Using a Symbolic Language Parser to Improve Markov Language Models

by

Duncan Clarke McIntire Townsend

S.B. Massachusetts Institute of Technology (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2015

Author . . . .
Department of Electrical Engineering and Computer Science
January 30, 2015

Certified by . . . .
Boris Katz
Principal Research Scientist
Thesis Supervisor

Accepted by . . . .
Prof. Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

Using a Symbolic Language Parser to Improve Markov Language Models

by

Duncan Clarke McIntire Townsend

Submitted to the Department of Electrical Engineering and Computer Science
on January 30, 2015, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Electrical Engineering

Abstract

This thesis presents a hybrid approach to natural language processing that combines an n-gram (Markov) model with a symbolic parser. In concert these two techniques are applied to the problem of sentence simplification. The n-gram system is comprised of a relational database backend with a frontend application that presents a homogeneous interface for both direct n-gram lookup and Markov approximation. The query language exposed by the frontend also applies lexical information from the START natural language system to allow queries based on part of speech. Using the START natural language system's parser, English sentences are transformed into a collection of structural, syntactic, and lexical statements that are uniquely well-suited to the process of simplification. After reducing the parse of the sentence, the resulting expressions can be processed back into English. These reduced sentences are ranked by likelihood by the n-gram model.

Thesis Supervisor: Boris Katz
Title: Principal Research Scientist

Acknowledgments

I would like to thank my advisor, Boris Katz, for his tenacious support of me not only through my M.Eng. thesis, but also through my undergraduate education. I would also like to thank Sue Felshin for her guidance and patience with me as I learned how to use the systems employed by the MIT CSAIL Infolab Group. Finally, I would like to thank my parents for their warmth and positive demeanor as they supported me through both my undergraduate and graduate careers at MIT.

Contents

List of Tables

1. Introduction

Markov language models, also known as n-gram language models, are a popular and effective method of measuring the relative correctness of natural language phrases. N-gram models are parameterized by the "Markov window", the number of words in sequence that the model captures the probability of before resorting to Markov approximation. If the Markov window is set too large, the model suffers from sparsity. Sparsity happens when the training data set is too small to accurately represent the probability of uncommon word sequences. With a small training data set, many possible word sequences will not appear in the training data. As so-called "big data" approaches to machine learning become more popular, training data sets become larger and so larger Markov windows become appropriate. There is, however, a limit to this approach. Training data sets need to become exponentially larger as Markov windows grow. Additionally, storage and indexing requirements grow exponentially with the Markov window, making n-gram models with large n inappropriate for environments with constrained computing resources. Sparsity is frequently addressed using a backoff or smoothing method. Among the most commonly used are additive smoothing, [2] Good-Turing estimation, [4] and Katz backoff [9].

The n-gram model used in this thesis uses a Markov window of 5. This means that word dependencies that span more than 5 words cannot be captured by the model. The result of this thesis is a model that can capture word dependencies with much larger spans without increasing the size of the model's database or the training data set.

This model combines the statistical approach to language as embodied by an n-gram model with a symbolic, grammatical approach to language as embodied by Boris

Katz's START natural language parser. [5, 6, 7, 8] The grammar-driven approach provided by START can improve the n-gram model by allowing its Markov window to "stride" over modifiers and dependent clauses. The statistics-driven approach of the n-gram model can correct unusual phrasing that prevents a match against START's knowledge base. To demonstrate this, two systems have been built on a common framework. A system of intelligent n-gram backoff demonstrates the ability of a symbolic approach to improve a statistical one. A system that ranks the relative correctness of alternative phrasings and chooses the most correct one demonstrates the ability of a statistical approach to improve a symbolic one.

This combined symbolic–statistical approach improves the statistical model without resorting to ever-larger training data sets and databases, and improves the symbolic model's permissiveness, which has historically been a limitation of those types of approaches.

This thesis will address the technical challenges of constructing an n-gram database with a large training corpus and large n. First, it will discuss the architectural challenges of building a query language for an n-gram database that naturally expresses the kinds of queries that are most useful. Next, the START natural language parsing system and its ternary expression internal representation will be described. Finally, this thesis will present the details of an intelligent backoff system combining both the START natural language parser and the n-gram database that is capable of analyzing longer and more complex structures than an ordinary n-gram model.

