Daniel Gindikin

Pierre Ponce

cs224n/linguistics237

Final Project

Automatic Mnemonic Device Generation

# Introduction

Mnemonic devices are useful memory aids that can be applied to many different aspects of daily life. There are always certain tidbits of information that people just can’t seem to remember in the form they are presented. People rely on mnemonics when they create some association between that information they wish to remember and other concepts that they already know, or find easier to remember. There are countless types of associations possible for fulfilling this purpose. People will tend to rely on the technique they feel will more easily enable them to recall the information they want at a later time.

Concepts of statistical natural language processing can be used in creating a system that will automatically generate mnemonic devices. This seems like a natural application for a subject that relies on analyzing the properties of language, especially since some mnemonics are used for the express purpose of expanding one’s vocabulary. The system as implemented creates mnemonics for two distinct types of information: 1. Numerical data and 2. Lexical data. Each mnemonic is represented as an utterance that is generated to present data in a way that might help the user in recall.

# About Mnemonics

The use of mnemonic devices dates back to the fifth century, BCE. It is recorded that Simonides of Ceos used a technique that associated specific items with locations that were well-known to the person recalling the information [1]. The effectiveness of this method of memory recall has been confirmed through a number of studies where subjects link new information to different locations [2,3]. Similar improvements in recall have been shown to materialize by using music as a mnemonic device [4].

The use of mnemonics has also been shown to be of significant value in the field of education. Manalo has shown that instruction using a specific type of mnemonics known as “process mnemonics” produced improvements of mathematical ability in students classified as learning disabled [5]. Process mnemonics are used specifically for remembering rules and procedures.

The mnemonics used in this system are used to allow the user to easily remember numerical and lexical information. Mnemonics for these types of information can be generated through the use of key NLP concepts. Sentences can be created that exploit the properties of items to be remembered. A classic example of numerical information is the collection of digits found in the value of . This system can produce sentences whose word lengths are determined by the digits to be remembered.

Lexical information can be coded by using the letters of an unknown word and forming a sentence whose words’ first letters match those letters. The sentence would then include one or more clue words that hint at the meaning of the unknown word. The following examples illustrate the two types of mnemonics produced by the system:

****: And I know, these daughters of Canaan shall see.

**3 1 4 1 5 9 2 6 5 3**

quiescence: Quickness Us Into Everlasting Stillness

# Why Mnemonics Work

Mnemonics work by using the associational nature of the human memory system, as well as the differences in the ease with which various things are remembered based on their semantic categorization and relatedness. There are many various types of mnemonics depending on what is being memorized, but they all usually leverage the relative ease with which people remember spatial or natural language information, as well as the ability to guide the recall using various constraints. For example, people are not very good at memorizing ordered sequences of unrelated objects; for instance stars are classified by their spectra into seven categories in order of decreasing temperature: O, B, A, F, G, K and M. Rather than memorizing this unrelated sequence of letters its much easier to remember the mnemonic “oh be a fine girl, kiss me”, where the first letter of each word gives the desired sequence.

# Mnemonic Schemes Implemented

We implemented generation of two different types of mnemonics, both using the same algorithm. The first one was for memorizing numbers, where each individual digit is encoded in the length of words of an utterance, with zero coded for by any word longer than nine characters. For instance the speed of light, 299,792,458 m/s, can be encoded as ‘He certainly presented himself incapable of such cruel behavior’ (this particular example was generated using the Jane Austen training corpus).

The second scheme was to assist with memorization of vocabulary words. We only considered words whose meanings could be captured by a single synonym, such as ‘obstinacy == tenacity’. We further cherry-picked only those words that contained the first letter of the synonym word, ‘obstinacy’ for example contains ‘t’. We then formed an utterance that ended with the synonym word, where the first letters of each word formed a prefix of the word being defined. For our ‘obstinacy’ example, ‘oh but such tenacity’ is such an utterance (also generated from the Jane Austen corpus). If the synonym letter came too late in the word being defined, such that we felt that the mnemonic would too long to be useful, we tried to put the synonym word earlier, hoping that the beginning of the sentence would be enough to help the person remember it, e.g. for ‘aphorism == maxim’, ‘all people have one maxim’.

# Corpora

The text sources used to extract the words necessary to build sentences were based from four collections of text: 1. Novels from Jane Austen, 2. Works from William Shakespeare, 3. Excerpts from The Wall Street Journal, and 4. The Bible. These documents, especially the excerpts from The Wall Street Journal, required extensive prefiltering to remove utility information that was not useful in creating the language model necessary to generate the sentences for the mnemonic devices. Punctuation marks were not completely removed from these sources since they proved to be valuable in allowing the system to produce sensible sentences using the language model. Since the language model relied on frequency bigram occurrences to quantify the likelihood that random string sequences were parts of sentences, the punctuation marks provided additional information to the system.

