated in miniature on the right). Each time WordCat tries a prediction from this list, it is on the output of the initial recognition (i.e. it is not iterative).

If the certainty goes above 75, WordCat commits to the number one best guess produced by the repaired word. If the certainty increases but remains below 75, WordCat returns the repaired word to the left-hand box of Fig. 2. to produce an entirely new set of predictions (i.e. iterative repairs). The process continues until WordCat can produce no more predictions or until the certainty goes above 75. It is seldom that more than two repairs of any type are made.

3.3. Using WordCat

When the program is run, the user is presented with 3 options:

```
Enter 1 to create, 2 to use in command line, 3 to use with file.
```

"1" creates a network and saves it in a file, `Network.txt`, based on training text found in the file `Text.txt`. "2" allows the user to enter words manually. "3" will recognize words contained in the file `2bRecognized.txt` and produce the results in the file `Recognized.txt`. Word in `Text.txt` must be delimited by an asterix (including at the end of lines), and words in `2bRecognized.txt` by spaces or the eol character.

4. Test Results

In order to test WordCat's efficacy, I arbitrarily chose 4 words of varying lengths from the training text and systematically altered them. I tested them on training sets of both 300 and 800 words and found them to be comparable — the problem areas are the same. The results before are based on the 300 word training set trials. For interest's sake, I also compare the same words with WordPerfect 7's Spell-Checker although it is impossible to draw concrete conclusions from such a comparison. The full results appear in Appendix D.

		Percent
% Correct	Overall correct (total = 55)	58%
	words with no mistakes (total = 4)	100%
	one error (total = 36)	67%
	two errors (total = 15)	27%
	small word size (total = 27)	44%
	large word size (total = 28)	71%
% of errors made (total = 23)	on words with initial errors	43%
	on words with non-initial errors	56%
	# of errors with low certainty	30%
	# of reasonable errors	78%

Of 55 words that were tested, WordCat correctly recognized 58% of them (WordPerfect - 38%). WordCat shows two classes of errors: "unreasonable" and "reasonable". By reasonable, I

mean errors in which the recognized word was of equal or closer similarity to the input text than of the intended form. For example, given the input "se" it is more reasonable to recognize "set" than the intended "seas" since "set" is in fact a closer match. As well, if the certainty of the recognized word is below the threshold of 75, I consider it a reasonable error since the system itself "realizes" in some sense that something is wrong with the result. In these case, the system's difficulty is in finding predictions.

WordCat made a total of 5 "unreasonable" errors in the test set (marked by shading in Appendix D). These errors can be generalized into two types: "input initial errors" and "subsumption errors". WordCat has a bias to make forward repairs; that is, it targets a problem area at the beginning of the word and makes a correction to the following characters. Thus typos that occur at the beginning of the word, will cause WordCat to fail.

Subsumption errors occur when the recognized letter pairs of an input word with mistakes correspond more or less exactly to a shorter word in WordCat's lexicon. For example:

infan: ***i** * **in** i nf n fa f an a **n*** n *
in: ***i** * **in** i **n*** n *

In this case, because the FamConn links have been normalized for word length, "in" will win over the intended "infant". Subsumption errors are mostly a result of the simple measure of certainty.

In general however, I consider WordCat at least as successful as commercial spell-checkers since in all cases the intended word appears high on the list of possibilities.

5. Advantages, Limitations, and Improvements

In this incarnation, WordCat has some significant problems.

speed: WordCat could not currently be implemented in realtime having been trained on a large training text with a reasonable vocabulary size.

biases: WordCat is better at recognizing larger words than smaller words and words that have fewer similar counterparts in the training text. Both of these are understandable and are endemic to the task as a whole — there is simply less information to make predictions about in these cases. WordCat also shows other unreasonable biases, towards the input initial and subsumption errors described above.

determinism: Unlike CopyCat, WordCat uses no stochastic processing which limits its flexibility.

However, there are several significant advantages to this approach.

universality: There is nothing in WordCat that is language specific. To change languages, one simply needs to change the training text — even for Asian languages, the only change that would required is to switch to double width character types.

expandability: Because WordCat works primarily with letters, the lowest level of text, there are no limits to the type of statistics that could be represented and employed. In this implementation, it was restricted to words and only letter pairs for time and flexibility restrictions. Ideally, the transition statistics of larger units, including words and phrases could be also extracted from training material. As well, based on exhibited transitions, it would be possible to propose category-like nodes that could then also be used in recording transition statistics similar to approaches to statistical linguistics exemplified by Landauer and Dumais (1997), Elman (1995), and Seidenberg (1997). Because WordCat is implemented at the letter level, this open up the possibility of better characterizations of morphology. It could potentially also characterize punctuation rules.

top-down and bottom-up processing: WordCat's processing of text occurs in both a top-down and bottom-up fashion. Although its implementation is not biologically plausible, this combination of processing in both directions is "psychologically plausible" (to borrow a term from Mitchell, 1993). It has the potential to characterize a number of phenomena exhibited in human subjects such as priming.