2. Building an n-gram Database

An n-gram database ought to have several key qualities:

- a query language that naturally expresses the kinds of equivalences that are present in natural language
- a homogeneous interface across both direct n-gram lookup and Markov approximation
- a "high recall" mode that accepts a generalized n-gram pattern and returns all specific n-grams that match

To easily express the equivalences in natural language, the database should store the canonical form of each n-gram that encapsulates the differences between n-grams that are relevant to a native speaker. Having a homogeneous interface for both lookup and Markov approximation requires that a frontend application be built that exposes a query language independent of the backend representation of the n-gram data. This query language must also support some form of pattern matching and generalize n-gram syntax to enable a "high recall" mode. The system constructed for this thesis supports all three of these qualities.

2.1 Backends

2.1.1 MongoDB

This thesis uses the Google Trillion Word Corpus [1] as its data set. The Google Trillion Word Corpus consists of the count of each 1-, 2-, 3-, 4-, and 5-gram that appeared in a trillion words scraped from the internet by the Google Machine Translation Team in January 2006. In the initial work performed for this thesis, these n-gram/count pairs were formatted as JSON and stored in a sharded MongoDB database. This was an initial success because MongoDB's relatively free-form JSON document structure lent itself well to expressing the concept of a canonical form of each n-gram.

Although MongoDB's document structure flexibility was great for constructing certain kinds of queries, the expressiveness of MongoDB's query language was lacking. The lack of joins in MongoDB's query language forces the front-end application to perform many sub-queries and aggregation steps to perform the approximation. Since the aggregation steps are being performed in the front-end application, they are less efficient in both processing time and memory usage. Since each sub-query requires a network round trip, they are also slow.

The second major problem with MongoDB was a result of MongoDB's lack of durability guarantees. This first manifested after attempting to perform a backup and restore of the MongoDB database following a migration to a new machine. It was discovered that the newly restored database had fewer documents stored in it than the source material. After investigation, this was attributed to both a failure to commit documents as they were being created and a disturbing phenomenon where the read-only MongoDB collections lose documents over time. A stop-gap solution was constructed that involved initially checking the collections for missing documents and then regularly restoring the database from a known-good backup.

2.1.2 PostgreSQL

Ultimately, the issues with MongoDB's lack of relational joins and durability outweighed the benefit of MongoDB's flexible document structure and easy sharding. During Markov approximation, n-gram data is highly relational. MongoDB's lack of joins and any other features supporting a relational dataset made it unsuitable for storing the n-gram data. A second iteration of the database backend was in order. After some work to separate the front-end query language from the backend database interface, a second backend interface was created based on PostgreSQL.

Although there were initial difficulties creating a schema that accurately reflected the kinds of canonicalization that most queries require while still being efficiently indexable, the ability to use joins in queries proved to be a great boon. PostgreSQL's durability guarantees also removed the problems faced with MongoDB. The current form of the database has completely removed support for MongoDB because its maintenance was too costly for it to be worthwhile.

2.2 Query Language

Although the structure of the database backend was a major technical challenge, the API presented to the end-user was a more interesting architectural challenge. Initially, the interface replicated a legacy system built by Yuan Shen of the CSAIL Infolab group. Shen's work was intended as a resource for comparing possible spelling corrections for a particular word in fixed context. Although his interface was excellent for that purpose, it lacked the flexibility of a more general-purpose system. To that end, an API was developed that exposed a uniform interface for directly looking up n-grams in the database and for performing Markov approximation.

2.2.1 Query Features

This API took a sequence of whitespace-separated words and a set of canonicalization parameters and returned all matching n-grams and their counts. For n-grams with length larger than the widest Markov window (5), the counts were replaced with the Markov approximation of the frequency. Initially, the query language engine was a very thin front-end for the underlying database, but as it was used, more features were added.

