LANGUAGE GENERATION 3
PROBABILISTIC SENTENCE REALIZATION

Theme

Statistical approach to language generation: realization only, no work at all on anything beyond the sentence.

Summary of contents

A statistical approach to language realization has been slow in coming, but since its introduction in the mid-1990s it has become the dominant paradigm. The approach parallels that n speech recognition, namely 'overgenerate and select': for the given input, use a simple grammar to generate all (or many) possible realizations, pack them in a lattice for efficiency, and then apply some ranking criterion to select the best one(s). Systems Nitrogen, Halogen, and others; ICT's Virtual Human realizer.

INTRODUCTION

As we have seen in previous lectures, a big part of realization is the problem of choice: for a given input which variant should you say? In other words, which expansion rule should you choose at each point, and how do you know with decisions early in the process ('high up' in the eventual syntax tree you are about to realize) which option(s) will allow you to include all the input you have been given lower down (and which options will run into a dead end)? The criteria of choice —of controlling the realizer's decisions— are very hard to formulate.

So, on the face of it, the statistical approach is ideally suited for realization: if you can realize all possible variations for the input and then apply a statistically trained ranking/selection function, you avoid the problem of explicit realizer control. To obtain the ranking probabilities, you have tons of data: normal text. The main problem is with the input: what representations to use? And when you have that, who will build a large enough training corpus of input-output pairs?

For many years, these questions stymied the NLG community interested in trying statistical methods.

1. **NITROGEN** — Knight and Hatzivassiloglou, ISI (1994–1996)

In 1994, Kevin Knight (ISI), working with Vasileios Hatzivassiloglou (then a graduate student at Columbia University, spending two summers at ISI), designed a method to bypass the above problems. Using a method analogous to the speech recognition community's approach, they split the task of realization into two parts:

1. creating many hypotheses for the sentence using a very simple grammar
2. selecting the best one using a lattice search procedure

To achieve the first step, they built, by hand, a small expansion grammar, similar to what you did in your homework. Each rule expanded just a small section of the input: various rules for NP creation, for VP creation, etc. This grammar was very general, and massively overproduced sentence fragments.

As input they used AMR, a form of the SPL notation that was created at ISI in the late 1980s for Penman:

```
  (h / |possible<latent|
     :domain (h2 / |obligatory<necessary|
                  :domain (e / |eat,take in|
                             :agent you
                             :patient (c / |poulet|))))
```

All symbols defined within bars (|) are defined inside the SENSUS ontology, a taxonomy of some 90,000 concepts derived from Princeton's WordNet. Each is linked to one or more English words.

Any idea what this representation means? ☺

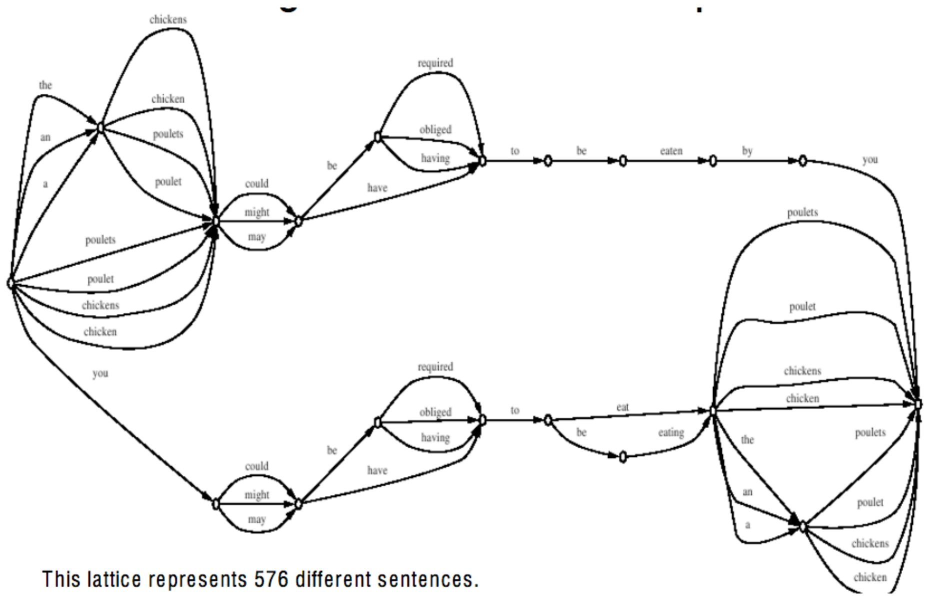
Read this as follows: make a sentence about the occurrence `h`, which is an instance of a `|possible<latent|` concept. What is it that's possible? The domain of `h` is an instance of a `|obligatory<necessary|` concept, which in turn has a domain `e`, which is an instance of an `|eat,take in|` concept. This eating was done by someone called `you` and was done to a thing `c`, an instance of a `|poulet|` (chicken).

