NLG Lecture 12: Statistical generation 1

19 March 2013

Johanna Moore

With thanks to Jon Oberlander
Irene Langkilde-Geary, Michael White and Albert Gatt

School of informatics

Advantages of using statistics

- Construction of NLG systems is extremely labour intensive!
  - e.g., Methodius system took ca. 2 years with 2.5 developers
- Many statistical approaches focus on specific modules
  - Example: stochastic content planner (Mellish et al., 1998)
    - generation as search; search as stochastic process
  - Best-studied: statistical realiser
    - realisers that take input in some canonical form and rely on language models to generate output
  - Advantages:
    - easily ported to new domains/applications
    - coverage can be increased (more data/training examples)

An Early Statistical NLG System

- Yngve (1962) built generator for machine translation that used a CFG and random number generator to produce "grammatical" sentences
- System randomly selects a production from those applicable at each point (starting from <S>)
- Randomly selects words to fill in word categories (<NOUN>, <VERB>, etc.)
- Example:
  - *The water under the wheels in oiled whistles and its polished shiny big and big trains is black.*

Overgeneration and ranking

- The core approaches we will consider rely on "overgenerate-and-rank" approach
  - also known as "generate and select"
- Given: input specification ("semantics" or canonical form)
  - Use a simple rule-based generator to produce many alternative realisations
  - Rank them using a language model
  - Output the best (= most probable) realisation

Advantages of overgeneration + ranking

- There are usually many ways to say the same thing.
  - e.g. ORDER(eat(you,chicken))
    - Eat chicken!
    - It is required that you eat chicken!
    - It is required that you eat poulet!
    - Poulet should be eaten by you.
    - You should eat chicken/chickens.
    - Chicken/Chickens should be eaten by you.

Where does the data come from?

- Some statistical NLG systems were built based on parallel data/text corpora.
  - allows direct learning of correspondences between content and output
  - rarely available
- Some work relies on Treebanks:
  - Extract input: process the treebank to extract "canonical specifications" from parsed sentences
  - train a language model
  - re-generate using a realiser and evaluate against original treebank

Nitrogen: Two-level generation for MT

- Langkilde's 1998 pioneering realisation system with wide coverage, i.e., handles many phenomena of English grammar
  - Based on overgeneration & ranking
- In the original Nitrogen
  - generator is a non-deterministic, symbolic generator
  - ranker is bigram or trigram language model.
- Application: Japanese/English MT, where the input to generation may lack crucial number information
  - Number agreement is treated as a "fluency" goal, since the propositional input doesn't specify it.
  - The n-gram model selects for number agreement

How well do statistical n-grams make linguistic decisions?

Subject-Verb Agreement

I am	2797
I are	47
I is	14

Article-Noun Agreement

a trust	394	an trust	0	the trust	1355
a trusts	2	an trusts	0	the trusts	115

Singular vs Plural

their trust	28
their trusts	8

Word Choice

reliance	567	trust	6100
reliances	0	trusts	1083

More Examples

Relative pronoun

visitor who 9 visitors who 20
visitor which 0 visitors which 0
visitor that 9 visitors that 14

Singular vs Plural

visitor 575 visitors 1083

Verb Tense

admire 212 admired 211
admires 107

Preposition

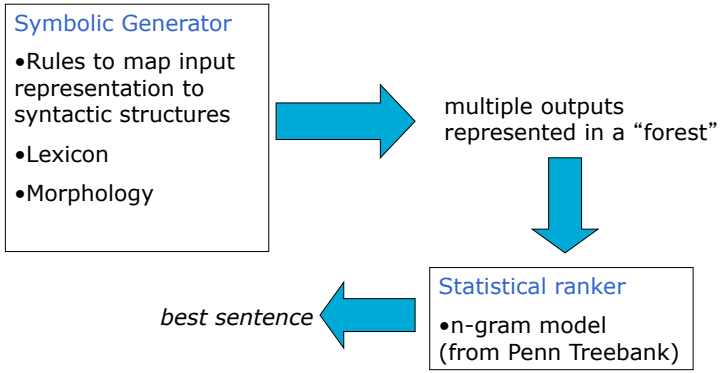
in Japan 5413 to Japan 1196

came to 2443 arrived in 544
came in 1498 arrived to 35
came into 244 arrived into 0

came to Japan 7 arrived to Japan 0
came into Japan 1 arrived into Japan 0
came in Japan 0 arrived in Japan 4

