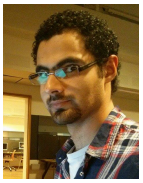
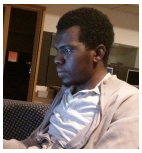
Supernova: NLG for wardrobe tips depending on weather

Omar Elshenawy
09/10/1990
omares@kth.se



Mohamed Abdulziz
31/07/1985
moaah@kth.se



Thomai Stathopoulou
10/10/1987
thomai@kth.se

Varsha Kirani Gopinath
04/06/1991
varsha@kth.se

Abstract

We create an application that provides the user with tips on what to wear depending on weather conditions. A sample output of the application would be "Today is rainy, don't forget your umbrella", or if the weather is windy, "Today is rainy and windy, wear a rain jacket". We attempt to build this application using a natural language generation system that would generate appropriate text according to the changing weather condition. We build the system using 3 NLG methods that come from 2 different trends in the NLG field, and we assess their performance for our application.

1 Introduction

Natural language generation is an important subtask of many applications that rely on human computer interaction. While as the Chinese proverb says, a picture is worth a thousand words, there are a lot of situations where pictures, schematic diagrams, maps, charts and plots cannot communicate information as effectively and efficiently as text would. There are no rules for when graphics is better than text, however, by means of observation, an annotated picture would better describe a physical location, charts can show progress, schematic diagrams can describe flow and hierarchy, but would all fail at describing abstract concepts such as causality, advice (that is why for instance the Internet phenomenon "AdviceAnimals" [9] memes are pictures with text overlaid), or even, what this paragraph attempts to describe, which are all straightforward to describe in text. Another issue on which the type of representation might depend is the expertise of the user, although graphical representation is considered an easy-to-understand way of representing information, research in psychology [11] has shown that in many cases, a considerable amount of information might be required to understand graphical representation and novice users might be better off with text.

Our application advices the user on what to wear with regard to the weather. As noted earlier, text is probably the best way to communicate advices. A possible approach to this would be to simply provide a structured text such as "Weather: Rainy, What to wear: Jacket", even though it might be informative, however it is not friendly, and could possibly cause user depression and frustration if they live in a place with generally bad, rapidly changing weather, such as Sweden. However, that is usually where users would need such an application in the first place. Ideally, the best way to convey the message would be naturally generated text with a funny joke, maybe overlaid on a meme, something similar to how a friend would describe the weather in order to uplift the user's spirit. However, due to limited time, resources and scope of this project, we will attempt to generate plain natural language tips.

We build our system using 3 different methods that come from 2 different trends in the NLG field, namely, Corpus-Based Statistical Knowledge (CBSK)[6] which is the first method to use Abstract Meaning Representation (AMR) and Robust-PCFG [2] and Treebank-trained statistical generator (TTSG) [1] which are Context Free Grammar(CFG) based methods.

1.1 Contribution

We will build our system using 3 NLG methods, each relies on a different text generation philosophy, and have been developed for a different application. We will evaluate these methods for our application and compare their performance.

1.2 Outline

In Section 2 we overview different trends in field of natural language generation such as Abstract Meaning Representation and TreeBank & Context Free Grammar based architectures. We also briefly overview different common evaluation methods for NLG systems. In Section 3 we describe our implementation details for each method, discussing different practical aspects such as design of corpus. In Section 4 we describe in detail our evaluation experiment for the 3 methods and discuss the effectiveness of the methodology of evaluation. Finally, in Section 5 we discuss the strengths and weaknesses of the different methods in light of our application.

2 Related work

Until now, there has been a lot of research in the field of text generation, and as is expected there exists a plethora of implemented methods, using different architectures and bases. As mentioned in Section 1 within the frame work of this project, three different methods are used:

CBSK In the work of [6], the system over generates sentences. These sentences are grouped together in an optimal way to form a state transition diagram which has English words as its edge. This state transition diagram of sentences is called as *Word Lattice* as shown in Figure 6. Word Lattice is fed to a corpus based statistical ranker to get the best path of sentence as output, more likely to find fluent sentences over others.

Robust-PCFG Using a PCFG the two different representations of an LFG are created, which then are turned into a chart-like data structure producing the most probable text-tree. Then, if possible, lexical smoothing is applied, in order to ensure that the resulting text is readable and makes sense.