The first few features were a heterogeneous collection of functionality that other members of the CSAIL Infolab group found useful. The first feature added to the query language was the automatic conversion of words not found in the Google Trillion Word Corpus lexicon to the special token "`<UNK>`". "`<UNK>`" in the Google Trillion Word Corpus represents words with frequency too low to appear in the lexicon. The next feature added was the ability to specify an n-gram prefix and obtain all n+1-grams with that prefix. This is useful in predictive text and autocorrection. This was later generalized by adding a "`<WILD>`" special token that matches all words. By using a "`<WILD>`" token at the end of a sequence, it is possible to query based on sequence prefixes.[1]

A more restrictive form of "`<WILD>`" was developed in response to the use case where after making a query containing "`<WILD>`", the results were then filtered using some lexicon. The special tokens "`<ADJECTIVE>`", "`<ADVERB>`", "`<NOUN>`", "`<PREPOSITION>`", and "`<VERB>`" were added, representing any word that can be that part of speech, according to START's lexicon.[2]

As a later enhancement, "`<NOUN>`" was given attributes such as gender, number, and proper/common that can be used to make it more restrictive. Similarly, "`<VERB>`" was given attributes for the five possible conjugations of a verb in English: "`past`", "`present-participle`", "`past-participle`", third person singular "`present`", and "`other`".[3]

2.2.2 Query Structure

The initial version of the API assumed that all queries were whitespace-separated sequences of words. This assumption was a poor one and introduced bugs from two sources. First, special tokens like "`<VERB present>`" contain whitespace. Second, the tokenization performed on the corpus before generating the n-grams was not simply whitespace-splitting. This second bug turned out to be the more complex one. According to the documentation supplied with the Google Trillion Word Corpus, tokenization was performed with a tokenizer similar to the one used in the Penn Treebank [10] Wall Street Journal section. After examination, however, the data in the n-gram database does not reflect the use of a tokenizer as described. Rather

[1]For example, the 3-gram query "`John <WILD> Mary`" matches both 3-grams "`John and Mary`" and "`John loves Mary`".

[2]Similar to the previous example, the 3-gram query "`John <VERB> Mary`" matches "`John loves Mary`" but not "`John and Mary`".

[3]The 3-gram query "`John <VERB present> Mary`" matches "`John loves Mary`" but not "`John loved Mary`".

than attempt to imperfectly replicate the tokenizer that was used on the original corpus, an interface was exposed that allows the user to explicitly tokenize their own sentences from strings into token sequences. This interface takes JSON-encoded arrays of strings and returns the appropriate n-grams and counts. Future work may include a tokenizer, but it is beyond the scope of this thesis to attempt to replicate the one used in the Google Trillion Word Corpus.

3. START

START [5, 6, 8] is a natural language system that models English syntax and semantics. In addition to its use as a natural language parser, START contains a commonsense knowledge engine that is closely coupled to its understanding of English semantics. START's primary application is to natural language question answering. The interface at `http://start.csail.mit.edu/` differs from conventional search engines in the kinds of responses START produces. START compares user questions to the information in its knowledge base to provide *direct* answers to those questions, instead of keyword matches as seen in conventional search engines. START's question-answering capability is not the capability that this thesis makes use of, however. START's English-language parsing and generation capabilities are the focus of this work.

3.1 T-expressions

START's internal language representation consists of a collection of ternary expressions (abbreviated as T-expressions or texps). Ternary expressions are 3-tuples in the form:

```
[subject relation object]
```

In this thesis, T-expressions will be written between square brackets and with whitespace separating the 3 elements (subject, relation, and object). The subject and object may be either T-expressions or atoms; the relation must be an atom. Atoms are either global constants (proper nouns, some special relations, and other singleton entities), or instances (actions, common nouns, properties, modifiers, etc.). Global constants are represented in this thesis by the name of the constant. Instances are represented in this thesis by the name of the instance and a unique base-10 number distinguishing that instance from all other instances. The name of the instance and the unique number are joined together with a "+" character to make the representation of the instance.

The simplest T-expressions are those such as `[cat eat+1 fish]`, expressing the action "eats" being performed by the agent "cat" on the object "fish". More complicated T-expressions can have "nested" subjects or objects, such as `[John know+1 [Mary like+2 cake+3]]`, which expresses "John knows that Mary likes cake".[1] START's T-expression language is much richer than this, however.