Structure of HALogen

- HALogen (Langkilde-Geary 2002) is a successor to Nitrogen
  - main differences:
    - representation data structure for possible realisation alternatives
    - HALogen handles more grammatical features



HALogen Input

Grammatical specification

```
(e1 / eat
   :subject (d1 / dog)
   :object (b1 / bone
         :premod (m1 / meaty))
   :adjunct (t1 / today))
```

Semantic specification

```
(e1 / eat
   :agent (d1 / dog)
   :patient (b1 / bone
         :premod (m1 / meaty))
   :temp-loc (t1 / today))
```

- Labelled feature-value representation specifying properties and relations of domain objects (e1, d1, etc)
- Recursively structured
- Order-independent
- Can be either grammatical or semantic (or mixture of both)
  - recasting mechanism maps from one to another

HALogen base generator

- Consists of about 255 hand-written rules
- Rules map an input representation into a packed set of possible output expressions.
  - Each part of the input is recursively processed by the rules, until only a string is left.
- Types of rules:
  - recasting
  - ordering
  - filling
  - morphing

Recasting

- Map semantic input representation to one that is closer to surface syntax

Semantic specification
```
(e1 /  eat
        :patient (b1 / bone
           :premod(m1 / meaty))
        :temp-loc(t1 / today)
        :agent (d1 / dog))
```

IF relation = :agent
AND sentence is not passive
THEN map relation to :subject

Grammatical specification
```
(e1 /  eat
        :object (b1 / bone
           :premod(m1 / meaty))
        :adjunct(t1 / today)
        :subject (d1 / dog))
```


Ordering

- Assign a linear order to the values in the input.

Grammatical specification
```
(e1 /  eat
        :object (b1 / bone
           :premod(m1 /
                      meaty))
        :adjunct(t1 / today)
        :subject (d1 / dog))
```

Put subject first unless sentence is passive.
Put adjuncts as sentence-final

Grammatical specification + order
```
(e1 /  eat
        :subject (d1 / dog)
        :object (b1 / bone
           :premod(m1 / meaty))
        :adjunct(t1 / today))
```


Filling

- If input is under-specified for some features, add all possible values for them.
  - NB: this allows for different degrees of specification, from minimally to maximally specified input.
  - Can create multiple "copies" of same input

Grammatical specification + order
```
(e1 /  eat
        :subject (d1 / dog)
        :object (b1 / bone
           :premod(m1 / meaty))
        :adjunct(t1 / today))
```

+:tense (past)

+:tense (present)

Morphing

- Given the properties of parts of the input, add the correct inflectional features.

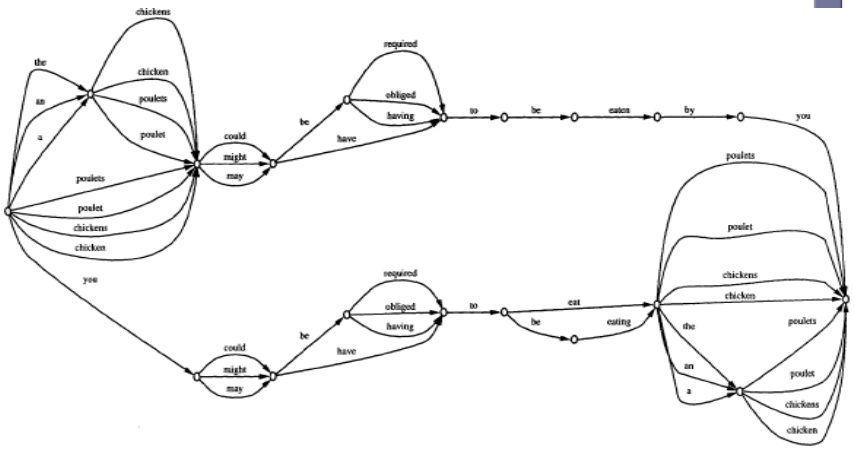
Grammatical specification + order
```
(e1 /  eat
        :tense(past)
        :subject (d1 / dog)
        :object (b1 / bone
           :premod(m1 / meaty))
        :adjunct(t1 / today))
```