TTSG "Treebank-trained statistical generator" [1] is a statistical generation method that trains a statistical model by using an annotated corpus. When an input is submitted to an NLG system, the input can

go through several generation stages before an output is produced. At each stage the NLG system needs to make a decision to select the next generation. The method we will implement here belong to a group of methods that separate between the space comprised of all possible generations on one side and the mechanisms used to make the selections at each of the generation stages on the other side.

Evaluating NLG systems is not a straightforward task. The quality of text can vary with regard to both the reader and the writer. Writers might not always produce text that is optimal from the reader's point of view, and text that is optimal from the reader's point of view, might not be optimally written. Also, since the target user of any NLG system is usually human, a variance in assessment will exist due to the variation in human understanding. Evaluation methods in the current literature can be grouped into three categories[10], each tries to assess the NLG system from different perspectives. In Section 4, we give details into our experiment and explain the methodology we used for evaluation.

2.1 Evaluation Methods

Current evaluation methods can be grouped into three categories[10]:

- Task-Based Evaluation
- Human Rating and Judgment
- Automatic Metrics

2.1.1 Task-Based Evaluation

Task-based evaluation involve measuring the impact of the text on end-users. It usually involves assigning the users with a task related to the purpose of the NLG, and then measuring the user's performance during the task, such as, how many mistakes did the user make, how much did the user learn, or how did the user rank or sort a list of items related to text generated by the NLG.

2.1.2 Human Rating and Judgment

In this group of methods, humans are asked to rate and judge the text generated by the NLG. This evaluation can be done in several ways. One way is to provide users with text that is written by humans along with that of the NLG system, and asking them to rate both without knowing which is

written by which. Another is asking them to rate the quality of the text on a 5 point scale, or asking them to choose their favorite text.

2.1.3 Automatic Metrics

Evaluation methods involving humans are usually quite expensive, time consuming, hard to design, hard to repeat. For these reasons, there has been a growing interest in evaluating NLG systems by comparing them to a reference-text corpus created by human experts using automated metrics such as string-edit distance, tree similarity and BELU [8]. This is cheap, allows for repetitive immediate evaluation of the system. However, there exists some concerns for using these methods, among them is the fact that the reference-text corpora created by experts is optimal from a writer's point of view, which might not be optimal from a reader's point of view, therefore the test is for similarity to writer's text.

3 My method

3.1 Probabilistic Context-Free Grammars (PCFGs)

3.1.1 An introduction to Context-Free Grammars (CFGs)

A Context-Free Grammar (CFG) is a 4-tuple $G = (N, \Sigma, R, S)$, where:

- N is a set of non-terminal symbols
- Σ is a set of terminal symbols
- R is a set of rules of the form $X \rightarrow Y_1 Y_2 ... Y_n$, where $X \in N, n \geq 0$ and $Y_i \in (N \cup \Sigma) \forall n = 1, 2, ...n$
- $S \in N$ is a distinguished start symbol [1]

A CFG creates sequences of symbols, replacing non-terminal symbols with a combination of non-terminal and terminal symbols, using the rules in R. The procedure of replacing symbols stops, when there are no non-terminal symbols left to be replaced. Every time a non-terminal symbol is replaced, the surrounding symbols are not taken into consideration, hence the term "Context-Free".

When creating a CFG for a language (e.g. English), usually non-terminal symbols represent basic syntactic and grammatical categories, whereas terminal symbols represent actual words.

[1] The definition of a CFG is borrowed from [3]

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, dog, telescope, the, with, in}

R =

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
NN	→	dog
DT	→	the
IN	→	with
IN	→	in

Figure 1: A Context-Free Grammar [3]

In Figure 1, an example of a CFG based on the English language is shown. It is obvious in this example, that the non-terminal symbols can either be generic syntactic terms (i.e. NP = Noun Phrase, VP = Verb Phrase etc.), or specific grammatical terms (i.e. NN = Noun Singular, Vi = Intransitive Verb, Vt = Transitive Verb etc.).