To produce candidate sentences, Knight and Hatzivassiloglou first matched their small grammar against the AMR representation, found all related English words, and, piece by piece, strung together the surface-level phrases to make 'sentences'. Given this crude assembly method, a simple input might produce several million output sentences: there is no checking for subject-verb agreement, consistent tense, or anything! *For this example, they obtained 10,734,304 different sentences*; some randomly selected examples are:

```
   You may be obliged to eat that there was the poulet.
   An consumption of poulet by you may be the requirements.
   It might be the requirement that the chicken are eaten by you.
   That the consumption of chicken by you is obligatory is possible.
```

Most of the sentences are awful, not really English at all. But somewhere among the 10 million, there are some good ones! How to find them? That's the problem for the second step…

To speed up the ranking, they packed the candidate sentences into a *lattice*, sharing word strings as much as possible. This meant that any sequence of words shared in multiple sentences sentence needed only to be ranked once. Some of the above sentences (576 of them) can be packed as follows:

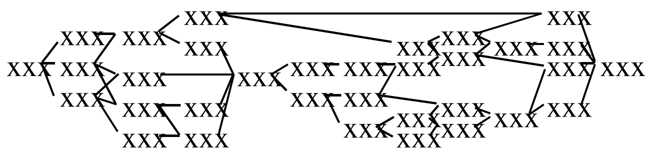


This lattice represents 576 different sentences.

Now comes the second step: ranking the alternatives. To achieve this step, they used a corpus to learn which expressions are preferred and which not. For example, from the corpus, Langkilde (another of Knight's students) obtained the following counts:

I am: 2797 instances
I are: 47 instances
I is: 14 instances

It's clear which option to choose, when they are nicely packed into the lattice as options. They built a bigram model of English from 46 million words of *Wall Street Journal* articles (1987–88) (and later, a trigram model for better results). (A *bigram model* is a list of all pairs of English words that occur in the corpus, with frequency counts, just like the above *I am* / *I is* / *I are* examples. A *trigram model* is all triples



LATTICE DATA:
260 nodes; 703 arcs;
10,734,304 paths

of words.)

To rank the sentences, they performed a Viterbi traversal of the lattice. This is a common procedure, a kind of beam search, used in speech recognition too:

- Starting at the first node (node 0), assign a bigram score to each alternative word w_i node at depth 1 according to its bigram score, namely (*START word$_i$ score*).
- Repeat:
    - For each next node w_{i+1}, find its bigram (w_i w_{i+1} *score*), and update its score by multiplying the score at node *i* with the new score.
    - If the resulting score drops below a cutoff threshold, discard that path; else keep it for future consideration.
    - Repeat for the next node, going from left to right, until done.
- When done, extract all sentences remaining in the search paths, and order them by score.

(In practice, to keep the scores from becoming too small —each one is a probability between zero and 1— they used log scores, which can just be added instead of multiplied:

$$\log P(Sentence) = \Sigma \log P(w_{i+1} \mid w_i) \quad \text{(for bigrams)}$$

Because this equation assigns lower and lower probabilities as sentence length increases, they added an additional heuristic amount of 0.5*L* (where *L* is the number of words in the sentence) to the final score.

After scoring, each sentence has a score that reflects how common in English its bigram sequence is. The higher the score, the more typical the bigrams, and the better the rank of the sentence.

For the above example, the best outputs, as ranked by bigram model, are:

```
1. You may have to eat chicken.
2. You might have to eat chicken.
3. You may be required to eat chicken.
4. You might be required to eat chicken.
5. You may be obliged to eat chicken.
```

Naturally, this is a slow process. But NITROGEN was a breakthrough system, and illustrated a whole new way of performing sentence realization.

A demo of NITROGEN is at http://www.isi.edu/natural-language/projects/nitrogen/demopage.html. Here are some more sample inputs:

```
(H / |detest|
   :agent they
   :patient (H2 / |hypocrisy<falseness|))

 (A / |obligatory<necessary|
   :domain (A2 / |adapt<vary|
              :patient (M / |method<knowhow|)
              :destination (C / |circumstance<status|)))
```

2. **HALogen —** Langkilde and Knight, ISI (1996–2001)

Knight's student Irene Langkilde took this work further in her Ph.D. research, producing HALogen. She addressed several computational problems. First, since the lattices were often too large to be traversed efficiently, HALogen packed whole sentences and sentence parts (many NPs, for example, used the same high-level structure) into a forest, leaving only the first and last words of each subtree to be considered if they occurred within other sentences. When ranking the encapsulating sentences, HALogen could then 'skip' over the included trees altogether, since all it needed are the end-point words.