Grammatical specification + order
```
(e1 /  ate
        :subject (d1 / dog)
        :object (b1 / bone
           :premod(m1 / meaty))
        :adjunct(t1 / today))
```

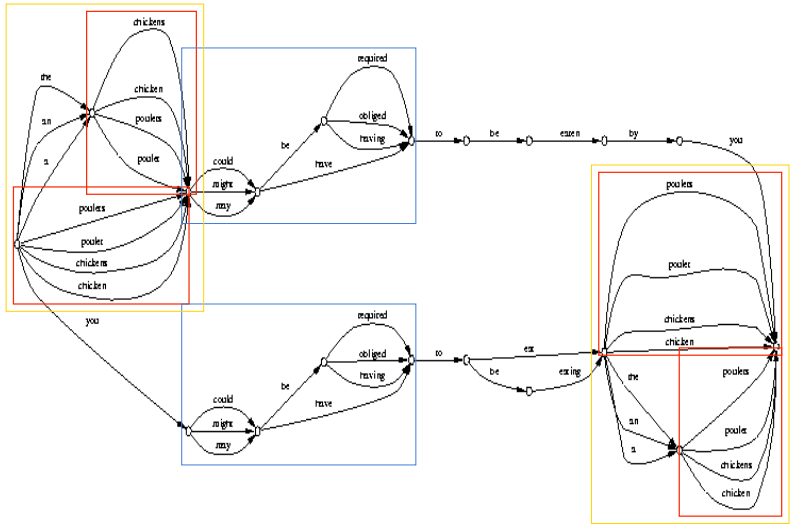
The output of the base generator

- Problem:
  - a single input may have literally hundreds of possible realisations after base generation
  - these need to be represented in an efficient way to facilitate search for the best output
- Options:
  - word lattice
  - forest of trees
  - (and in OpenCCG, chart)

Option 1: lattice structure (Langkilde 2000)



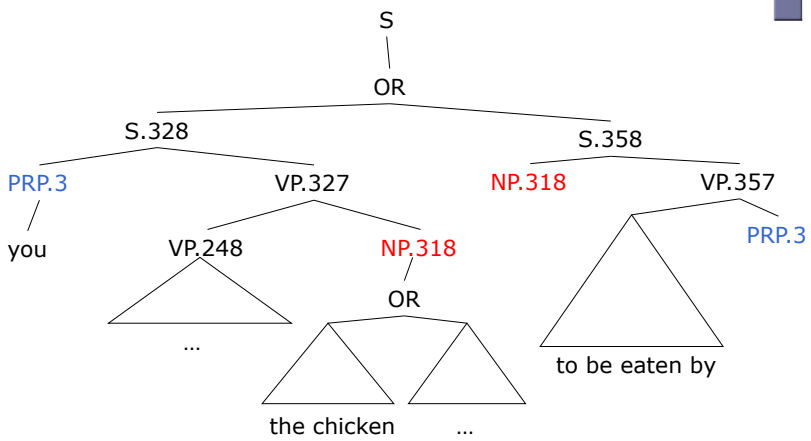
"You may have to eat chicken": 576 versions