There is no restriction when it comes to the rules, as long as the symbol on the left-hand side of the rule is a *non-terminal*. The symbols on the right-hand side, can be any combination of symbols (terminal and/or non-terminal), or even an empty string (ϵ). A very specific kind of rule, is the unary rule. These rules have only *one* symbol on the right-hand side and this symbol is always a terminal symbol (the table on the right side of Figure 1 shows all the unary rules of the example CFG). The set of the unary rules can also be described as a lexicon, i.e. a tag for every word in the language (or a subset of the used language), representing what part of speech each word is (part-of-speech tagging, POS tagging).

3.1.2 Parse Trees

As mentioned in Section 3.1.1, given a CFG one can produce a sequence of symbols (i.e. a sentence), simply by replacing non-terminal symbols with other symbols, according to the rules in R. One very common method is, starting with the start symbol S, to always replace the left most non-terminal symbol of the sequence, until there are only terminal symbols left. The set of rules that were used, in order to create the sentence, can be represented

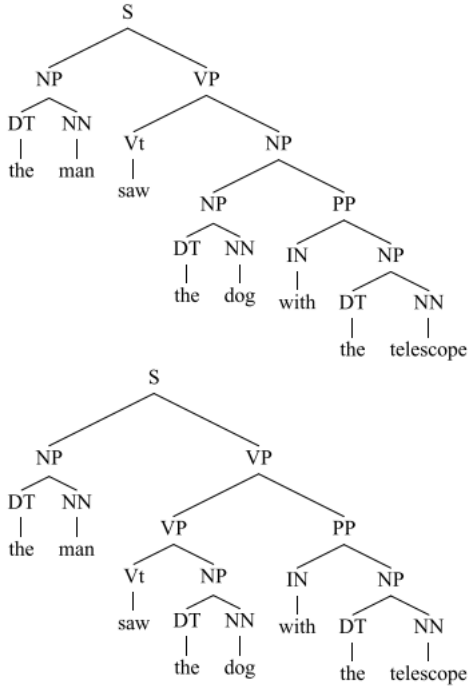


Figure 2: Two different parse trees for the same sentence (symbol sequence) [3]

as a tree. This tree is called a **Parse Tree**.

Because of the fact that there can be different rules with the same non-terminal symbol on the left-hand side (the same non-terminal symbol can be replaced by different combinations of non-terminal and terminal symbols), there can be more than one parse trees for one sentence. Additionally, given a sentence there a different methods (parsers) to generate parse trees. More details about different parsers are discussed in Subsection [reference subsection].

3.1.3 Probabilistic Context-Free Grammars (PCFGs)

Given a CFG, as defined in Section 3.1.1, a Probabilistic Context-Free Grammar (PCFG), can be defined as an extension of the CFG. Therefore, it consists of the 4-tuple $G = (N, \Sigma, R, S)$, as well as the new variable:

$$q(\alpha \to \beta) \forall rule \in R$$

The new variable represents the conditional probability of choosing rule $\alpha \to \beta$, given that the left-hand side non-terminal symbol is α. So, for any

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, dog, telescope, the, with, in}

R, q =

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4

Figure 3: A Probabilistic Context-Free Grammar [3]

non-terminal symbol X, there is the constraint:

$$\sum_{X \to \beta \in R} q(X \to \beta) = 1 \text{ [3]}$$

An example PCFG is presented in Figure 3. This example is based on the CFG of Figure 1. Given a PCFG, one is able to calculate the probability of a parse tree, by multiplying the probabilities of all the rules that have been used for the production of the sentence.

In order to produce a PCFG based on a CFG, one would need to have a corpus, i.e. a set of sentences (symbol sequences) that can be produced by the given CFG. These sentences are converted into parse trees (according to the chosen parsing method) and the occurrences of every rule are counted. In the end the probability of every rule is the number of the rule's occurrences divided by the number of the occurrences of rules with the same non-terminal symbol on the left-hand side:

$$q(\alpha \to \beta) = \frac{Count(\alpha \to \beta)}{Count(\alpha)}$$

After having created a PCFG, the next step would to actually try and generate sentences based on that particular grammar. For that, exist different methods, that are to be be discussed in Section [reference].

We are basing our system upon 3 methods as previously mentioned. In this section we first describe each method in detail, then we cover implementation aspects that we changed to fit our application needs.