Also, since the lattice and the bigrams cannot capture long-distance dependencies (for example, when more than 2 words separate the subject's head noun from the verb, there is no way to determine if the verb should be singular or plural), HALogen included some syntactic information.

3. FERGUS — Bangalore, Rambow, and Walker, AT&T (1998–2000)

One potential shortcoming of the NITROGEN model is the lack of any kind of syntactic representation. One might argue that this is overcome by the bigram (or trigram) model. However, a team at AT&T built their own version of this approach, FERGUS, which included a separate step using trees from a Tree Adjoining Grammar (TAG). Using these trees they can exercise more control on the candidate sentences that are packed into the lattice, and can propose more elaborate variations of them—even, for example, splitting out an adjective into a relative clause ("the blue book" → "the book that is blue").

This enables them to start including some steps from microplanning into the statistical alternatives, as we discuss later. (After all, there's no reason the lattice should contain only syntactic variations!)

4. A simple statistical realizer — Bhagat and Hovy, ISI (2004–2005)

Is it possible to build a statistical generator using extremely simple methods? And with little training data? Indeed it is. The input is a frame-based representation (see below for two variant forms):

Sentence to generate: "Clear the area, sergeant!"

Corresponding input frames

```
     Nested form                 Collapsed form
 s [
   :addressee sgt            s.addressee sgt
   :mood imperative          s.mood imperative
   :sem [
     :type action            s.sem.type action
     :event clear            s.sem.event clear
     :patient area           s.sem.patient area
```

```
            :time present           s.sem.time present
        ]]
```

Bhagat and Hovy built a system that used one central 'correspondence table' to record the correlation between a slot-value pair of the input and one or more words (in their case, trigram sequences) of the output. The training data was a set of 500 or so sentences, each sentence with its associated frame of slot-value pairs, built by a human.

Learning phase: Let C(w_{j-2} w_{j-1} w_j | m_i) be the number of times a word sequence (trigram) w_{j-2} w_{j-1} w_j occurs in sentences in the training data whenever the slot-value pair (meaning element) m_i appears in the associated frame. The corresponding probability is calculated as follows, counting the number of times this trigram is seen together with this slot-value pair, divided by the total number of times this trigram is seen in total (i.e., with and without m_i):

$$P(w_{j-2}\, w_{j-1}\, w_j \mid m_i) = \frac{C(w_{j-2}\, w_{j-1}\, w_j \mid m_i)}{\sum_{k=3} C(w_{k-2}\, w_{k-1}\, w_k \mid m_i)}$$

Realization phase: How are sentences realized, once the table is built? Taking as input a semantic frame, the generator considers each slot-value pair in the input in sequence and collects the (most likely) trigram(s), producing a long list of word triples. It then considers various overlappings of them to try to produce a grammatical sentence. In more detail, this is a 3-step process: voting, overlapping, and filtering.

a. Voting: All entities in the input frame that belong to one of the three classes are first replaced by their corresponding class labels. (That is, each person's name is replaced by the symbol PERSON, each number by NUMBER, each time expression by TIME. The reason for this is to 'concentrate' the training data—since there are a few sentences, it just hurts by treating every different name differently, etc.)

From the table P(W / M) is collected a set of candidate trigrams "w_{j-2} w_{j-1} w_j" for each slot-value pair (meaning element) "m_i" of the input frame, each with an associated conditional probability P(w_{j-2} w_{j-1} w_j | m_i). Of course, the same trigram may appear several times. To complete this step, the probabilities for each trigram are summed to provide its weight:

$$Wt(w_{j-2}\, w_{j-1}\, w_j) = \sum_{i=1}^{n} P(w_{j-2}\, w_{j-1}\, w_j \mid m_i)$$

b. Overlapping: In this step, the candidate trigrams obtained after voting are sorted in descending order of their weights and the top 30 trigrams are selected. Then a word-overlapper is applied to these final candidates in order to tie together trigrams with shared words, to form a set of sentences.