Immediate improvements that could be made to this system that would increase its use and efficacy, are of a few different types.

implementation: WordCat would be greatly sped up by implementing much more efficient searching and sorting algorithms.

expanding to word level: Currently, the statistics represented in the network are only letter to letter and letter to letter pair. Increasing that representation to the word level, I

feel, would allow for better recognition of small and similar words by incorporating grammatical relationships as well as graphotactic.

more prediction / error finding heuristics: the methods that WordCat uses for finding problem areas, making predictions, and assessing the certainty of best-guesses are very limited. There are dozens more criteria that could be included that would increase WordCat's performance and reduce its errors, although WordCat's architecture is currently not well suited to their inclusion.

adding stochastic processing: adding more CopyCat-like features to WordCat would improve its flexibility and robustness. For example, Codelets and the Coderack are perfectly suited to employing the heuristics that are mentioned above.

It seems significant that such a "dumb" system — simply recording transition statistics is capable of performing so well on the task of word recognition. Overall, WordCat is successful as a first step in testing the viability of employing CopyCat architectures in domains other than analogy-making. And, more generally, WordCat lends weight to the increasingly popular hypothesis that statistical relationships are important to the analysis and modeling of language.

Appendix A: Training Text

```
100
he*dreamt*of*horizons*and*seas*of*canyons*and*moons*
of*mountains*and*stars*roads*and*lanes*appeared*wherever*he*
set*his*feet*the*evening*winds*of*a*new*season*
lay*at*his*back*and*the*wide*unknown*world*opened*
her*arms*whispering*come*away*the*mole*rose*to*follow*
and*woke*beneath*a*sky*of*spring*leaves*swaying*in*
the*afternoon*breeze*with*a*smile*then*a*sigh*then*
a*shrug*he*stood*dusted*off*and*continued*on*his*
way*in*a*small*clearing*at*the*heart*of*the*
wood*stood*the*elder*tree*high*in*its*silver*branches*
200
a*caucus*of*hoary*ravens*sat*brooding*their*eyes*jet*
as*ebony*their*wings*singed*white*by*time*a*clear*
brook*welled*soundlessly*from*the*tree's*knotted*roots*flowed*down*
a*steep*incline*and*arced*smoothly*over*a*low*bank*
into*the*river*the*mole*peered*from*behind*flowered*thickets*
he*knew*the*four*winds*gathered*here*whispering*news*of*
the*world*in*a*language*only*the*ravens*understood*all*
animals*from*the*proud*wolf*to*the*timid*sparrow*came*
when*in*need*and*bowed*before*the*dark*birds*taking*
a*deep*breath*the*mole*stepped*out*onto*the*path*
300
and*lifted*a*hand*in*greeting*back*again*he*said*
bowing*a*little*they*waited*as*he*ambled*forward*into*
the*thin*shadows*of*the*tree*hope*rose*in*his*
heart*this*was*as*close*as*they*had*ever*let*
him*come*before*it*had*always*been*not*yet*and*
come*in*a*fortnight*this*time*he*told*himself*they*
must*have*an*answer*the*winds*have*brought*news*the*
weight*of*age*was*all*around*he*felt*an*infant*
to*the*ravens*and*a*mayfly*to*the*tree*as*
he*came*near*he*gazed*up*into*its*branches*and*
400
```