3.2 Robust-PCFG

This method [2] is based on a Probabilistic Context-Free Grammar (PCFG) [3] for the text generation, which in turn is based on an Lexical Functional Grammar (LFG)[12], that is automatically acquired.

The LFG consists of two different kinds of representations, the Constituent (c-)Structure and the Functional (f-)Structure. c-Structure is a structured tree, one that describes the structure of the words in a sentence, in terms of word (or phrase) order. It basically resembles the parsing tree created by a PCFG, with certain additions to it. f-Structure is the c-structure transformed into an attribute-value matrix, providing specific syntactic terms, such as "subject", "object" etc.

The algorithm that is used, needs an f-structure and produces the most probable c-structure for the given input. There are two different approaches in creating the f-structures[2]: the **pipeline** architecture and the **integrated** architecture.

In the first method we are given a treebank and based on that we create a PCFG. We, then, use this grammar to convert unseen text into PCFG-format trees. These trees are then annotated, in order to create the desired f-structure.

In the integrated architecture, we are again given a treebank. In this case the treebanks are first annotated, and then an annotated PCFG is extracted from it. Using this grammar, the unseen text is parsed into a tree with only f-structure annotations, which are then converted to an f-structure by a constraint solver.

For the text to be generated the grammar is first converted into Chomsky Normal Form (CNF) and after that, a chart-like data structure is created[5]. When two different chart items have equivalent rule left hand sides and lexical coverage, the most probable one is used, thus ensuring that the tree produced, is the most probable one with respect to the given f-structure.

After the creation of the tree, lexical smoothing takes place, when possible. At this point the part-of-speech (POS) tag is used, i.e. each word is marked up as a specific part of speech, taking into account the texts context, along with the definition of the word. That is, if certain words have different uses, in terms of meaning (being used literally or metaphorically), or in terms of grammar (being used as a noun or verb etc).

This method has been tested for the generation of sentences of varying lengths. It is evaluated using the Simple String Accuracy and BLEU evaluation metrics. In addition, the coverage is measured as the percentage of the given f-structures that generate a string.

3.3 TTSG

This method [1] is using a context free grammar, and a statistical model trained on an annotated corpus. The model gives different priorities to the grammar rewrite rules. Follows is a brief overview of the method.

1. A raw parallel corpus will be built. Each row of the corpus will be comprised of two data fields: one field contains the weather prediction, and the other field contains the associated tip written in NL. Below table shows and example two rows from the corpus.

2 C, 5k/h,0mm	Its cold, wear your heavy jacket
15 C, 10k/h,33mm	Take your raincoat

2. A rewrite grammar will be created; in particular a context free grammar. The grammar will contain few rules sufficient to represent the tip part of the corpus. Below shows the form of the planned grammar rules.

   Sentence $->$ VerbPhrase NounPhrase

   VerbPhrase $->$ Verb

   NounPhrase $->$ Pronoun Noun

3. The raw corpus will be annotated to generate a Treebank. This will be carried out following below steps:

   (a) For each tip sentence from the raw parallel corpus, the possible derivations will be created using the grammar defined in the previous step. A derivation shows the sequence of rewrite rules followed to reach to the sentence. A sentence may have more than one derivation. Below shows one way the sentence "Take your raincoat" can be derived using the example grammar in the previous step.

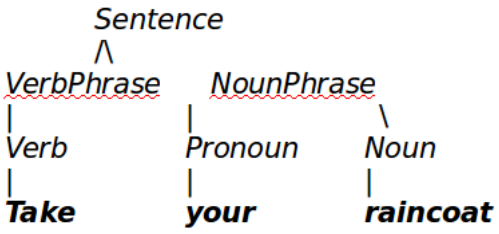


Figure 4: Sentence Derivation

(b) Using the derivations from (a), each substring of the sentence will be annotated with the corresponding rewrite rule that generated it. When (a) and (b) are carried for all the corpus, the resulting annotated corpus is the treebank.

4. The count of each rewrite rule in the treebank will be made. Using that, the probability distribution for all the rewrite rules will be generated.

5. The generation logic will be designed to recursively apply the rewrite rules using the defined grammar until all the non-terminals are completely replaced with terminals. At each generation step, when more than one rewrite rule are possible to be applied, the rewrite rule with the highest probability will be selected. This is the place where the built statistical model is used.