The word-overlapper considers all possible combinations of trigrams. Two trigrams (w_a w_b w_c) and (w_x w_y w_z) are considered overlapping trigrams if and only if $w_b = w_x$ and $w_c = w_y$. These trigrams together form a (partial) sentence w_a w_b w_c w_z (or w_a w_x w_y w_z). All possible combinations are computed and stored in an acyclic graph (where the vertices are the trigrams and the edges connect overlapping trigrams), and all possible sentences are obtained by finding all possible paths (starting from each node) in the graph.

Each sentence obtained has a weight. This sentence weight is the sum of logs of weights of all the trigrams that form the sentence divided by the number of trigrams in the sentence. To give preference to longer sentences, we also add a small scaling factor that increases proportional to the number of trigrams in the sentence. The final sentence weight is calculated as follows:

$$\text{weightOfSentence} = \frac{\sum \log(Wt(w_{j-2}\, w_{j-1}\, w_j))}{n} + \frac{n}{10}$$

where n = # of trigrams contained the in sentence. The sentence with the highest weight is chosen.

c. Filtering: The final step of generation cleans up the sentence. All class labels in the sentences (NAME, etc.) are replaced by their corresponding instances from the semantic frame; ones that cannot be replaced by any of the instances from the semantic frame are removed.

Evaluation phase: How well did this procedure work? Many times the sentences contained additional material not licensed by the input frame, but 'dragged in' by the sentences in the training data. Often, little pieces of the input frame were not expressed in the output. Bhagat and Hovy asked two human evaluators to score the performance of the generator. The evaluators compared the outputs generated by the system with the corresponding gold standard sentences, and categorized the output sentences into one of the following:

(a) **Exact**: Word by word exact match with the gold standard
(b) **Close+high-grammar**: Same meaning as gold standard, but slightly different words. Still grammatical
(c) **Close+low-grammar**: Same meaning as gold standard, but slightly different words. May have some small grammatical errors, but in general, the meaning is clear
(d) **Partial+high-grammar**: Partial meaning overlap with gold standard, and grammatical
(e) **Partial+low-grammar**: Partial meaning overlap with gold standard, but some grammatical error
(f) **Bad**: No meaning overlap with gold standard

When the system in trained on the entire training data, an average of 74% of the outputs were acceptable. (The kappa score for the inter-judge agreement is a respectable 0.69.)

Problems: No checking of grammaticality for longer distances — if an agreement stretches over three words, no trigram will notice!

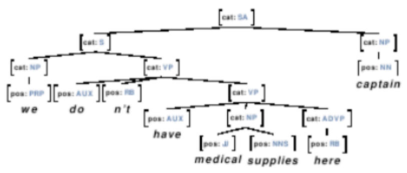
5. **Virtual Human realizer** — DeVault, USC/ICT (2007–)

In current work at USC's ICT center, David DeVault is building a large statistical sentence realizer. The domain is one in which computational agents interact with humans in a kind of virtual world rather like the Star Trek Holodeck. The humans are military trainees, being confronted with a simulated situation in which they have to solve a problem under pressure. For a sentence that might be realized as "There are no medical supplies here, captain", the Virtual Human system uses a frame-like input representation:

speech-act.action	*assert*
speech-act.content.polarity	*negative*
speech-act.content.object-id	*market*
speech-act.content.attribute	*resourceAttribute*
speech-act.content.value	*medical-supplies*
addressee	*captain-kirk*
dialogue-act.addressee	*captain-kirk*
speech-act.addressee	*captain-kirk*

This system resembles the simple one of Bhagat and Hovy above, but in order to address the grammaticality problems mentioned above, it includes syntax explicitly, and is much larger. It contains two large 'correspondence tables': one that maps word (sequences) to fragments of grammar, and one that maps frame slot-value pairs to word sequences:

$u =$
we don't have medical supplies here captain
$\mathrm{syntax}(u) =$



$\mathrm{semantics}(u) =$

we do n't	speech-act.action = assert
	speech-act.content.polarity = negative

have	speech-act.content.attribute = resourceAttribute
medical supplies	speech-act.content.value = medical-supplies
here	speech-act.content.object-id = market

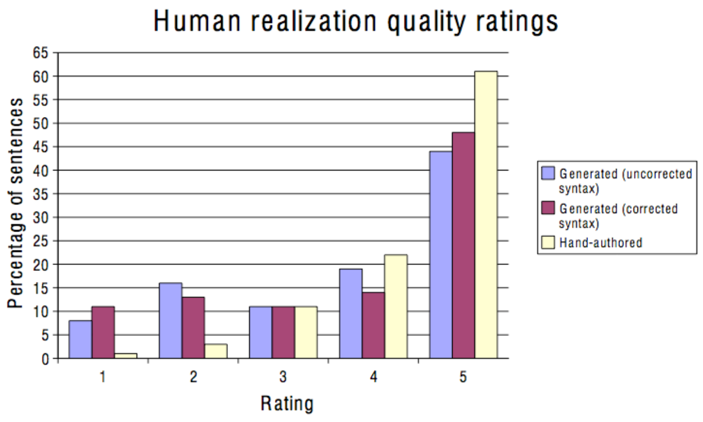
captain	addressee = captain-kirk
	dialogue-act.addressee = captain-kirk
	speech-act.addressee = captain-kirk