saw*a*net*in*which*time*itself*had*been*caught*
and*held*for*a*moment*his*consciousness*swept*into*the*
tree*as*a*leaf*is*drawn*by*a*swift*current*
into*the*heart*of*the*river*he*was*no*longer*
aware*of*ravens*nor*wind*nor*even*of*the*wood*
itself*but*only*of*himself*and*the*tree*he*felt*
the*age*of*the*world*in*an*instant*and*glimpsed*
himself*in*the*perspective*of*centuries*as*the*spell*of*
the*tree*fell*away*he*remembered*his*errand*and*stepped*
closer*still*heart*pounding*he*lifted*squint*eyes*and*whispered*
500
when*shall*i*see*her*again*after*a*pause*filled*
only*with*the*restless*news*of*the*wind*the*eldest*
raven*spoke*a*single*word*never*never*echoed*a*second*
never*tolled*a*third*and*a*fourth*for*a*moment*
the*mole*stood*motionless*waiting*as*his*heart*cracked*waiting*
for*the*fifth*raven*to*contradict*the*rest*but*the*
fifth*without*even*glancing*his*way*repeated*the*verdict*the*
last*mournful*never*fell*upon*him*bowing*his*head*he*
stood*gazing*at*the*ground*and*suddenly*felt*as*old*
as*the*tree*itself*the*ravens*of*the*brook*have*
600
spoken*the*mole*began*timidly*the*rooks*of*the*tree*
have*judged*your*wisdom*is*legend*you*are*older*than*
memory*has*the*sackcloth*council*ever*been*wrong*have*the*
ebon*elders*ever*spoken*in*haste*two*seasons*i*have*
waited*two*seasons*you*have*listened*to*the*winds*that*
are*witness*to*all*and*now*your*judgment*is*delivered*
and*the*word*is*never*but*love*he*ended*with*
a*quiet*smile*and*helpless*shrug*cannot*endure*the*word*
never*neither*sadness*nor*sympathy*shone*in*the*dark*eyes*
above*from*simple*feelings*and*brief*lives*the*ravens*were*
700
separated*by*a*gulf*of*years*they*had*forgotten*compassion*
long*before*the*mole*had*ever*been*in*need*of*
it*never*the*eldest*spoke*again*their*heads*one*by*

one*sank*between*stiff*shoulders*their*eyes*closed*into*silver*
seams*their*attention*returned*to*the*voices*in*the*wind*
which*the*mole*could*only*feel*but*never*understand*he*
lowered*his*eyes*and*slowly*turned*away*unaware*that*birds*
were*singing*or*sunlight*shining*he*followed*the*widening*stream*
wherever*it*led*rusted*hearts*and*dusted*souls*never*is*
it*set*as*many*nevers*on*my*back*as*peaks*
800
on*a*mountain*this*is*love*you*sooty*birds*love*
yet*though*by*desire*brave*he*was*by*nature*timid*
his*frame*was*not*built*to*sustain*courage*and*he*
walked*on*in*whispers*hushed*by*doubt*i'll*wait*out*
the*world*for*you*love*i'll*extend*my*tunnels*to*
the*horizon's*ring*i'll*reach*for*you*as*the*tide*
reaches*for*the*moon*but*his*brave*promises*sounded*too*
brave*he*felt*suddenly*small*and*the*word*never*whispered*
in*his*mind*as*the*ocean*in*a*shell*falling*
slowly*to*his*knees*he*wept*without*a*sound*by*

Appendix B: Training text size and network size

Training text size	Nodes	SuccLinks	FamLinks
100	291	431	808
200	440	680	1558
300	547	802	2094
400	657	914	2582
500	733	1003	3070
600	818	1113	3654
700	896	1184	4196
800	990	1255	4648

300 and 800 were the networks that were used to assess WordCat's performance.

Appendix C: Files Included

`Wordcat.pdf`	- this document
`Wordcat.exe`	- the program
`Wordcat.cpp`	- main source code
`Makecat.cpp`	- source code that makes the network
`Usecat.cpp`	- source code that employs the network
`Network.txt`	- the network file that WordCat makes / loads
`Networkxxx.txt`	- network file trained on a training text of xxx words
`Text.txt`	- the training text file WordCat uses to make the network
`Mole&Owl.txt`	- full text used to train WordCat (Appendix A)
`2bRecognized.txt`	- file used as input for WordCat
`wclog300.txt`	- log of the run of the 300-network trials

Appendix D: Test Results: 300

	not				seas			
manipulation	**form**	**recognized**	**certainty**	**WP7**	**form**	**recognized**	**certainty**	**WP7**
original	not	not	177	not	seas	seas	154	seas
missing first letter	ot	on	178	of	eas	as	62	east
missing first 2 letters	t	to	168	t	as	as	158	as
missing last letter	no	to	178	no	sea	seas	124	sea
missing last 2 letters	n	to	168	n	se	set	140	se
missing center letter	nt	to	178	--	sas	sat	144	--
missing 2 central letters	N/A	N/A	N/A	N/A	ss	seas	40	ss
changed first letter	got	it	23	got	geas	as	52	gas
changed central letter	nyt	not	77	not	syas	seas	154	says
changed last letter	noh	not	75	no	seah	seas	80	sea
changed 2 letters	nyh	new	150	--	syah	sigh	112	shah
extra final letter	notr	not	115	not	seasr	seas	93	sear
extra initial letter	jnot	jet	117	not	jseas	as	22	seas
extra internal letter	nbot	not	157	not	sebas	seas	110	seas