Properties of lattices

- In a lattice, a complete left-right path represents a possible sentence.
- Lots of duplication!
  - e.g., word "chicken" occurs multiple times
  - ranker will be scoring the same substring more than once
- In a lattice path, every word is dependent on all other words.
  - can't model local dependencies

Option 2: Forests (Langkilde 2000,2002)



Properties of forests

- Efficient representation:
  - each individual constituent represented only once, with pointers
  - alternatives represented by disjunctive ("OR") nodes
  - ranker will only compute a partial score for a subtree once
- Equivalent to a non-recursive context-free grammar
  - S.469 → S.328
  - S.469 → S.358
  - …

Statistical ranking

- Uses n-gram language models to choose the best realisation *r*:

$$r_{best} = \arg\max_{r \in forest} \prod_{i=1}^{n} P(w_i \mid w_1 \ldots w_{i-1})$$

$$= \arg\max_{r \in forest} \prod_{i=1}^{n} P(w_i \mid w_{i-1}) \text{ [Markov assumption]}$$

Sample Results of Bigram model

Random path: (out of a set of 11,664,000 semantically-related sentences)

Visitant which came into the place where it will be Japanese has admired that there was Mount Fuji.

Top three:

Visitors who came in Japan admire Mount Fuji .

Visitors who came in Japan admires Mount Fuji .

Visitors who arrived in Japan admire Mount Fuji .

Strengths

- Reflects reality that 55% (Stolke et al. 1997) of dependencies are binary, and between adjacent words
- Embeds linear ordering constraints

Performance of HALogen

Minimally specified input frame (bigram model):

- *It would sell its fleet age of Boeing Co. 707s because of maintenance costs increase the company announced earlier.*

Minimally specified input frame (trigram model):

- *The company earlier announced it would sell its fleet age of Boeing Co. 707s because of the increase maintenance costs.*

Almost fully specified input frame:

- *Earlier the company announced it would sell its aging fleet of Boeing Co. 707s because of increased maintenance costs.*

Limitations of Bigram model

Example	Reason
Visitors come in Japan.	A three-way dependency
He planned increase in sales.	Part-of-speech ambiguity
A tourist who admire Mt. Fuji...	Long-distance dependency
A dog eat/eats bone.	Previously unseen ngrams
I cannot sell their trust.	Nonsensical head-arg relationship
The methods must be modified to *the circumstances.*	Improper subcat structure

N-gram models: strengths and weaknesses

Strengths:

- Fully automatic method for ranking realizations
- Easy to train
- Based on statistical models, so are robust
- Can potentially be used for deep as well as surface generation

But:

- The usual issues with n-gram models apply:
  - bigger *n* → better output, but more data sparsity
- Expensive to apply
  - To rank candidates, have to generate all possible realizations and compute probabilities
- Bias towards shorter strings

Generation as decision making (Belz, 2005)

- Start with input data, trace a path of decisions to final realization

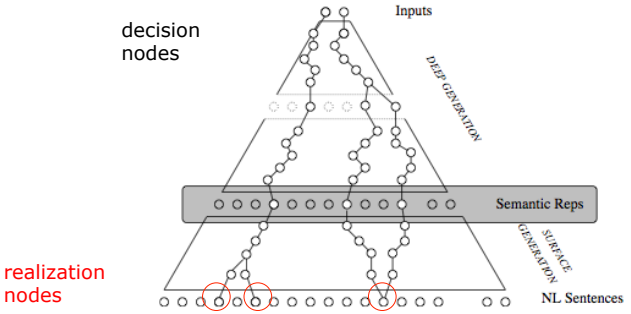
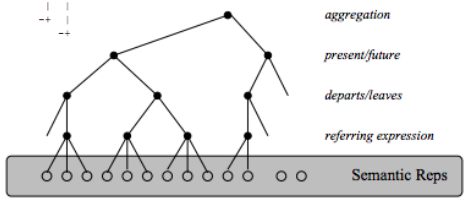


Figure 1: Generation space as decision tree.