T-expressions can be broken down into 3 broad categories. Structural T-expressions describe the semantic relationships between the words that compose a sentence without regard to their inflection or part of speech. Syntactic T-expressions describe the inflection and syntactic roles of the words in the sentence and serve to disambiguate word usage. Lexical T-expressions provide information about word definitions and parts of speech. This thesis is primarily concerned with altering the structure of sentences by manipulating the structural T-expressions, although it makes use of syntactic T-expressions to find alternate inflections and uses lexical T-expressions to identify structural T-expressions that may be of interest.

Structural T-expressions come in two major forms. T-expressions in the form:

[*subject verb object*][2]

form the "backbone" of a sentence and express the main actions. There may be more than one T-expression in this form, but all but one of them will be subordinated by some other relation. T-expressions may also have the form:

[*subject special_relation object*]

In this case, *special_relation* is one of a fixed list of special structural relations.[3]

[1] Additional lexical T-expressions are required to disambiguate tense and the use of "that".

[2] *verb* is always written in the infinitive; other, syntactic T-expressions provide information about its inflection. If *verb* is intransitive, then *object* will be the constant `null`.

[3] The special structural relations are `has_effect`, `has_intensifier`, `has_method`, `has_modifier`, `has_quantity`, `has_property`, `has_purpose`, `has_rel_clause`, `is`, `is-a`, and `related-to`.

The meaning of these special structural relations depends on the relation.[4] For example, the special structural T-expression

`[cat+1 has_property fluffy+2]`

tells us that the noun "cat" has an adjective modifier "fluffy". So the noun phrase is "fluffy cat".

Syntactic T-expressions are all in the form:

[*subject special_relation object*]

Because these T-expressions control the inflection of words, they have an effect on the corresponding sentence. Although there are many syntactic T-expressions, we are primarily concerned with those that control the inflection of nouns and verbs.[5] As with structural T-expressions, the meaning of the special syntactic relations is dependent on the relation in question.[6]

Lexical T-expressions are also all in the form:

[*subject special_relation object*]

Lexical T-expressions do not always have an effect on the corresponding sentence, and are sometimes included in parses as supplementary information. Similar to syntactic T-expressions, there are many special lexical relations. However, only relations that assist in the identification of subordinate clauses are of any concern. These are the `is_clausal` and `has_category` relations.[7]

[4] `has_effect`, `has_method`, `has_purpose`, `has_rel_clause`, and `related-to` join two other T-expressions together. `has_intensifier`, `has_modifier`, `has_quantity`, and `has_property` apply simple modifiers to words. `is` and `is-a` represent verbs of being (am, are, be, been, being, is, was, were) and are similar in use to ordinary verbs, but are special-cased because of the complicated behavior of verbs of being.

[5] These syntactic relations are `has_det`, `is_perfective`, `is_progressive`, `has_modal`, and `has_position`.

[6] `has_det` controls which determiner a noun has, if any. `is_perfective` and `is_progressive` control the perfective and progressive aspects of verbs. `has_modal` specifies the aspectual modal, if any. `has_position` controls the position of a subordinate clause relative to its subject.

[7] `is_clausal` is used to mark clausal verbs and subordinating conjunctions. `has_category` is used to mark the part of speech of any word that may be ambiguous.

4. Intelligent Backoff

4.1 Justification

Markov approximation gives the frequency of n-grams of any length that could have appeared in the corpus (up to the Markov window). However, due to the problem of sparsity (as discussed in the Introduction), there are possible utterances that do not appear in the database simply because they did not appear in the training data (or in the case of the Google Trillion Word Corpus, did not have frequency above 40). It would be useful to be able to evaluate the relative likelihood of these n-grams, even at low accuracy.

To that end, a system of intelligent backoff was implemented for n-grams with length beyond that of the Markov window. This intelligent backoff encapsulates the idea that although the vast majority of English-language word dependencies are captured by the Markov window of 5 [3], as seen in Table 4.1, there are still a significant minority that are not. The backoff system shortens sentences by removing modifiers and dependent clauses, splitting independent clauses from each other, and altering the inflection of the words in the sentence.