Test Results: 300 con't

	infant				mountains			
manipulation	**form**	**recognized**	**certainty**	**WP7**	**form**	**recognized**	**certainty**	**WP7**
original	infant	infant	174	infant	mountains	mountains	172	mountains
missing first letter	nfant	not	107	enfant	ountains	mountains	140	fountains
missing first 2 letters	fant	felt	140	fanned	untains	mountains	90	entwines
missing last letter	infan	in	77	infant	mountain	mountains	134	mountain
missing last 2 letters	infa	in	107	Inca	mountai	mountains	90	mountain
missing center letter	infnt	infant	86	infant	mounains	mountains	84	mountain
missing 2 central letters	ifat	it	79	fiat	montins	moons	96	montan
changed first letter	gnfant	infant	120	enfant	gountains	mountains	150	fountain
changed central letter	inyant	it	61	infant	mouytains	mountains	114	mountain
changed last letter	infanh	infant	116	infant	mountainh	mountains	170	mountain
changed 2 letters	iyfaht	infant	116	--	moyntahns	moons	40	montan
extra final letter	infantr	infant	134	infant	mountainsr	mountains	160	mountains
extra initial letter	jinfant	infant	112	infant	jmountains	mountains	160	mountains
extra internal letter	infbant	infant	134	infant	mounbtains	mountains	124	mountains

Test Results: 800

	not				seas			
manipulation	**form**	**recognized**	**certainty**	**WP7**	**form**	**recognized**	**certainty**	**WP7**
original	not	not	144	not	seas	seas	144	seas
missing first letter	ot	or	158	of	eas	as	62	east
missing first 2 letters	t	the	104	t	as	as	158	as
missing last letter	no	no	158	no	sea	seas	134	sea
missing last 2 letters	n	no	148	n	se	see	90	se
missing center letter	nt	no	158	--	sas	saw	150	--
missing 2 central letters	N/A	N/A	N/A	N/A	ss	seas	40	ss
changed first letter	got	not	77	got	geas	as	52	gas
changed central letter	nyt	net	150	not	syas	seas	144	says
changed last letter	noh	now	144	no	seah	seas	80	sea
changed 2 letters	nyh	now	144	--	syah	sigh	112	shah
extra final letter	notr	not	134	not	seasr	seas	93	sear
extra initial letter	jnot	jet	117	not	jseas	as	22	seas
extra internal letter	nbot	net	90	not	sebas	seas	114	seas

Test Results: 800 con't

	infant				mountains			
manipulation	**form**	**recognized**	**certainty**	**WP7**	**form**	**recognized**	**certainty**	**WP7**
original	infant	infant	172	infant	mountains	mountains	122	mountains
missing first letter	nfant	not	124	enfant	ountains	mountains	74	fountains
missing first 2 letters	fant	infant	34	fanned	untains	mountains	55	entwines
missing last letter	infan	in	77	infant	mountain	mountain	128	mountain
missing last 2 letters	infa	in	107	Inca	mountai	mountain	84	mountain
missing center letter	infnt	infant	86	infant	mounain	mountain	94	mountain
missing 2 central letters	ifat	it	118	fiat	montin	moon	94	montan
changed first letter	gnfant	infant	87	enfant	gountain	in	126	fountain
changed central letter	inyant	it	61[1]	infant	mouytain	mountain	126	mountain
changed last letter	infanh	infant	116	infant	mountaih	mountains	94	mountain
changed 2 letters	iyfaht	it	98	--	moyntahn	mountain	104	montan
extra final letter	infantr	infant	134	infant	mountainsr	mountains	92	mountains
extra initial letter	jinfant	infant	112	infant	jmountains	mountains	94	mountains
extra internal letter	infbant	infant	134	infant	mounbtains	mountains	124	mountains

References:

Elman, J. (1995) Language as as dynamical system. In R.F. Port & T. van Gelder (eds.). *Mind as Motion: Explorations in the Dynamics of Cognition*. Cambridge, MA: MIT Press. 195-223.

Landauer, T., and S. Dumais. (1997) A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211-240.

Mitchell, M. (1993) *Analogy-Making as Perception*. Cambridge, MA: MIT Press.

Seidenberg, M. (1997) Language Acquisition and Use: Learning and Applying Probabilistic Constraints. *Science*, 275:1599-1603.