Closer Look at Generation Space

```
+-                                          -+
| message-id:msg02                           |
| relation: DEPARTURE                        |
|             +-                         -+ |
|             | departing-entity: CALEDONIAN-EXPRESS | |
| arguments:  | departure-location: ABERDEEN | |
|             | departure-time: 1000 | |
|             +-                         -+ |
+-                                          -+
```




```
DEPARTING_EVENT --> TRAIN DEPART
TRAIN --> the Caledonian Express
TRAIN --> the train
TRAIN --> it
DEPART --> leaves LOCATION TIME
DEPART --> departs LOCATION TIME
DEPART --> departs from LOCATION
LOCATION --> Aberdeen
TIME --> at 10am
TIME --> at 10 o'clock in the mo
```

it will leave at 12am
the train will leave at 12am
the caledonian express will leave at 12am
it leaves at 12am
the train leaves at 12am
the caledonian express leaves at 12am
it departs at 12am
the train departs at 12am
the caledonian express departs at 12am
it leaves at 12am from platform 4
the train leaves at 12am from platform 4

Belz - comparing rankers (n-gram vs treebanks)

Weather domain example

- Built basic generator semi-automatically
  reflects all the variation found in the corpus
- Gold standard = human generated weather report
- Compare bigrams with rankers trained on treebank, with different (local vs global) decision algorithms
  - Given a treebank, can compute the probability of each expansion of non-terminal in induced CFG grammar
  - Treebank rankers compute probability of a string given its semantics

10Nov2000_16/1:

Gold	N 10 OR LESS VEERING SE AND RISING 20-24 LATE IN THE PERIOD
Random	N 10 OR LESS THEN LATE IN PERIOD BACKING SE STEADILY INCREASING TO 20-24 IN FRONTAL ZONE FOR A TIME EARLY EVENING
Greedy	SOON N 10 OR LESS VEERING SE INCREASING 20-24 LATER
Viterbi	SOON N 10 OR LESS VEERING SE INCREASING 20-24 LATER
2-gram	N 10 OR LESS BACKING SE AND 20-24 BY EVENING

Belz - comparing rankers (n-gram vs treebanks)

- Bigrams:
  - Reduce error compared to treebank-based rankers
  - But quite slow

	Training set	Test set
Random	14.92	14.37
TT/Greedy	-50.64% (7.36)	-50.30% (7.41)
TT/Viterbi	-52.32% (7.11)	-52.11% (7.14)
2-gram	-58.44% (6.20)	-57.91% (6.27)

% reduction in string edit distance, absolute in ()

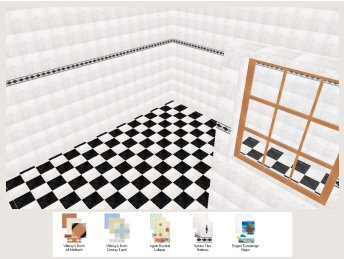
Total time (minutes)	Training set	Test set
TT/greedy	23m52s	2m06s
TT/Viterbi	3h22m33s	27m05s
2-gram	24h04m05s	3h35m06s

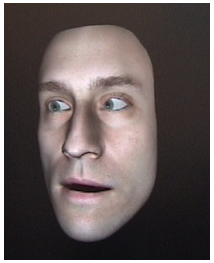
Current approaches to statistical generation

- Different levels of generation process can be addressed
  - In MATCH, sentence planner is trained (Next lecture)
- But most work has looked at realization
- N-grams give best results, but are inefficient
- OpenCCG combines a principled (and thus only mildly overgenerating) grammar with an efficient approach to overgeneration and ranking (White 2004, 2006).

COMIC: Conversational Multimodal Interaction with Computers

OpenCCG was developed for COMIC, a multimodal generation system with an embodied conversational agent (ECA).