Table 4.1 English Word Dependency Locality

Markov window	2	3	6	10
dependencies captured	74.2%	86.3%	95.6%	99.0%

from Collins, *A New Statistical Parser Based on Bigram Lexical Dependencies*[3]

4.2 Simplification

In this method, backoff effectively increases the length of the Markov window by removing modifiers and subordinate clauses. After the intervening modifiers and subordinate clauses between two words that share a dependency have been removed, the Markov window can capture that dependency and give an estimate of how likely that dependency is. By selectively deleting T-expressions from the parse of the sentence, the system trims these intervening elements and simplifies the sentence. The simplification process produces a set of possible simplifications of the given sentence. Many of these simplifications are so reduced as to be almost meaningless, while others have awkward phrasing. Only the combined symbolic–statistical approach can eliminate those simplifications that would never occur in normal English.

4.2.1 Parsing

The first step of simplification is to parse the raw sentence into a set of T-expressions that represent it. It is on this set of T-expressions that the remainder of the simplification process operates. START takes a whitespace-separated, case-agnostic string that represents the sentence and returns the parse as XML, which gets read into a more idiomatic in-memory structure for later processing.

For example, the sentence "That fluffy cat quickly ate the fish which swims because the cat was bored" corresponds to the T-expression set in Table 4.3. Of the T-expressions in the parse listed in Table 4.3, only the structural T-expressions and a subset of the syntactic and lexical T-expressions as discussed in § 3.1 are of interest. This reduced set of T-expressions is listed in Table 4.2. The next sections refer to these T-expressions in examples, shown in footnotes.

4.2.2 Pruning

Pruning proceeds in four stages. The first stage identifies pruning opportunities. These can be as simple as identifying all T-expressions with `has_det` as the relation or identifying those T-expressions that are the object of a relation that has been

Table 4.2 START parse of "That fluffy cat quickly ate the fish which swims because the cat was bored", reduced for clarity.

```
[[cat+6 eat+1 fish+5] because+2 [somebody bore+2 cat+6]]
[[cat+6 eat+1 fish+5] has_modifier+1 quickly]
[cat+6 has_property+3 fluffy]
[fish+5 has_rel_clause+3 [fish+5 swim+3 null]]
[cat+6 has_det that]
[[[cat+6 eat+1 fish+5] has_modifier+1 quickly] has_position mid_verbal]
[[cat+6 eat+1 fish+5] has_position leading]
[cat+6 has_det definite]
[because+2 is_clausal Yes]
[fish+5 has_det definite]
[because+2 has_category relation]
```

marked with `[* is_clausal Yes]`.[1] The second stage takes the opportunities that were identified in the previous stage and from them, computes those elements that should be merely deleted (leaving any other elements that are still referenced elsewhere intact), and those elements that should be purged entirely (deleting any other elements that reference them).[2] The third stage considers all *combinations* of the opportunities identified in the first stage and applies all the deletions for those opportunities. The fourth stage takes each of the reduced T-expression sets produced by the third stage and performs a cleanup step where any T-expressions that modify or are subordinate to elements that have been deleted are themselves deleted.[3] The resulting set of reduced T-expression sets is passed back to START for "generation".

[1]For example, the T-expression `[because+2 is_clausal Yes]` signals that there is an opportunity to prune the subordinate clause headed by `because+2`.

[2]Although the T-expression `[because+2 is_clausal Yes]` signals the opportunity for pruning, the pruner actually deletes the object of the T-expression `[[cat+6 eat+1 fish+5] because+2 [somebody bore+2 cat+6]]`, namely `[somebody bore+2 cat+6]`. The pruner purges it entirely, deleting it and all references to it.

[3]After the third and fourth stages, the maximally reduced set of T-expressions includes the following structural and "interesting" syntactic and semantic T-expressions:

```
[[cat+6 eat+1 fish+5] has_position leading]
[cat+6 has_det definite]
[fish+5 has_det definite]
```

