Natural Language Generation

(mostly) for Spoken Dialogue Systems

Ondřej Dušek

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University in Prague

May 11th, 2016

Outline of this talk

1. Introduction: what is NLG?

Outline of this talk

1. Introduction: what is NLG?
   a) Textbook NLG pipeline
   b) How real systems differ

Outline of this talk

1. Introduction: what is NLG?
   a) Textbook NLG pipeline
   b) How real systems differ
2. Examples of real NLG systems

Outline of this talk

1. Introduction: what is NLG?
   a) Textbook NLG pipeline
   b) How real systems differ
2. Examples of real NLG systems **(many)**

Outline of this talk

1. Introduction: what is NLG?
   a) Textbook NLG pipeline
   b) How real systems differ
2. Examples of real NLG systems **(many)**
   - different stages of NLG

Outline of this talk

1. Introduction: what is NLG?
   a) Textbook NLG pipeline
   b) How real systems differ
2. Examples of real NLG systems **(many)**
   - different stages of NLG
   - past & state-of-the-art

Outline of this talk

1. Introduction: what is NLG?
   a) Textbook NLG pipeline
   b) How real systems differ
2. Examples of real NLG systems **(many)**
   - different stages of NLG
   - past & state-of-the-art
   - including our work

Outline of this talk

1. Introduction: what is NLG?
   a) Textbook NLG pipeline
   b) How real systems differ
2. Examples of real NLG systems **(many)**
   - different stages of NLG
   - past & state-of-the-art
   - including our work
3. What next?

Introduction

Objective of NLG

Given (whatever) input and a **communication goal**, create a natural language string that is **well-formed** and **human-like**.

Introduction

Objective of NLG

Given (whatever) input and a **communication goal**, create a natural language string that is **well-formed** and **human-like**.

- Desired properties: variation, simplicity, trainability (?)

Introduction

Objective of NLG

Given (whatever) input and a **communication goal**, create a natural language string that is **well-formed** and **human-like**.

- Desired properties: variation, simplicity, trainability (?)
- Actually not a very well-defined task

Introduction

Objective of NLG

Given (whatever) input and a **communication goal**, create a natural language string that is **well-formed** and **human-like**.

- Desired properties: variation, simplicity, trainability (?)
- Actually not a very well-defined task

Usage

- Spoken dialogue systems
- Machine translation
- Short texts: Personalized letters, weather reports …
- Summarization
- Question answering in knowledge bases

Standard NLG Pipeline (*Textbook*)

[Inputs]

Standard NLG Pipeline (*Textbook*)

[Inputs]

↓ Content/text planning (“what to say”)

- Content selection, basic ordering

[Content plan]

Standard NLG Pipeline (*Textbook*)

[Inputs]

↓ Content/text planning ("what to say")

- Content selection, basic ordering

[Content plan]

↓ Sentence planning/microplanning ("middle ground")

- aggregation, lexical choice, referring…

[Sentence plan(s)]

Standard NLG Pipeline (*Textbook*)

[Inputs]

↓ Content/text planning ("what to say")

- Content selection, basic ordering

[Content plan]

↓ Sentence planning/microplanning ("middle ground")

- aggregation, lexical choice, referring…

[Sentence plan(s)]

↓ Surface realization ("how to say it")

- linearization, conforming to rules of the target language

[Text]

Standard NLG Pipeline (*Textbook*)

Inputs

- Communication goal (e.g. “inform user about search results”)
- Knowledge base (e.g. list of matching entries in database, weather report numbers etc.)
- User model (constraints, e.g. user wants short answers)
- Dialogue history (referring expressions, repetition)

Standard NLG Pipeline (*Textbook*)

Inputs

- Communication goal (e.g. "inform user about search results")
- Knowledge base (e.g. list of matching entries in database, weather report numbers etc.)
- User model (constraints, e.g. user wants short answers)
- Dialogue history (referring expressions, repetition)

Content planning

- Content selection according to communication goal
- Basic structuring (ordering)

Standard NLG Pipeline (*Textbook*)

Sentence planning (micro-planning)

- Word and syntax selection (e.g. choose templates)
- Dividing content into sentences
- Aggregation (merging simple sentences)
- Lexicalization
- Referring expressions

Standard NLG Pipeline (*Textbook*)

Sentence planning (micro-planning)

- Word and syntax selection (e.g. choose templates)
- Dividing content into sentences
- Aggregation (merging simple sentences)
- Lexicalization
- Referring expressions

Surface realization

- Creating linear text from (typically) structured input
- Ensuring grammatical correctness

Real NLG Systems

Few systems implement the whole pipeline

- Systems focused on content planning with trivial surface realization
- Surface-realization-only, word-order-only systems
- One-step (holistic) approaches
- SDS: content planning done by dialogue manager
  → **only sentence planning and realization here**

Real NLG Systems

Few systems implement the whole pipeline

- Systems focused on content planning with trivial surface realization
- Surface-realization-only, word-order-only systems
- One-step (holistic) approaches
- SDS: content planning done by dialogue manager
  - → **only sentence planning and realization here**

Approaches

- Templates, grammars, rules, statistics, or a mix thereof

Real NLG Systems

Few systems implement the whole pipeline

- Systems focused on content planning with trivial surface realization
- Surface-realization-only, word-order-only systems
- One-step (holistic) approaches
- SDS: content planning done by dialogue manager
  $\rightarrow$ **only sentence planning and realization here**

Approaches

- Templates, grammars, rules, statistics, or a mix thereof

Data representations

- Varied, custom-tailored, non-compatible

Two-step or one-step?

Why go two-step

- Dividing makes the tasks simpler
  - no need to worry about morphology in sentence planning

Two-step or one-step?

Why go two-step

- Dividing makes the tasks simpler
  - no need to worry about morphology in sentence planning
- Surface realization can be rule based
  - you can hardcode the grammar, it is more straightforward to fix

Two-step or one-step?

Why go two-step

- Dividing makes the tasks simpler
  - no need to worry about morphology in sentence planning
- Surface realization can be rule based
  - you can hardcode the grammar, it is more straightforward to fix
- Surface realizer is (relatively) easy to implement
  - and you can use a third-party one

Two-step or one-step?

Why go two-step

- Dividing makes the tasks simpler
  - no need to worry about morphology in sentence planning
- Surface realization can be rule based
  - you can hardcode the grammar, it is more straightforward to fix
- Surface realizer is (relatively) easy to implement
  - and you can use a third-party one

Why go one-step

- Problem of all pipelines: error propagation
  - the more steps, the more chance to screw it up

Two-step or one-step?