The system worked better when there were many total words processed from the sources. Although the number of unique word token did not necessarily increase rapidly as the source of text became large, the number of unique bigrams increased and made the training data somewhat less sparse. This improved the occurrence rate of valid sentences appearing at the output of the system.

# Training

The ability of the system to generate good mnemonic devices relies on proper training that enables the system to recognize the proper structure of valid sentences. The language model used for this system required only a basic count of the occurrences of bigrams in different text corpora. These counts, produced a collection of probabilities that characterized the general properties of the corpora processed. Obviously, using different corpora can produce training sets that are possibly very different from each other in terms of lexical content and word pair relationships. Common word pairings found in the Bible might not appear with the same frequency as those found in a Jane Austen novel. For example, a large portion of sentences in the Bible start with the word “And,” which is rare in any of the other text sources. Such differences alter the character of the sentence generated by the system.

Due to the possibility of word pairings never occurring in the text processed, a uniform small probability was assigned to unseen bigrams. We found that smoothing was not very important for our application, we controlled utterance generation, and our corpora was rich enough to almost always allow us to use previously seen bigrams, even when subject to the constraints of the mnemonic being generated; this was highly desirable as it increased the likelihood that the utterance generated would make sense.

# Language Model

The language model used to generate the sentences should emulate the general structure of the words found in the text corpora. Such a model is derived from examining the occurrences of word pairs in the text sources. A probabilistic model is derived from those counts and used in the sentence generation process. The language model is used to assign probability values to sequences of word combinations that appear in the lattice. The use of the Viterbi algorithm, with beam search, allows the system to discriminate and eliminate those word sequences that are deemed improbable according to the language model. The use of a simple bigram model is amazingly effective in generating a collection of words that compose a sentence. Previous work has shown that such a method will give comparable results when compared to higher order n-gram models [6].

The use of a bigram model also makes it likely that the mnemonic devices will replicate small segments of the text found in the training data. This is especially valuable if the user selects a text source that is very familiar. The mnemonic devices generated by the system may be easy for the user to recall once memorized.

# Lattice

We first turn the mnemonic into a sequence of constraints. For example, let’s say we wanted to find a mnemonic for the first three digits of pi, ‘314’ (an example of would be ‘and I said’). The constraints for our search are that the first word is three letters, the second one, and the third four. We can now form a lattice:

#### Thy

Who

Yet

Now

All

Let

But

And

# For

Give

# Know

Said

Have

Pray

,

a

### I

Any path through this lattice corresponds to an utterance, such as ‘Who I know’, or ‘For a give’, which correctly encodes ‘314’, though some of them may well be grammatically ill-formed and some will make no sense.

# Viterbi

## We next use our language model to select the most likely path, which will hopefully be a well-formed natural language utterance that a person will find easy to remember. Because our lattice in this case is far from a freeform graph, we used its structure to speed up computation. For this simplest of cases, where the lattice is simply a table of word lists, we did the obvious thing. We kept the N most probable paths in a priority heap, and processed the levels of the lattice one at a time, trying each potential word with each path so far, scoring the new extended paths and remembering the best N of those. We avoided generating the lattice a priori, instead having a list of constraints along which our search advanced. Given the lexicon, our current constraint (e.g. the word must begin with ‘a’, have three syllables, and rhyme with ‘-ing’) would return to us a list of candidate words.

## Each path is assigned the probability using our language model. We use the usual Markov conditional independence assumption, thus given a path (*w1,w2,…,wn)* its probability is given by

**P***(w1,w2,…,wn)=***P***(w2|w1)\****P***(w3|w2)\*…\****P***(wn|wn-1)*

for a bigram model, which is what we usually ended up using.

The algorithm became more complicated when we started dealing with sentences. In our number encoding scheme, we counted punctuation as coding for the digit one (e.g. ‘but, said I’ would correspond to 3141), but we wanted to be able to insert the ‘.’ token anywhere, so as not to have incredibly long sentences and, in general, to give the algorithm more flexibility in finding likely utterances. The complication came because now, if the paths “Why , that I would usurper’s” and “But I have a heart .” were in our priority heap, we could not treat them in the same way. The first advanced in our list of constraints and needed a word of length 2, the other did not, and needed a word of length 9. In this case we split our priority heap by which paths advanced along in satisfying a constraint and which did not, recursed separately on each part, combined the results from the two recursive calls and returned the best N.

