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
Adapted from slide by Albert Gatt

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

<table>
<thead>
<tr>
<th></th>
<th>I am</th>
<th>I are</th>
<th>I is</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2797</td>
<td>47</td>
<td>14</td>
</tr>
</tbody>
</table>

Article-Noun Agreement

<table>
<thead>
<tr>
<th></th>
<th>a trust</th>
<th>an trust</th>
<th>the trust</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>394</td>
<td>0</td>
<td>1355</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td>reliance</td>
<td>trust</td>
<td>6100</td>
</tr>
<tr>
<td></td>
<td>567</td>
<td>trust</td>
<td>1083</td>
</tr>
</tbody>
</table>

Singular vs Plural

<table>
<thead>
<tr>
<th></th>
<th>their trust</th>
<th>their trusts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>28</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>reliance</td>
<td>trusts</td>
</tr>
<tr>
<td></td>
<td>567</td>
<td>1083</td>
</tr>
</tbody>
</table>
**More Examples**

<table>
<thead>
<tr>
<th>Relative pronoun</th>
<th>Preposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>visitor who</td>
<td>in</td>
</tr>
<tr>
<td>visitors who</td>
<td>in Japan</td>
</tr>
<tr>
<td>came to</td>
<td>in 5413</td>
</tr>
<tr>
<td>came in</td>
<td>in 1498</td>
</tr>
<tr>
<td>came into</td>
<td>in 244</td>
</tr>
<tr>
<td>visited</td>
<td>in Japan</td>
</tr>
<tr>
<td>visitors that</td>
<td>arrived in</td>
</tr>
<tr>
<td>visitors which</td>
<td>arrived in</td>
</tr>
<tr>
<td>visitors</td>
<td>in Japan</td>
</tr>
<tr>
<td>verb</td>
<td>to</td>
</tr>
<tr>
<td>to</td>
<td>arrived to</td>
</tr>
<tr>
<td>singular</td>
<td>vs</td>
</tr>
<tr>
<td>plural</td>
<td></td>
</tr>
<tr>
<td>verb</td>
<td></td>
</tr>
<tr>
<td>to Japan</td>
<td></td>
</tr>
<tr>
<td>arrived to Japan</td>
<td></td>
</tr>
</tbody>
</table>

**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
- Symbolic Generator
  - Rules to map input representation to syntactic structures
    - Lexicon
    - Morphology
- Statistical ranker
  - n-gram model (from Penn Treebank)
  - Multiple outputs represented in a “forest”
  - Best sentence

**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
  :agent (d1 / dog))

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

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

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

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.

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

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))

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)

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)**

- 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 \( r \):

\[
 r_{best} = \arg \max_{r \in \text{forest}} \prod_{j=1}^{n} P(w_j | w_1...w_{j-1}) \\
= \arg \max_{r \in \text{forest}} \prod_{j=1}^{n} P(w_j | w_{j-1}) \text{[Markov assumption]}
\]

**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

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

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 $n \rightarrow$ 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

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

10 Nov 2000 16/1:
Gold
- N 10 or less, veering SE and rising 20-24 later in the period
- Some period backing SE steadily increasing to 20-24 in frontal
- Somewhere for a time early evening
Random
- N 10 or less
- Tomorrow evening
Viterbi
- N 10 or less
- Tomorrow evening
2-gram
- N 10 or less
- Increasing 20-24 by evening

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

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

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 found quickly, even when it would take a long time to find best realization out of all possible realizations.

(White, 2006)

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 (!)

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%.
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

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Score</th>
<th>Time ’til First Mean (±σ)</th>
<th>Time ’til Best Mean (±σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline 1</strong></td>
<td>284/549</td>
<td>0.78</td>
<td>497 (±380)</td>
<td>2273</td>
</tr>
<tr>
<td><strong>Baseline 2</strong></td>
<td>41/549</td>
<td>0.38</td>
<td>400 (±286)</td>
<td>1932</td>
</tr>
<tr>
<td><strong>Topline</strong></td>
<td>549/549</td>
<td>1</td>
<td>152 (±193)</td>
<td>744</td>
</tr>
<tr>
<td><strong>CV-25</strong></td>
<td>548/549</td>
<td>0.99</td>
<td>206 (±138)</td>
<td>1013</td>
</tr>
</tbody>
</table>

- 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