Why go two-step

- Dividing makes the tasks simpler
  - no need to worry about morphology in sentence planning
- Surface realization can be rule based
  - you can hardcode the grammar, it is more straightforward to fix
- Surface realizer is (relatively) easy to implement
  - and you can use a third-party one

Why go one-step

- Problem of all pipelines: error propagation
  - the more steps, the more chance to screw it up
- Need to provide training sentence plans (statistical planners)
  - sometimes you may use existing analysis tools

NLG systems examples

- Divided by NLG stage:

NLG systems examples

- Divided by NLG stage:
  1. Sentence planning

NLG systems examples

- Divided by NLG stage:
  1. Sentence planning
  2. Surface realization

NLG systems examples

- Divided by NLG stage:
  1. Sentence planning
  2. Surface realization
  3. One-step approaches to NLG

NLG systems examples

- Divided by NLG stage:
  1. Sentence planning
  2. Surface realization
  3. One-step approaches to NLG
- Each stage:
  1. History
  2. Current state-of-the art / our works

Sentence Planning Examples

- Various input/output formats, not very comparable

Sentence Planning Examples

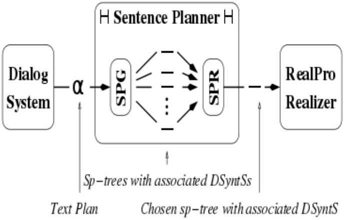
- Various input/output formats, not very comparable
- Actually typically handcrafted or non-existent

Sentence Planning Examples

- Various input/output formats, not very comparable
- Actually typically handcrafted or non-existent
  - One-step approaches or simplistic systems
- Here we focus on trainable approaches
  - …and especially on our own ☺

Trainable Sentence Planning: *SPoT*

- Spoken Dialogue System in the flight information domain
- Handcrafted generator + overgeneration
- Statistical reranker (RankBoost) trained on hand-annotated sentence plans



implicit-confirm(orig-city:NEWARK)
implicit-confirm(dest-city:DALLAS)
implicit-confirm(month:9)
implicit-confirm(day-number:1)
request(depart-time)

Alt	Realization	H	RB
0	What time would you like to travel on September the 1st to Dallas from Newark?	5	.85
5	Leaving on September the 1st. What time would you like to travel from Newark to Dallas?	4.5	.82
8	Leaving in September. Leaving on the 1st. What time would you, traveling from Newark to Dallas, like to leave?	2	.39

Trainable Sentence Planning: Parameter Optimization

- Requires a flexible handcrafed planner
- No overgeneration
- Adjusting its parameters “somehow”

Trainable Sentence Planning: Parameter Optimization

- Requires a flexible handcrafed planner
- No overgeneration
- Adjusting its parameters “somehow”

I see, oh Chimichurri Grill is a latin american place with sort of poor atmosphere. Although it doesn’t have rather nasty food, its price is 41 dollars. I suspect it’s kind of alright.	extra=2.50 ems=4.50 agree=3.50 consc=4.75 open=4.25
Did you say Ce-Cent’anni? I see, I mean, I would consider it because it has friendly staff and tasty food, you know buddy.	extra=4.75 ems=5.00 agree=6.25 consc=6.25 open=5.25

Examples

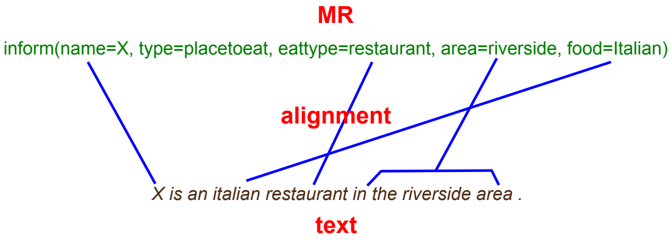
- *Paiva&Evans*: linguistic features annotated in corpus generated with many parameter settings, correlation analysis
- *PERSONAGE-PE*: personality traits connected to linguistic features via machine learning

Our A∗/Perceptron Sentence Planner (*TGEN1*)

1. Requires no handcrafted module
2. Learns from unaligned data

Our A∗/Perceptron Sentence Planner (*TGEN1*)

1. Requires no handcrafted module
2. Learns from unaligned data
   - Typical NLG training:
     a) requires alignment of MR elements and words/phrases
     b) uses a separate alignment step



Our A∗/Perceptron Sentence Planner (*TGEN1*)

1. Requires no handcrafted module
2. Learns from unaligned data
   - Typical NLG training:
     a) requires alignment of MR elements and words/phrases
     b) uses a separate alignment step
   - Our sentence planner learns alignments jointly
     - training from pairs: **MR + sentence**

MR

inform(name=X, type=placetoeat, eattype=restaurant, area=riverside, food=Italian)

X is an italian restaurant in the riverside area .

text

I/O formats

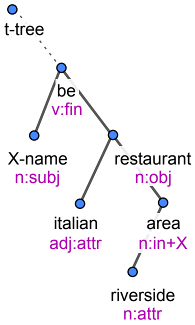
inform(name=X, type=placetoeat,
eattype=restaurant,
area=riverside, food=Italian)

- **Input**: a MR
  - dialogue acts: “inform” + slot-value pairs
  - other formats possible

I/O formats

- **Input**: a MR
  - dialogue acts: “inform” + slot-value pairs
  - other formats possible
- **Output**: deep-syntax dependency trees
  - based on *TectoMT*'s t-layer, but very simplified
  - two attributes per tree node: *t-lemma* + *formeme*
  - using surface word order

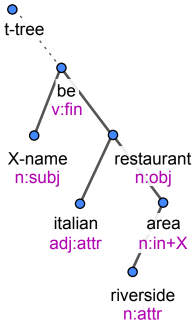
inform(name=X, type=placetoeat,
eattype=restaurant,
area=riverside, food=Italian)



I/O formats

- **Input**: a MR
  - dialogue acts: “inform” + slot-value pairs
  - other formats possible
- **Output**: deep-syntax dependency trees
  - based on *TectoMT*'s t-layer, but very simplified
  - two attributes per tree node:
    t-lemma + *formeme*
  - using surface word order
- Conversion to plain text sentences – surface realization
  - *Treex*/*TectoMT* English synthesis (rule-based, later)

inform(name=X, type=placetoeat,
eattype=restaurant,
area=riverside, food=Italian)



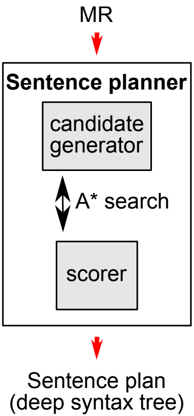
X is an Italian restaurant in the riverside area.

Overall Structure of Our Sentence Planner

- A*-style search – “finding the path”
  empty tree $\rightarrow$ full sentence plan tree
  - always expand the most promising candidate sentence plan
  - stop when candidates don't improve for a while

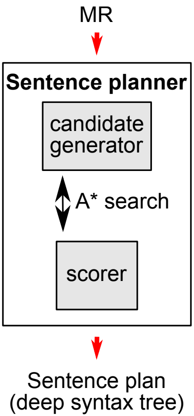
Overall Structure of Our Sentence Planner

- A*-style search – “finding the path”
  empty tree $\rightarrow$ full sentence plan tree
  - always expand the most promising candidate sentence plan
  - stop when candidates don't improve for a while
- Using two subcomponents:
  - **candidate generator**
    - churning out candidate sentence plan trees
    - given an incomplete candidate tree, add node(s)



Overall Structure of Our Sentence Planner

- A*-style search – "finding the path"
  empty tree $\rightarrow$ full sentence plan tree
  - always expand the most promising candidate sentence plan
  - stop when candidates don't improve for a while
- Using two subcomponents:
  - **candidate generator**
    - churning out candidate sentence plan trees
    - given an incomplete candidate tree, add node(s)
  - **scorer**/ranker for the candidates
    - influences which candidate trees will be expanded (selects the most promising)