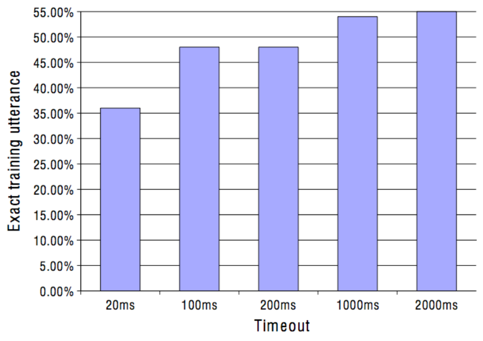
The grammar is a variant of Tree Adjoining Grammar (TAG; Joshi et al., 1975), a very famous and useful form of grammar. Each syntax tree node contains a feature structure with syntactic derivation information that allows it to be composed with others to form a legal grammatical sentence. DeVault trained the likelihood of each different structure from a corpus, obtaining a probabilistic TAG.

Generation proceeds as follows: Given an input, the slot-value pairs are used in the second table to identify the likely word (sequences) to include in the sentence. Given these words, the first table is used to identify likely tree fragments. These tree fragments are then composed in various ways to see if one (or more) coherent sentence(s) can be constructed. The probability scores on the various tree fragments provide preference information. Thus realization here is also a kind of search problem, now over various tree fragments, and beam search is also used here.

Note, interestingly, that if no connected tree can be constructed, the system is able to use the derivational feature structure information to 'hallucinate' tree fragments in places they will best help. In doing so, it is adding words into the sentence that were not licensed by the input of the second table. So its outputs might be wrong in two ways: it might not say what is supposed to be said, or it might even say things that were not in the input!

How well does this work? The left hand table below shows human ratings of the outputs (levels 1 to 5, 5 is best), and the right hand table shows the percentage of times the system produces *exactly* the sentence that the frame required, as per the training data (allowing the system more time to search, along the x-axis).





6. Conclusion

Statistical NLG is just in its infancy; it is probably the future, once there are large enough training corpora.

Optional Readings

NITROGEN: Knight, K. and V. Hatzivassiloglou. 1995. Two-Level, Many-Paths Generation. *Proceedings of the ACL-95 conference*. Cambridge, MA.

HALOGEN: Langkilde, I. 2000. Forest-Based Statistical Sentence Generation. *Proceedings of the NAACL-00 conference*.

HALOGEN: Langkilde, I. & S. Geary. 2002. A foundation for general purpose natural language generation sentence realization using probabilistic models of language.

FERGUS: Chen, J., S. Bangalore, O. Rambow and M. Walker. 2002. Towards Automatic Generation of Natural Language Generation Systems. *Proceedings International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan.

FERGUS: Bangalore, S. and O. Rambow. 2000. Exploiting a Probabilistic Hierarchical Model for Generation. *Proceedings of the COLING 2000 conference*. Saarbrücken, Germany.

Simple method: Bhagat, R. and E.H. Hovy. 2005. Trainable Reversible Language Analysis and Generation. Submitted.

DeVault's ICT system: DeVault, D., D. Traum, and R. Artstein. 2008. Practical Grammar-Based NLG from Examples. *Proceedings of the Fifth International Natural Language Generation Conference (INLG 2008)*, Ohio.

TAGs: Aravind K. Joshi, L. Levy, and M. Takahashi. 1975. Tree adjunct grammars. *Journal of the Computer and System Sciences* 10:136–163.

General: Ratnaparkhi. 2000. Trainable methods for surface natural language generation.

General: Zhong & Stent. 2005. Building surface realizers automatically from corpora using general purpose tools.

Lexical choice using statistics: Bangalore, S. and O. Rambow. 2000. Corpus-Based Lexical Choice in Natural Language Generation. *Proceedings ACL 2000 conference*. Hong Kong.

Evaluation: Bangalore, S., O. Rambow, S. Whittaker. 2000. Evaluation Metrics for Generation. *Proceedings of the International Conference on Natural Language Generation INLG 2000*. Mitzpe Ramon, Israel.