3.4 CBSK

This method is the first method to generate text using AMR. It uses a word lattice as seen in Figure 6, to represent the overgenerated text and feeds this to statistical extractor for ranking as seen in Figure 5.

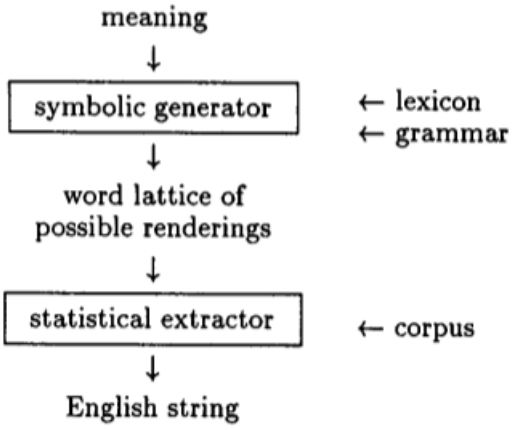


Figure 5: Combining symbolic generator and statistical extractor [6]

3.4.1 Input representation (using AMR language)

AMR: Abstract Meaning Representation is labeled directed graph or feature structure, we represent input in the form of meanings.

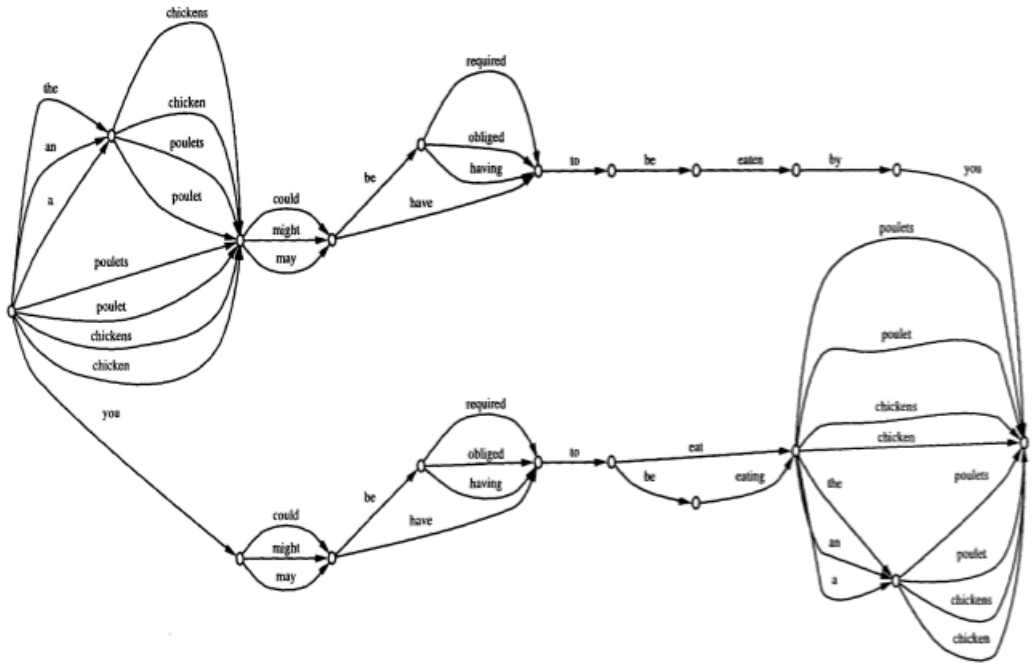


Figure 6: Word Lattice [7]

- Basic form of AMR is (label/concept).
  For eg:

  (p1 / |cold > weather|) // "Cold Weather"

- Concepts can be modified by keywords. For eg:

  (p2 / |jacket,coat| :quant plural)

  This will cover word : jacket, jackets,coat,coats.