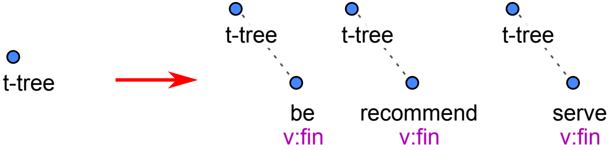


Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)

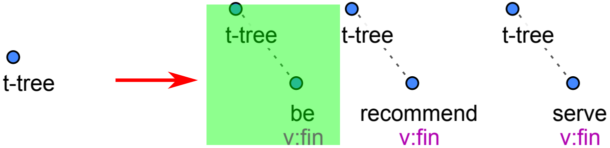
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)



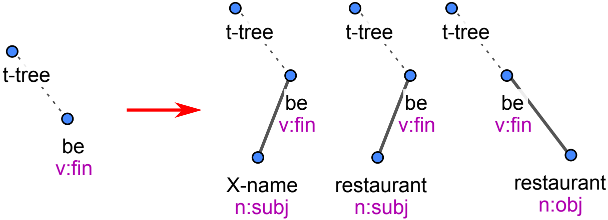
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)



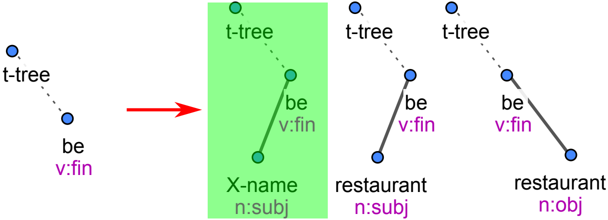
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)



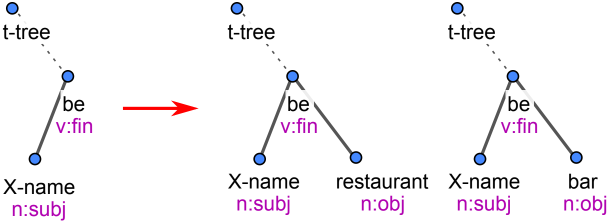
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)



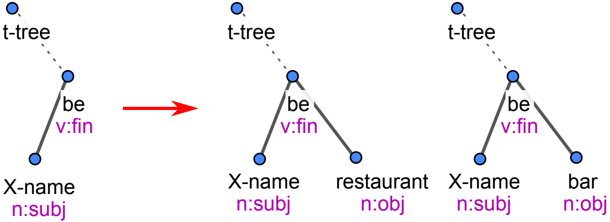
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)



Candidate generator

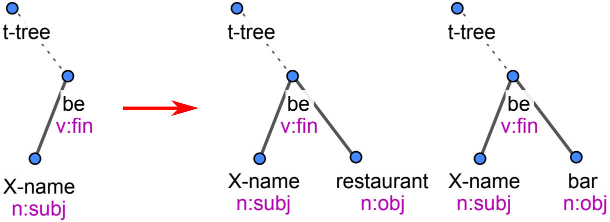
- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)



- "possible places" must be limited in practice

Candidate generator

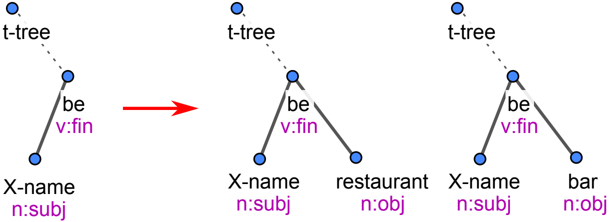
- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)



- “possible places” must be limited in practice
- using combination of things seen in training data

Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)



- "possible places" must be limited in practice
- using combination of things seen in training data
  - parent–child
  - t-lemma + formeme
  - number of children, tree size …

Scorer

- a function:

  sentence plan tree t, MR $m \rightarrow$ real-valued score

  - describes the fitness of t for m

Scorer

- a function:

  sentence plan tree t, MR $m \rightarrow$ real-valued score

  - describes the fitness of t for m

How to describe the fitness?

Scorer

- a function:

  sentence plan tree t, MR $m \rightarrow$ real-valued score

  - describes the fitness of t for m

How to describe the fitness?

Features:

Scorer

- a function:

  sentence plan tree t, MR $m \rightarrow$ real-valued score

  - describes the fitness of t for m

How to describe the fitness?

Features:

- occurence of input DA slots + t-lemmas / formemes

Scorer

- a function:

  sentence plan tree t, MR $m \rightarrow$ real-valued score

  - describes the fitness of t for m

How to describe the fitness?

Features:

- occurence of input DA slots + t-lemmas / formemes
- tree shape

Scorer

- a function:

  sentence plan tree t, MR $m \rightarrow$ real-valued score

  - describes the fitness of t for m

How to describe the fitness?

Features:

- occurence of input DA slots + t-lemmas / formemes
- tree shape
- tree edges (parent-child)

Scorer

- a function:

  sentence plan tree *t*, MR *m* $\rightarrow$ real-valued score

  - describes the fitness of *t* for *m*

How to describe the fitness?

Features:

- occurence of input DA slots + t-lemmas / formemes
- tree shape
- tree edges (parent-child)
- …

Perceptron scorer

Basic form

- score $= \mathbf{w}^\top \cdot \text{feat}(t, m)$

Perceptron scorer

Basic form

- score $= \mathbf{w}^\top \cdot \text{feat}(t, m)$
- Training:
  - given m, generate the best tree t_{top} with current weights
  - update weights if $t_{top} \neq t_{gold}$ (gold-standard)

Perceptron scorer

Basic form

- score $= \mathbf{w}^\top \cdot \text{feat}(t, m)$
- Training:
  - given m, generate the best tree t_{top} with current weights
  - update weights if $t_{top} \neq t_{gold}$ (gold-standard)
- Update: $\mathbf{w} = \mathbf{w} + \alpha \cdot (\text{feat}(t_{gold}, m) - \text{feat}(t_{top}, m))$

Perceptron scorer

Basic form

- score $= \mathbf{w}^\top \cdot \mathrm{feat}(t, m)$
- Training:
  - given m, generate the best tree t_{top} with current weights
  - update weights if $t_{top} \neq t_{gold}$ (gold-standard)
- Update: $\mathbf{w} = \mathbf{w} + \alpha \cdot (\mathrm{feat}(t_{gold}, m) - \mathrm{feat}(t_{top}, m))$

Our improvements

Trying to guide the search on incomplete trees

Perceptron scorer

Basic form

- score $= \mathbf{w}^\top \cdot \text{feat}(t, m)$
- Training:
  - given m, generate the best tree t_{top} with current weights
  - update weights if $t_{top} \neq t_{gold}$ (gold-standard)
- Update: $\mathbf{w} = \mathbf{w} + \alpha \cdot (\text{feat}(t_{gold}, m) - \text{feat}(t_{top}, m))$