"This design is also modern. The tiles draw from the Helenus collection, by Sphinx Tiles. It features …"

Project Team: Mary Ellen Foster, John Lee, Johanna Moore, Jon Oberlander, Michael White

The OpenCCG Realiser was first developed in COMIC

- Surface realiser based on Steedman's theory of Combinatory Categorial Grammar
  - Input: logical form (meaning representation)
  - Output: text with prosodic markup (APML) suitable for Festival speech synthesizer
- Novel ensemble of efficiency methods, with integrated n-gram scoring
- First implementation of CCG realisation that is practical for dialogue systems (!)

Adapted from slide by Michael White

OpenCCG Efficiency methods

1. Small set of hand-crafted rules for chunking input logical forms into sub-problems to be solved independently before combination
   - Solves problem that chart realizers waste time generating paths containing semantically incomplete phrases
2. Prune edges from chart based on n-gram score
3. Formulate search as a best-first anytime search using n-gram scores to sort edges on the agenda

- Basic Idea: ensure a good realization can be fund quickly, even when it would take a long time to find best realization out of all possible realizations.

(White, 2006)

Case Study

- White measured effect of n-grams on accuracy and search times
- COMIC test suite
  - HLDS LF/target pairs from the COMIC system
  - Example (with pitch accents and boundary tones)

    once_again_L+H LH% there are floral_H* motifs_H* LH% and geometric_H* shapes_H* on the decorative_H* tiles LL% , but L here_L+H* LH% the colours are off_white_H* LH% and dark_red_H* LL% .*

Adapted from slide by Michael White

Experiment

- Baseline 1: no n-gram scoring, breadth-first search, other efficiency methods in use
- Baseline 2: same, but with depth-first search
- Topline: n-gram scoring based on target string
- All efficiency methods, 3-best pruning
  - Accuracy = exact match
  - Score = modified BLEU (n-gram precision) score
  - Times in ms.

N-grams

- N-gram models: deliver probabilities of words given n-1 previous words
- 25-fold cross-validation for training models
- 5-gram backoff models with semantic class replacement
  - *the tiles draw from SERIES_H* LL% , by MANUFACTURER_H* LL% .*
- Modifier order and type/placement of boundary tones largely determined by n-grams

White's OpenCCG results

			Time 'til First		*Time 'til Best*	
	Accuracy	*Score*	*Mean (±σ)*	*Max*	*Mean (±σ)*	*Max*
Baseline 1	284/549	0.78	497 (±380)	2273	497 (±380)	2273
Baseline 2	41/549	0.38	400 (±286)	1932	400 (±286)	1932
Topline	549/549	1	152 (±93)	744	154 (±95)	744
CV-25	548/549	0.99	206 (±138)	1013	206 (±138)	1013

➢ With n-gram scoring, possible to get near perfect results!

Summary

- Most statistical approaches to generation have focused on realization
- Many have used over-generation and ranking
  - Inspired by approaches to Machine Translation
- OpenCCG offers state of the art facilities
  - N-grams can get you a long way, so long as the process is "reined in"
  - Where do its n-grams come from?
    - Corpora … processed into language models

References

- Anja Belz. 2005. Statistical Generation: Three Methods Compared and Evaluated. In ENLG-05.
- Kevin Knight and Vasileios Hatzivassiloglou.1995.Two-level, many-paths generation. In Proc. ACL-95.
- I. Langkilde and K. Knight. 1998. Generation that exploits corpus-based statistical knowledge. In Proc COLING-ACL-98, 704-710.
- Irene Langkilde. 2000. Forest-based statistical sentence generation. In Proc. NAACL-00.
- Irene Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In Proc. INLG-02.
- Mellish, C., Knott, A., Oberlander, J., & O'Donnell, M., 1998. Experiments using stochastic search for text planning. In Proc. INLG-9.
- M. Walker, O. Rambow, and M. Rogati. 2001. SPoT: A trainable sentence planner. In Proc. NAACL- 01.
- Michael White. 2004. Reining in CCG Chart Realization. In Proc. INLG-04.
- Michael White. 2006. Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. *Research on Language and Computation* 4:39-75.