The two final caveats were that we pretended that all paths in the lattice always started with ‘.’ so that we did not get utterances that began in the middle of sentences; also our bigram model clearly did not have enough context to accurately model the sentence lengths, so we would frequently get “sentences” of length 2 or 3. We added a simple check to prevent that from happening. Here is a typical result for N=10, this is for the first few digits of *e=*2.71828183, with the Jane Austen corpus:

In October , director of Sanditon . I remember the

In October , director of analysts ' interest . The

In October , director of analysts ' earnings . The

In October , director of analysts ' chairman . The

In October , director of analysts ' business . The

In October , director of analysts ' Paradise The

In October , director of analysts ' estimate for

In October , director of analysts ' earnings and

In October , director of analysts ' estimate the

In October , director of analysts ' chairman and

Initially we had the idea that for the vocabulary memorization task, we would generate not just simple utterances, but rhymes. Syllabic counts and rhyming lines of structured poetry can be viewed as simply additional constraints for the search, for example a limerick must have 5 lines, the first two and the last with 8 syllables, the 3rd and 4th with 5, with the rhyming scheme AABBA. That is the reason why we came up with such a flexible and general system of constraints, which may seem like overkill for simpler utterances we ended up generating. We dropped the idea, partly because of lack of time, but also partly because we felt that a limerick, or any other reasonable rhyme would be too long to be useful in memorization.

In qualitatively evaluating our results, we discovered that sometimes, when the algorithm came up with about 20 different utterances, it was usually possible to find among them something sensible. However, frequently some path prefix would be scored highly likely, and would come to dominate our priority queue; all the paths in the priority queue would start with it. If it was non-sensical (e.g. ‘And I will I’), as was frequently the case, none of the resulting utterances would be good candidates for a mnemonic. To address this problem, we made the search interactive, allowing the user to select at each stage of the search which of the paths in our priority queue to keep.

# Interactivity

This system was designed to generate sensible mnemonic devices based on certain specifications provided by the user. This seems to be a logical condition for usage due to the nature of the application: to produce a memory aid that is easy to remember. A user might want to generate a mnemonic device from a collection of words that are commonly themed or come from a specific text source. For example, it is sensible to remember a word like “amendment” by using words from political texts. This is a way that the system can be customized to the preference of the user.

The system outputs a set of possible mnemonics that match the specifications of the item to be remembered and the user’s preference in sentence lexicon. This allows the user to subjectively decide which generated sentence will be easier to remember and recall (and decipher) later. A larger number of sentences also precludes the possibility of all sentences being invalid.

# Conclusion

In this project we created an automatic mnemonic generator. One of the most interesting things we discovered, was that the part of problem we thought would be the hardest -- generating good, natural language text, turned out to be surprisingly simple. Initially we thought that we would have to deal with things like word morphology, tense agreement, and conjugation, but we discovered, as stated in [6], that even a simple bigram model turned out to be sufficient.

We had less success than we hoped with the qualitative results. While it was certainly amusing to see the system generating utterances like ‘Every moving thing that liveth shall be meat for thy’ when trained on the bible corpus, we discovered that in fully automatic mode, the system would rarely come up with good mnemonics. On the other hand, we found that the computer can help in the mnemonics generating process, helping with the parts of the task that humans find hard, by using the stochastic language model to suggest likely words of length 9 or those that start with the required letter. We believe that, with some improvements, our vocabulary words system can be useful in assisting people in coming up with their own mnemonics. Our background research suggests that when people spontaneously generate their own devices, the recall is improved because people tend to come up with utterances, or other forms of encoding, that are personally salient to them. While we do not suggest that people should generate mnemonics for all the words they are attempting to memorize, we believe that it is useful to do so for those few that, due to the quirks of human memory, people repeatedly have trouble remembering.

In developing this system, we have come to the realization that generating really good mnemonics is an AI-complete problem. Mnemonics leverage the very human quality of making connections and associations between things, and it is precisely this common sense knowledge and understanding of what in the world is so connected and associated that computers lack. Still, using statistical techniques, the computers can begin to mimic such knowledge, and become useful in offloading parts of the task from people.

# References

[1] Wang, A.Y. & Thomas, M.H. (2000). Looking for long-term mnemonic effects on serial recall: The legacy of Simonides. *American Journal of Psychology, Vol. 113*, pp. 331-340.

[2] Ross, J. & Lawrence, K.A. (1968). Some observations on memory artifice. *Psychonomic Science, Vol. 13*, pp. 107-108.

[3] de Beni, R. & Cornoldi, C. (1988). Does the repeated use of loci create interference? *Perceptual and Motor Skills, Vol. 67*, pp. 415-418.

[4] Wilson VanVoorhis, C.R. (2000). Stat Jingles: To Sing or Not To Sing. *The Annual Meeting of the Midwestern Psychological Association*, Chicago, IL.

[5] Manalo, E. (2001). Uses of Mnemonics in Educational Settings: A Brief Review of Selected Research. Invited lecture at Kyoto University.

[6] Langkilde, I. & Knight, K. (1998). The Practical Value of n-grams in Generation. *Proceedings of the 9th International Natural Language Workshop (INLG '98)*, Niagara-on-the-Lake, Ontario.