Our improvements

Trying to guide the search on incomplete trees

- Updates based on partial trees

Perceptron scorer

Basic form

- score $= \mathbf{w}^\top \cdot \text{feat}(t, m)$
- Training:
  - given m, generate the best tree t_{top} with current weights
  - update weights if $t_{top} \neq t_{gold}$ (gold-standard)
- Update: $\mathbf{w} = \mathbf{w} + \alpha \cdot (\text{feat}(t_{gold}, m) - \text{feat}(t_{top}, m))$

Our improvements

Trying to guide the search on incomplete trees

- Updates based on partial trees
- Estimating future value of the trees

Evaluation of Our Sentence Planner

Data

- Restaurant recommendations from the *BAGEL* generator
  - restaurant location, food type, etc.
  - just 404 utterances for 202 DAs

Evaluation of Our Sentence Planner

Data

- Restaurant recommendations from the *BAGEL* generator
  - restaurant location, food type, etc.
  - just 404 utterances for 202 DAs

Results

- basic setup 54.24% BLEU, best version 59.89%

Evaluation of Our Sentence Planner

Data

- Restaurant recommendations from the *BAGEL* generator
  - restaurant location, food type, etc.
  - just 404 utterances for 202 DAs

Results

- basic setup 54.24% BLEU, best version 59.89%
- less than *BAGEL*'s ~ 67% BLEU…

Evaluation of Our Sentence Planner

Data

- Restaurant recommendations from the *BAGEL* generator
  - restaurant location, food type, etc.
  - just 404 utterances for 202 DAs

Results

- basic setup 54.24% BLEU, best version 59.89%
- less than *BAGEL*'s ~ 67% BLEU… but:
  - we do not use alignments
  - our generator decides itself what to include

Evaluation of Our Sentence Planner

Data

- Restaurant recommendations from the *BAGEL* generator
  - restaurant location, food type, etc.
  - just 404 utterances for 202 DAs

Results

- basic setup 54.24% BLEU, best version 59.89%
- less than *BAGEL*'s ~ 67% BLEU… but:
  - we do not use alignments
  - our generator decides itself what to include
- outputs mostly fluent and meaningful

Evaluation of Our Sentence Planner

Data

- Restaurant recommendations from the *BAGEL* generator
  - restaurant location, food type, etc.
  - just 404 utterances for 202 DAs

Results

- basic setup 54.24% BLEU, best version 59.89%
- less than *BAGEL*'s ~ 67% BLEU… but:
  - we do not use alignments
  - our generator decides itself what to include
- outputs mostly fluent and meaningful
- problems:
  - repeated/missing/irrelevant information on the output

Evaluation of Our Sentence Planner

Data

- Restaurant recommendations from the *BAGEL* generator
  - restaurant location, food type, etc.
  - just 404 utterances for 202 DAs

Results

- basic setup 54.24% BLEU, best version 59.89%
- less than *BAGEL*'s ~ 67% BLEU… but:
  - we do not use alignments
  - our generator decides itself what to include
- outputs mostly fluent and meaningful
- problems:
  - repeated/missing/irrelevant information on the output
  - slooooow, doesn't scale very well

Example Outputs

Input DA	inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant)
Reference	X restaurant is near X on the riverside.
Generated	X is a restaurant in the riverside area near X.

Example Outputs

Input DA	inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant)
Reference	X restaurant is near X on the riverside.
Generated	X is a restaurant in the riverside area near X.
Input DA	inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)
Reference	X is a moderately priced restaurant in X.
Generated	X is a restaurant in the X area.

Example Outputs

Input DA	inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant)
Reference	X restaurant is near X on the riverside.
Generated	X is a restaurant in the riverside area near X.
Input DA	inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)
Reference	X is a moderately priced restaurant in X.
Generated	X is a restaurant in the X area.
Input DA	inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French)
Reference	X is a French restaurant on the riverside.
Generated	X is a French restaurant in the riverside area which serves French food.

Example Outputs

Input DA	inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant)
Reference	X restaurant is near X on the riverside.
Generated	X is a restaurant in the riverside area near X.
Input DA	inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)
Reference	X is a moderately priced restaurant in X.
Generated	X is a restaurant in the X area.
Input DA	inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French)
Reference	X is a French restaurant on the riverside.
Generated	X is a French restaurant in the riverside area which serves French food.
Input DA	inform(name=X-name, type=placetoeat, eattype=restaurant, area=citycentre, near=X-near, food="Chinese takeaway", food=Japanese)
Reference	X is a Chinese takeaway and Japanese restaurant in the city centre near X.
Generated	X is a Japanese restaurant in the centre of town near X and X.

Surface Realization Examples

- Also various input formats, at least output is always text
- From handcrafted to different trainable realizers
- Also including our own (developed here at ÚFAL): *Treex/TectoMT* realizer
    - actually handcrafted for the most part

Grammar-based Realizers (90's): *KPML*, *FUF/SURGE*

KPML

- General purpose, multilingual
- Systemic Functional Grammar

```
(EXAMPLE
  :NAME   EX-SET-1
  :TARGETFORM   "It is raining cats and dogs."
  :LOGICALFORM
    (A / AMBIENT-PROCESS :LEX RAIN
        :TENSE PRESENT-CONTINUOUS :ACTEE
      (C / OBJECT :LEX CATS-AND-DOGS :NUMBER MASS))
)
```

Grammar-based Realizers (90's): *KPML*, *FUF/SURGE*

KPML

- General purpose, multilingual
- Systemic Functional Grammar

```
(EXAMPLE
  :NAME   EX-SET-1
  :TARGETFORM   "It is raining cats and dogs."
  :LOGICALFORM
    (A / AMBIENT-PROCESS :LEX RAIN
        :TENSE PRESENT-CONTINUOUS :ACTEE
      (C / OBJECT :LEX CATS-AND-DOGS :NUMBER MASS))
)
```

FUF/SURGE

- General purpose
- Functional Unification Grammar

Input Specification (I_1):

$$\begin{bmatrix} cat & clause \\ process & \begin{bmatrix} type & composite \\ relation & possessive \\ lex & \text{“hand”} \end{bmatrix} \\ partic & \begin{bmatrix} agent & \begin{bmatrix} cat & pers\_pro \\ gender & feminine \end{bmatrix} \\ affected & \boxed{1} \begin{bmatrix} cat & np \\ lex & \text{“editor”} \end{bmatrix} \\ possessor & \boxed{1} \\ possessed & \begin{bmatrix} cat & np \\ lex & \text{“draft”} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Output Sentence (S_1): "She hands the draft to the editor"