Table 4.3 START parse of “That fluffy cat quickly ate the fish which swims because the cat was bored”

```
[[cat+6 eat+1 fish+5] because+2 [somebody bore+2 cat+6]]
[[cat+6 eat+1 fish+5] has_modifier+1 quickly]
[cat+6 has_property+3 fluffy]
[fish+5 has_rel_clause+3 [fish+5 swim+3 null]]
[cat+6 has_det that]
[[[cat+6 eat+1 fish+5] has_modifier+1 quickly] has_position mid_verbal]
[[cat+6 eat+1 fish+5] has_person 3]
[[cat+6 eat+1 fish+5] has_tense past]
[[cat+6 eat+1 fish+5] has_position leading]
[somebody has_number singular]
[somebody is_proper Yes]
[cat+6 has_det definite]
[[somebody bore+2 cat+6] has_person 3]
[[somebody bore+2 cat+6] has_tense past]
[[somebody bore+2 cat+6] has_voice passive]
[[somebody bore+2 cat+6] passive_aux be]
[[somebody bore+2 cat+6] has_clause_type tensed]
[because+2 is_clausal Yes]
[[[cat+6 eat+1 fish+5] because+2 [somebody bore+2 cat+6]] is_main Yes]
[cat+6 has_number singular]
[fish+5 has_number singular]
[fish+5 has_det definite]
[[fish+5 swim+3 null] has_person 3]
[[fish+5 swim+3 null] has_tense present]
[cat+6 has_category noun]
[eat+1 has_category verb]
[fish+5 has_category noun]
[has_modifier+1 has_category relation]
[quickly has_category adv]
[because+2 has_category relation]
[somebody has_category noun]
[bore+2 has_category verb]
[has_property+3 has_category relation]
[fluffy has_category adj]
[swim+3 has_category verb]
[null has_category nil]
[has_rel_clause+3 has_category relation]
```

4.2.3 Generation

In addition to its parsing capabilities, START has the ability to generate English sentences from their T-expression sets. This thesis takes advantage of this capability by transforming the reduced T-expression sets produced by pruning back into their "flat" English representation.[4]

4.2.4 Ranking

After parsing, pruning, and generation, the resulting sentences are ranked by their likelihood. Because these sentences are of different lengths, for each sentence, the likelihood is the probability that a sentence of the same length has the same content (ignoring case, diacritics, and punctuation). This step suffers from the same Markov approximation problems as discussed earlier, but because the reduced sentences are shorter, the Markov window can capture a larger proportion of the dependencies in the sentence.

4.3 Backoff

Using the simplification capabilities described in § 4.2, it is possible to augment the normal Markov approximation mechanism with a backoff mechanism. Normal Markov approximation suffers from problems of sparsity and the limited Markov window. Using a backoff system that incrementally removes modifiers and subordinate clauses helps to solve both of these problems. Backoff using simplification helps to solve the sparsity problem by making sentences more generic. More generic sentences are more likely to have been observed in the training corpus. Although making sentences more generic makes them more likely to appear in the training corpus, if the "backbone" of the original sentence is nonsense, the simplified sentence is likewise unlikely to appear.

[4]The maximally reduced T-expression set mentioned earlier, when generated back into English by START, produces "The cat ate the fish.".

Backoff using simplification also helps to solve the limited Markov window problem by shortening the distance between words that share a dependency. In particular, by removing intervening modifiers and subordinate clauses, the Markov model can analyze more long-distance word dependencies. Sometimes, relations can be exposed to analysis by the Markov model just by reordering the modifiers and subordinate clauses. In this case, the system can analyze the sentence without the information loss caused by removing those modifiers and subordinate clauses.

Using this backoff scheme, the system can discover that "The well-dressed tall soft-spoken man walked slowly down the curved street." is potentially correct, while "The talkative red shoe vigorously eats the mountains." is almost certainly not. "The talkative red shoe vigorously eats the mountains." does not appear in the training corpus at all, even after simplification to "Shoe eats mountains.". However, "The man walked." *does* appear in the database, so at least the action is reasonable, even if the modifiers are not.

5. Summary

The work of this thesis is composed of three parts: an n-gram language model, the interface to the START natural language system, and the simplification system built on top of those. To support the kinds of queries that are most useful to the user of an n-gram model, a query language was designed that can query based on part of speech, as well as being able to ignore capitalization, punctuation, and diacritical marks.