- Concepts can be associated with each other in a nested fashion to get more complex meaning. This can be done using keywords.
  For eg:
  (p3 / |is,be| :subject (p4 / |weather|) :object (p5 / |cold|))
  Notations:

  – slash "/" is shorthand for type feature or instance
  – "<" , ">" hierarchy indication

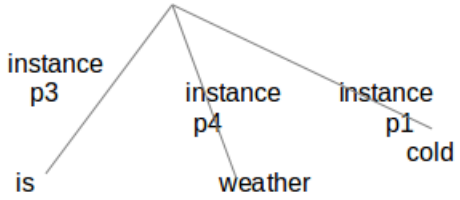


Figure 7: Representation of "Weather is cold"

- "subject" , "object" are roles (out of many other roles)
- "," is to represent OR

3.4.2 Lexical and Morphological Knowledge

To this meaning (AMR) we apply lexical knowledge (lexicon) and morphological knowledge. The English lexicon used in AMR is basically SENSUS concept ontology. *Rank* field is available in each lexicon, which orders the concepts by sense frequency for the given word. Lexicon here is kept simple and overrules many features, like transitivity, which are required for generators which produce correct grammatical sentences. But here, the post processor will handle this by ranking different grammatical realization according to their likelihood. Word choice issue is handled by taking advantage of the word sense ranking offered by lexicon. If several concepts can be expressed by same word then it will return only that word.

Lexicon contains word in the root form. Morphological inflections must be applied. Inflections are applied without considering the resultant output meaning, as this issue is handled by the statistical extractor.

3.4.3 Grammar Formalism

Syntactic and semantic roles are mapped to grammatical word lattice by using keyword rules. Instance rules will build initial word lattices for each lexical item (keyword- :polarity, :quant) and for basic noun and verb groups. Other rules are defined to relate concepts based on role arrangements to build lattice. All the remaining keywords which are not relevant for the above two rules are kept under :rest role in order to avoid combinatorial explosion.

3.4.4 Generation Algorithm

The algorithm used here is compositional. AMR representation is transformed to word lattice, using keywords (which avoids syntactic representa-

tion). This provisions to have simple inputs.

- Top level keywords is matched with the rule.
- Matched rule splits AMR and associate sub-AMR with each keyword of the rule and place all other relations in :rest role.
- Each sub-AMR is recursively matched against the keyword rules until it bottoms out in order to apply instance rule.
- Lexical and morphological knowledge is applied to build word lattice. Instance rules concatenates lattices to make phrases and filter them to desired category (sentence as default)
- Statistical extractor ranks the lattices for desired output. Bi-gram statistical extractor is used here.

3.5 Implementation

We will train the CFG methods [1, 2] using a hand made corpus. For the AMR method, we will use the AMR Bank [4] to create AMR.

4 Experimental results

4.1 Experiemntal setup

We have decided to use Human Rating and Judgment evaluation philosophy. For each scenario (e.g. {rainy, cloudy, windy} or {sunny, clear, calm}), we will provide the users with 4 texts, one from each system and a human written text. We will then ask the users to rank the texts according to their preference.

4.2 Experiment ...

bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla

5 Summary and Conclusions

bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla

References

[1] Anja Belz. Statistical generation: Three methods compared and evaluated. 2005.

[2] Aoife Cahill and Josef van Genabith. Robust PCFG-Based Generation using Automatically Acquired LFG Approximations. *. . . International Conference . . .*, (July):1033–1040, 2006.

[3] Michael Collins. Probabilistic Context-Free Grammars (PCFGs). *Lecture Notes*, pages 1–18, 2013.

[4] South California ISI. AMR Bank http://amr.isi.edu/index.html.

[5] M Kay. Chart generation. *Proceedings of the 34th annual meeting on Association . . .*, pages 200–204, 1996.

[6] I Langkilde and K Knight. Generation that exploits corpus-based statistical knowledge. *Proceedings of the 36th Annual Meeting of the . . .*, pages 704–710, 1998.

[7] Irene Langkilde. Forest-based statistical sentence generation. *Proceedings of the 1st North American chapter of the . . .*, 2000.

[8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, page 311, Morristown, NJ, USA, July 2001. Association for Computational Linguistics.

[9] Reddit. AdviceAnimals, http://www.reddit.com/r/AdviceAnimals/.

[10] Ehud Reiter and A Belz. An investigation into the validity of some metrics for automatically evaluating natural language generation systems. *Computational Linguistics*, (December 2008), 2009.

[11] Ehud Reiter and Robert Dale. Natural Language Generation in Practice. pages 23–40.

[12] Jürgen Kaplan Ronald M Wedekind. LFG Generation by Grammar Specialization. *Computational Linguistics*, (March 2011), 2012.