Grammar-based Realizer: *OpenCCG*

- General purpose, multi-lingual
- Combinatory Categorial Grammar
- Used in several projects
- With statistical enhancements

$$
\begin{array}{llll}
(>) & X/Y \;\; Y & \Rightarrow & X \\
(<) & Y \;\; X\backslash Y & \Rightarrow & X \\
(>\mathbf{B}) & X/Y \;\; Y/Z & \Rightarrow & X/Z \\
(<\mathbf{B}) & Y\backslash Z \;\; X\backslash Y & \Rightarrow & X\backslash Z \\
(>\mathbf{T}) & X & \Rightarrow & Y/(Y\backslash X) \\
(<\mathbf{T}) & X & \Rightarrow & Y\backslash(Y/X)
\end{array}
$$

man $\vdash$ n
that $\vdash$ $(n\backslash n)/(s_{vform=fin}/np)$
Bob $\vdash$ np
saw $\vdash$ $(s_{tense=past,vform=fin}\backslash np)/np$

$$
\dfrac{\dfrac{man}{n} \quad \dfrac{\dfrac{that}{(n\backslash n)/(s/np)} \quad \dfrac{\dfrac{\dfrac{Bob}{np}}{s/(s\backslash np)}{>T} \quad \dfrac{saw}{(s\backslash np)/np}}{s/np}{>B}}{n\backslash n}{>}}{n}{<}
$$

```
be [tense=pres info=rh id=n1]
<Arg> flight [num=sg det=the info=th id=f2]
      <HasProp> cheapest [kon=+ id=n2]
<Prop> has-rel [id=n3]
       <Of> f2
       <Airline> Ryanair [kon=+ id=n4]
```

$$
@_x(\mathbf{man} \wedge \langle \textsc{GenRel}\rangle(e \wedge \mathbf{see} \wedge \langle \textsc{tense}\rangle\mathbf{past} \\ \wedge \langle \textsc{Act}\rangle(b \wedge \mathbf{Bob}) \wedge \langle \textsc{Pat}\rangle x))
$$

Procedural Realizer: *SimpleNLG*

- General purpose
- English, adapted to several other languages
- Java implementation (procedural)

```
Lexicon lexicon = new XMLLexicon("my-lexicon.xml");
NLGFactory nlgFactory = new NLGFactory(lexicon);
Realiser realiser = new Realiser(lexicon);

SPhraseSpec p = nlgFactory.createClause();

p.setSubject("Mary");
p.setVerb("chase");
p.setObject("the monkey");

p.setFeature(Feature.TENSE, Tense.PAST);

String output = realiser.realiseSentence(p);
System.out.println(output);

>>> Mary chased the monkey.
```

Trainable Realizers: Overgenerate and Rank

- Require a handcrafted realizer, e.g. CCG realizer
- Input underspecified $\rightarrow$ more outputs possible
- Overgenerate
- Then use a statistical reranker

Trainable Realizers: Overgenerate and Rank

- Require a handcrafted realizer, e.g. CCG realizer
- Input underspecified $\rightarrow$ more outputs possible
- Overgenerate
- Then use a statistical reranker
- Ranking according to:
  - *n*-gram models (*NITROGEN, HALOGEN*)
  - Tree models (XTAG grammar – *FERGUS*)
  - Predicted Text-to-Speech quality (*Nakatsu and White*)
  - Personality traits (extraversion, agreeableness… – *CRAG*)
    + alignment (repeating words uttered by dialogue counterpart)

Trainable Realizers: Overgenerate and Rank

- Require a handcrafted realizer, e.g. CCG realizer
- Input underspecified $\rightarrow$ more outputs possible
- Overgenerate
- Then use a statistical reranker
- Ranking according to:
  - *n*-gram models (*NITROGEN, HALOGEN*)
  - Tree models (XTAG grammar – *FERGUS*)
  - Predicted Text-to-Speech quality (*Nakatsu and White*)
  - Personality traits (extraversion, agreeableness… – *CRAG*) + alignment (repeating words uttered by dialogue counterpart)
- Provides variance, but at a greater computational cost

Trainable Realizers: Syntax-Based

- *StuMaBa*: general realizer based on SVMs
- Pipeline:
  - ↓ Deep syntax/semantics
  - ↓ surface syntax
  - ↓ linearization
  - ↓ morphologization

Treex/TectoMT Surface Realizer

- Domain-independent

Treex/TectoMT Surface Realizer

- Domain-independent
- We use it for our experiments (*TGEN1*)
  - analysis $\rightarrow$ synthesis on *BAGEL* data = 89.79% BLEU

Treex/TectoMT Surface Realizer

- Domain-independent
- We use it for our experiments (*TGEN1*)
    - analysis $\rightarrow$ synthesis on *BAGEL* data = 89.79% BLEU
- Pipeline approach
- Mostly simple, single-purpose, rule-based modules (blocks)
    - Word inflection: statistical (*Flect*)

Treex/TectoMT Surface Realizer

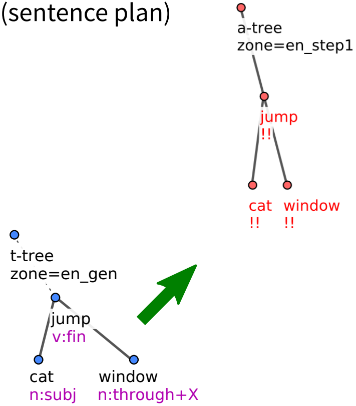
- Domain-independent
- We use it for our experiments (*TGEN1*)
  - analysis $\rightarrow$ synthesis on *BAGEL* data = 89.79% BLEU
- Pipeline approach
- Mostly simple, single-purpose, rule-based modules (blocks)
  - Word inflection: statistical (*Flect*)
- Gradual transformation of deep trees into surface dependency trees
  - Surface trees are then simply linearized

Treex/TectoMT Surface Realization Example

- Realizer steps (simplified):

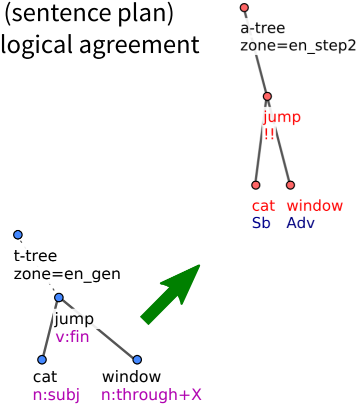
Treex/TectoMT Surface Realization Example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)



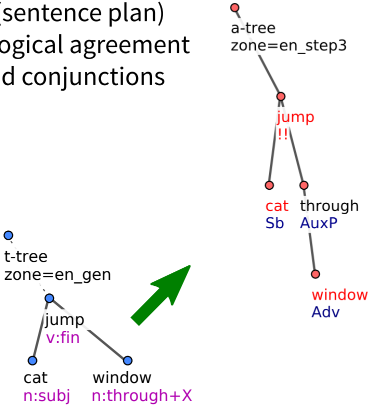
Treex/TectoMT Surface Realization Example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement



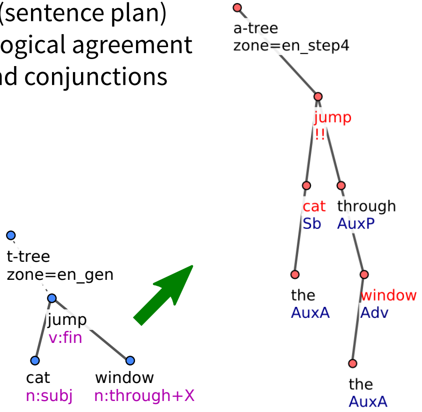
Treex/TectoMT Surface Realization Example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions



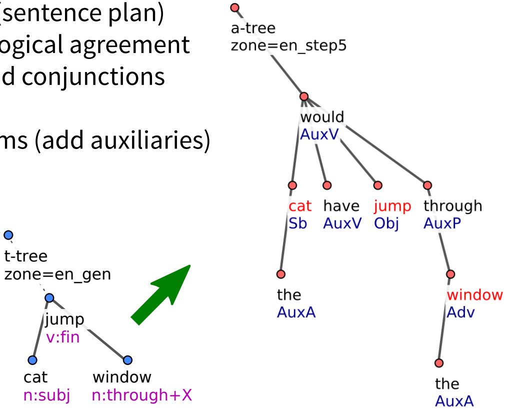
Treex/TectoMT Surface Realization Example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions
  - Add articles



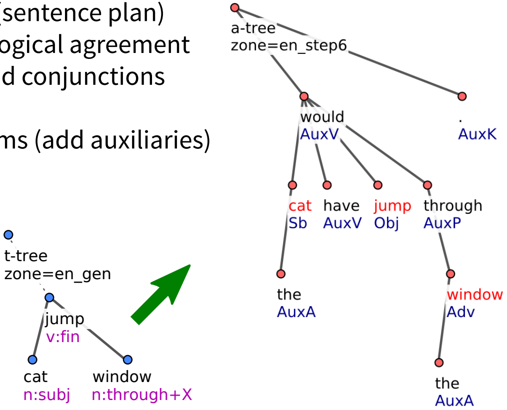
Treex/TectoMT Surface Realization Example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions
  - Add articles
  - Compound verb forms (add auxiliaries)



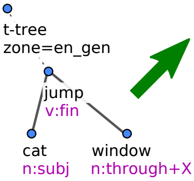
Treex/TectoMT Surface Realization Example

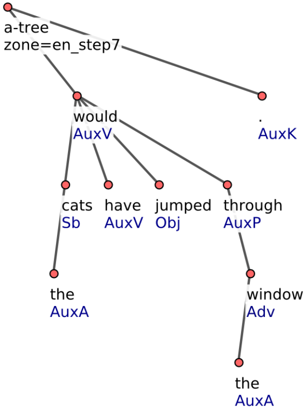
- Realizer steps (simplified):
    - Copy the deep tree (sentence plan)
    - Determine morphological agreement
    - Add prepositions and conjunctions
    - Add articles
    - Compound verb forms (add auxiliaries)
    - Punctuation



Treex/TectoMT Surface Realization Example

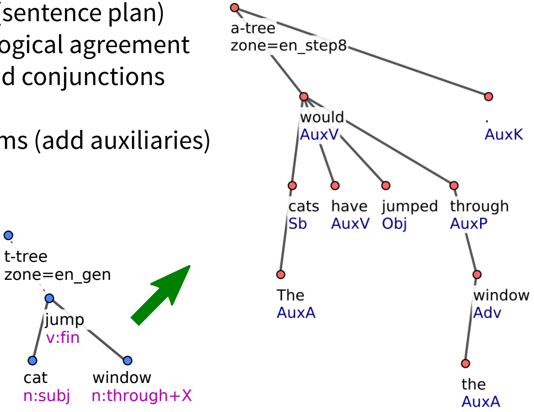
- Realizer steps (simplified):
    - Copy the deep tree (sentence plan)
    - Determine morphological agreement
    - Add prepositions and conjunctions
    - Add articles
    - Compound verb forms (add auxiliaries)
    - Punctuation
    - Word inflection





Treex/TectoMT Surface Realization Example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions
  - Add articles
  - Compound verb forms (add auxiliaries)
  - Punctuation
  - Word inflection
  - Capitalization



One-step NLG

One-step (Holistic) NLG

- Only one stage – no distinction
- Typical for limited domains, also in SDS
- Handcrafted/templates + statistical
  (lately also neural networks)

One-step NLG

One-step (Holistic) NLG

- Only one stage – no distinction
- Typical for limited domains, also in SDS
- Handcrafted/templates + statistical
  (lately also neural networks)

Template-based systems

- Most common, also in commercial NLG systems
- Simple, straightforward, reliable (custom-tailored for domain)
- Lack generality and variation, difficult to maintain
- Enhancements for more complex utterances: rules

Example: Templates

- Just filling variables into slots
- Possibly a few enhancements, e. g. articles

```
inform(pricerange="{pricerange}"):
'It is in the {pricerange} price range.'

affirm()&inform(task="find")
    &inform(pricerange="{pricerange}"):
'Ok, you are looking for something in the'
    + ' {pricerange} price range.'

affirm()&inform(area="{area}"):
'Ok, you want something in the {area} area.'

affirm()&inform(food="{food}")
    &inform(pricerange="{pricerange}"):
'Ok, you want something with the {food} food'
    + ' in the {pricerange} price range.'

inform(food="None"):
'I do not have any information'
    + ' about the type of food.'
```

Alex (English restaurant domain)

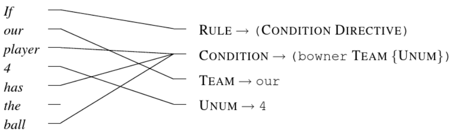
Facebook templates

One-step Statistical Non-neural NLG Approaches

- Limited domain
- Based on supervised learning
  (typically: MR + sentence + alignment)
- Typically: phrase-based

One-step Statistical Non-neural NLG Approaches

- Limited domain
- Based on supervised learning
  (typically: MR + sentence + alignment)
- Typically: phrase-based



Examples

- *BAGEL*: Bayesian networks
  - semantic stacks, ordering
- *Angeli et al.*: log-linear model
  - records ↘ fields ↘ templates
- *WASP*$^{-1}$: Synchronous CFGs
  - noisy channel, similar to MT

Neural NLG

- Using recurrent neural networks
- Input: 1-hot DA representation

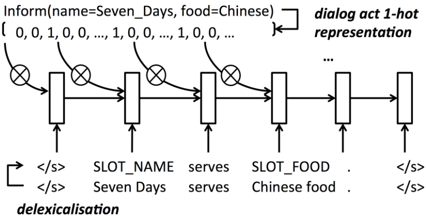
Neural NLG

- Using recurrent neural networks
- Input: 1-hot DA representation
- Generating word-by-word

Neural NLG

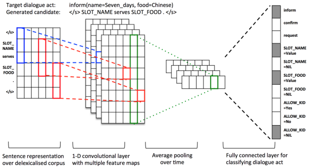
- Using recurrent neural networks
- Input: 1-hot DA representation
- Generating word-by-word
- Overgenerating + reranking/sanity checks

Neural NLG

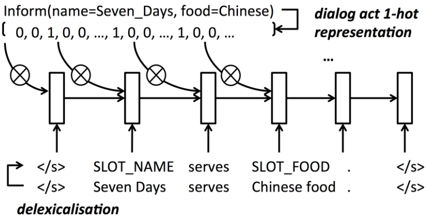


- Using recurrent neural networks
- Input: 1-hot DA representation
- Generating word-by-word
- Overgenerating + reranking/sanity checks

1. RNN language model + convolutional reranking (*RNNLM*)

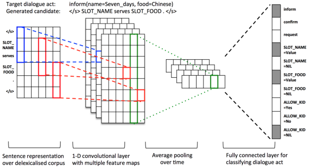


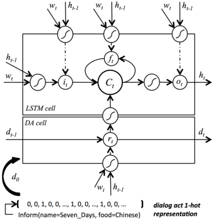
Neural NLG



- Using recurrent neural networks
- Input: 1-hot DA representation
- Generating word-by-word
- Overgenerating + reranking/sanity checks