The START natural language system is a broadly applicable system which contains the natural language parser that is used in this thesis. START's natural language parser transforms English sentences into a set of T-expressions that expose the structural, syntactic, and lexical information of the sentence. The separability of these T-expressions and the ease with which they can be manipulated by programs facilitate the construction of systems that work by parsing sentences with START, modifying the resulting T-expression set, and then generating the T-expressions back into English with START.

This thesis describes a simplifier that works in four stages built on top of START and the n-gram system developed for this thesis. The simplifier parses English into T-expression sets using START. These T-expression sets are then reduced in a pruning step that exposes the "backbone" of the sentence. These reduced T-expression sets are generated back into English by START before being ranked for correctness by the n-gram system. The result is a list of simpler sentences, ordered by their "correctness" as analyzed by the n-gram system. These simpler sentences can be used to analyze the correctness of words in a more generic context than the original, or to widen the apparent window of the n-gram model. Systems that rely on an n-gram model could use the work in this thesis to enhance their models.

The simplifier is used for intelligent backoff when Markov approximation cannot accurately capture the dependencies between words, or when sparsity causes otherwise correct utterances to be declared incorrect. By reducing the distances between words sharing a dependency, the simplifier exposes that dependency to analysis by the n-gram model. By making sentences more generic, the simplifier reduces the problem of sparsity in the training dataset of the n-gram model.

5.1 Applications

The system of simplification and intelligent backoff has applications anywhere overly specific sentences need to be made more generic or n-grams are being applied to data where the Markov assumption does not hold or sparsity is a problem. Specifically, this system was developed to assist the START question answering system with overly specific questions. For example, if START is asked "How many bright shiny pennies are there in a dollar?" it cannot answer the question. However, after applying the simplification process, one of the sentences we obtain is "Are there pennies in a dollar?". START *can* answer this question and arrives at the correct answer of "Yes. There are 100 cents in one dollar.".

Other applications include automatically simplifying documents by removing complicated modifiers and subordinate clauses. This system also could be applied to work with the Genesis [11, 12, 13, 14] system as an automated method of extracting the backbone of a sentence for story understanding. Future work may augment this simplifier to replace words with a special token (as discussed in § 2.2.1) representing their part of speech or enhance the simplifier to transform passive voice sentences into active voice and vice versa.

Bibliography

[1] Thorsten Brants and Alex Franz. All our n-gram are belong to you. `http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html` Accessed: January 28, 2015.

[2] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 310–318, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.

[3] Michael John Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 184–191. Association for Computational Linguistics, 1996.

[4] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.

[5] Boris Katz. A three-step procedure for language generation. A.I. Memo 599, Massachusetts Institute of Technology Artificial Intelligence Laboratory, December 1980.

[6] Boris Katz. Annotating the world wide web using natural language. In *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO 1997)*, pages 136–159, June 1997.

[7] Boris Katz, Gary Borchardt, and Sue Felshin. Natural language annotations for question answering. In *Proceedings of the 19th International FLAIRS Conference (FLAIRS 2006)*, Melbourne Beach, FL, May 2006.

[8] Boris Katz and Patrick H. Winston. Parsing and generating English using commutative transformations. A.I. Memo 677, Massachusetts Institute of Technology Artificial Intelligence Laboratory, May 1982.

[9] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, 1987.

[10] Mitchell Marcus, Ann Taylor, Robert MacIntyre, Ann Bies, Constance Cooper, Mark Ferguson, and Alyson Littman. The Penn Treebank Project. `http://www.cis.upenn.edu/~treebank/` Accessed: January 28, 2015.

[11] Patrick H. Winston. The genesis story understanding and story telling system a 21st century step toward artificial intelligence. Technical report, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, June 2014. `http://groups.csail.mit.edu/genesis/papers/StoryWhitePaper.pdf` Accessed: January 28, 2015.

[12] Patrick Henry Winston. The strong story hypothesis and the directed perception hypothesis. In Pat Langley, editor, *Technical Report FS-11-01, Papers from the AAAI Fall Symposium*, pages 345–352, Menlo Park, CA, 2011. AAAI Press.

[13] Patrick Henry Winston. The next 50 years: a personal view. *Biologically Inspired Cognitive Architectures*, 1:92–99, 2012.

[14] Patrick Henry Winston. The right way. *Advances in Cognitive Systems*, 1:23–36, 2012.