1. RNN language model + convolutional reranking (*RNNLM*)
2. Long-short term memory (*SC-LSTM*)



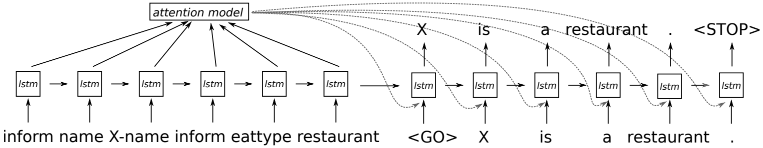


Sequence-to-Sequence Neural NLG *(TGEN2)*

- also based on RNN (LSTM)

Sequence-to-Sequence Neural NLG *(TGEN2)*

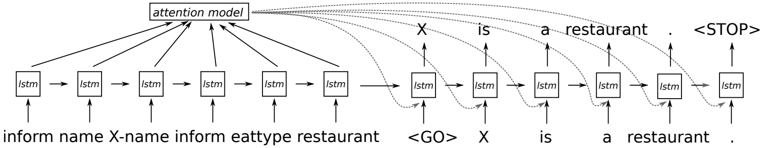
- also based on RNN (LSTM)
- sequence-to-sequence / encoder-decoder approach



Sequence-to-Sequence Neural NLG *(TGEN2)*

- also based on RNN (LSTM)
- sequence-to-sequence / encoder-decoder approach

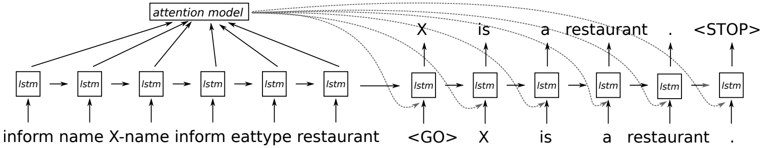
encode: DAs converted step-by-step by a RNN into a *hidden state*



Sequence-to-Sequence Neural NLG *(TGEN2)*

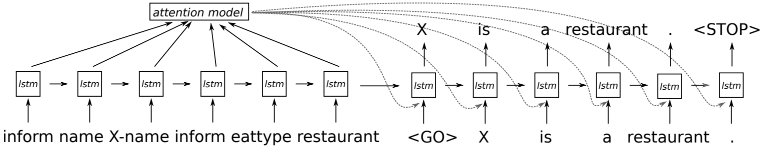
- also based on RNN (LSTM)
- sequence-to-sequence / encoder-decoder approach

encode: DAs converted step-by-step by a RNN into a *hidden state*
decode: output generated from the hidden state by a different RNN



Sequence-to-Sequence Neural NLG *(TGEN2)*

- also based on RNN (LSTM)
- sequence-to-sequence / encoder-decoder approach
  - encode: DAs converted step-by-step by a RNN into a *hidden state*
  - decode: output generated from the hidden state by a different RNN
- attention model – access to all encoder states
  - weighted by a simple feed-forward NN



Seq2Seq NLG variants

Trees/Strings

- one-step and two-step NLG with the same architecture

Seq2Seq NLG variants

Trees/Strings

- one-step and two-step NLG with the same architecture
- we can generate words or trees

Seq2Seq NLG variants

Trees/Strings

- one-step and two-step NLG with the same architecture
- we can generate words or trees
    - deep syntax trees same as *TGEN1*

Seq2Seq NLG variants

Trees/Strings

- one-step and two-step NLG with the same architecture
- we can generate words or trees
  - deep syntax trees same as *TGEN1*
  - using simple bracketed notation

(<root> <root> ((X-name n:subj) be v:fin ((Italian adj:attr) restaurant n:obj (river n:near+X))))

Seq2Seq NLG variants

Trees/Strings

- one-step and two-step NLG with the same architecture
- we can generate words or trees
    - deep syntax trees same as *TGEN1*
    - using simple bracketed notation
    - *Treex*/*TectoMT* realizer postprocessing

(<root> <root> ((X-name n:subj) be v:fin ((Italian adj:attr) restaurant n:obj (river n:near+X))))

Seq2Seq NLG variants

Trees/Strings

- one-step and two-step NLG with the same architecture
- we can generate words or trees
  - deep syntax trees same as *TGEN1*
  - using simple bracketed notation
  - *Treex/TectoMT* realizer postprocessing

(<root> <root> ((X-name n:subj) be v:fin ((Italian adj:attr) restaurant n:obj (river n:near+X))))

Controlling the output

- Using beam search + reranker

Seq2Seq NLG variants

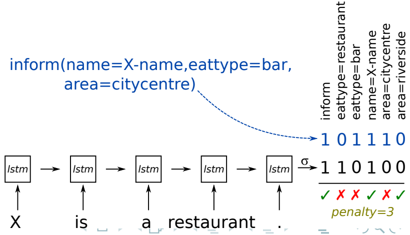
Trees/Strings

- one-step and two-step NLG with the same architecture
- we can generate words or trees
  - deep syntax trees same as *TGEN1*
  - using simple bracketed notation
  - *Treex/TectoMT* realizer postprocessing

(<root> <root> ((X-name n:subj) be v:fin ((Italian adj:attr) restaurant n:obj (river n:near+X))))

Controlling the output

- Using beam search + reranker
- Reranker: RNN + classifying
  0/1 presence of all DA slots/values



Seq2Seq NLG Evaluation

- on *BAGEL* data, same as *TGEN1*

Seq2Seq NLG Evaluation

- on *BAGEL* data, same as *TGEN1*
- measuring BLEU/NIST + semantic errors (manual, on a sample)

Seq2Seq NLG Evaluation

- on *BAGEL* data, same as *TGEN1*
- measuring BLEU/NIST + semantic errors (manual, on a sample)

Setup		**BLEU**	**NIST**	**ERR**
BAGEL (with alignments)		∼67	-	0
TGEN1		59.89	5.231	30
TGEN2	Greedy with trees	53.95	5.110	22
	+ Beam search	58.86	5.336	25
	+ Reranking classifier	59.72	5.491	19
	Greedy into strings	58.69	5.330	43
	+ Beam search	61.68	5.393	33
	+ Reranking classifier	59.82	5.515	8

Seq2Seq NLG Evaluation

- on *BAGEL* data, same as *TGEN1*
- measuring BLEU/NIST + semantic errors (manual, on a sample)

Setup		BLEU	NIST	ERR
BAGEL (with alignments)		∼67	-	0
TGEN1		59.89	5.231	30
TGEN2	Greedy with trees	53.95	5.110	22
	+ Beam search	58.86	5.336	25
	+ Reranking classifier	59.72	5.491	19
	Greedy into strings	58.69	5.330	43
	+ Beam search	61.68	5.393	33
	+ Reranking classifier	59.82	5.515	8

- *TGEN2* ~ same BLEU as *TGEN1*, but less semantic errors

Seq2Seq NLG Evaluation

- on *BAGEL* data, same as *TGEN1*
- measuring BLEU/NIST + semantic errors (manual, on a sample)

Setup		BLEU	NIST	ERR
BAGEL (with alignments)		∼67	-	0
TGEN1		59.89	5.231	30
TGEN2	Greedy with trees	53.95	5.110	22
	+ Beam search	58.86	5.336	25
	+ Reranking classifier	59.72	5.491	19
	Greedy into strings	58.69	5.330	43
	+ Beam search	61.68	5.393	33
	+ Reranking classifier	59.82	5.515	8

- *TGEN2* ~ same BLEU as *TGEN1*, but less semantic errors
- one-step (into strings) works better

New NLG developments

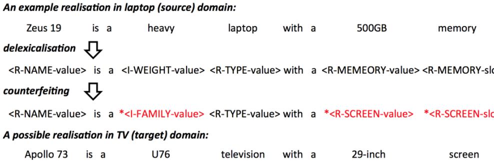
Multi-domain NLG *(DT-RNN)*

- Large out-of-domain data
  + very small in-domain data
- e.g., laptops $\rightarrow$ TVs

New NLG developments

Multi-domain NLG *(DT-RNN)*

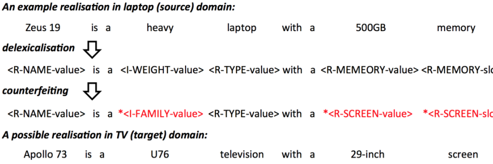
- Large out-of-domain data
  + very small in-domain data
- e.g., laptops → TVs
- “data counterfeiting”



New NLG developments

Multi-domain NLG *(DT-RNN)*

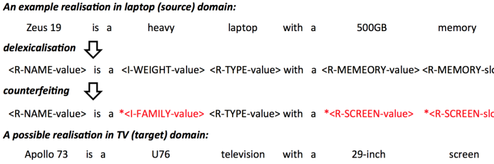
- Large out-of-domain data
  + very small in-domain data
- e.g., laptops → TVs
- “data counterfeiting”
- discriminative training



New NLG developments

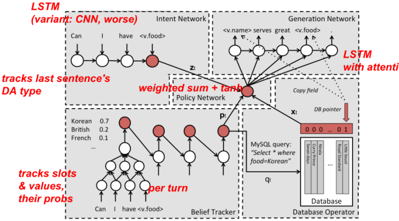
Multi-domain NLG *(DT-RNN)*

- Large out-of-domain data + very small in-domain data
- e.g., laptops → TVs
- “data counterfeiting”
- discriminative training



End-to-end *(E2E)* Spoken Dialogue Systems

- Based on a complex neural network
- Generation (word-by-word) is a part of the whole setup, trained jointly



Our Latest Experiments

- Using context in NLG

Our Latest Experiments

- Using context in NLG
- NLG system “entrains” (aligns) to the user
  - uses the same words, phrases

Our Latest Experiments

- Using context in NLG
- NLG system “entrains” (aligns) to the user
  - uses the same words, phrases

inform_no_match(alternative=next)

Sorry, I did not find a later option.

Our Latest Experiments

- Using context in NLG
- NLG system “entrains” (aligns) to the user
  - uses the same words, phrases

how bout the next ride
inform_no_match(alternative=next)
Sorry, I did not find a later option.

Our Latest Experiments

- Using context in NLG
- NLG system “entrains” (aligns) to the user
  - uses the same words, phrases

how bout the next ride

inform_no_match(alternative=next)

I'm sorry, the next ride was not found.

Our Latest Experiments

- Using context in NLG
- NLG system "entrains" (aligns) to the user
  - uses the same words, phrases

how bout the next ride
inform_no_match(alternative=next)
I'm sorry, the next ride was not found.

- First experiments
  - Adding context to the encoder with the DA

Our Latest Experiments

- Using context in NLG
- NLG system "entrains" (aligns) to the user
  - uses the same words, phrases

how bout the next ride
inform_no_match(alternative=next)
I'm sorry, the next ride was not found.

- First experiments
  - Adding context to the encoder with the DA
  - Two encoders & combining their hidden state

Our Latest Experiments

- Using context in NLG
- NLG system "entrains" (aligns) to the user
  - uses the same words, phrases

how bout the next ride
inform_no_match(alternative=next)
I'm sorry, the next ride was not found.

- First experiments
  - Adding context to the encoder with the DA
  - Two encoders & combining their hidden state
  - Reranker with BLEU against the context

Our Latest Experiments

- Using context in NLG
- NLG system "entrains" (aligns) to the user
  - uses the same words, phrases

how bout the next ride
inform_no_match(alternative=next)
I'm sorry, the next ride was not found.

- First experiments
  - Adding context to the encoder with the DA
  - Two encoders & combining their hidden state
  - Reranker with BLEU against the context
- Mild success (BLEU 66.7% $\rightarrow$ 69.5% at best)
  - wish me luck with manual rankings 😨😉

Thank you for your attention

Any comments, questions?
Corrections, protests?

Download these slides:

`http://bit.ly/nlg2016`

Contact me:

`odusek@ufal.mff.cuni.cz`
office 424

References

Angeli — Angeli, G. et al. 2010. A Simple Domain-Independent Probabilistic Approach to Generation. *EMNLP*

BAGEL — Mairesse, F. et al. 2010. Phrase-based statistical language generation using graphical models and active learning. *ACL*

CRAG — Isard, A. et al. 2006. Individuality and alignment in generated dialogues. *INLG*

DT-RNN — Wen, T. H. et al. 2016. Multi-domain neural network language generation for spoken dialogue systems. *NAACL* (to appear)

E2E — Wen, T. H. et al. 2016. A network-based end-to-end trainable task-oriented dialogue system. *arXiv*

FERGUS — Bangalore, S. and Rambow, O. 2000. Exploiting a probabilistic hierarchical model for generation. *COLING*

Flect — Dušek, O. and Jurčíček, F. 2013. Robust Multilingual Statistical Morphological Generation Models. *ACL-SRW*

FUF/SURGE — Elhadad, M. and Robin, J. 1996. An overview of SURGE: A reusable comprehensive syntactic realization component.
`http://www.cs.bgu.ac.il/surge/`

HALOGEN — Langkilde-Geary, I. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. *INLG*

KPML — Bateman, J. A. 1997. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*
`http://purl.org/net/kpml`

OpenCCG — White, M. and Baldrige, J. 2003. Adapting Chart Realization to CCG. *ENLG*
Moore, J. et al. 2004. Generating Tailored, Comparative Descriptions in Spoken Dialogue. *FLAIRS*
`http://openccg.sourceforge.net/`

Nakatsu&White — Nakatsu, C. and White, M. 2006. Learning to say it well: reranking realizations by predicted synthesis quality. *COLING-ACL*

NITROGEN — Langkilde, I. and Knight, K. 1998. Generation that exploits corpus-based statistical knowledge. *ACL-COLING*

Paiva&Evans — Paiva, D. S. and Evans, R. 2005. Empirically-based control of natural language generation. *ACL*

PERSONAGE-PE — Mairesse, F. and Walker, M. 2008. Trainable generation of big-five personality styles through data-driven parameter estimation. *ACL*

References

RNNLM Wen, T. H. et al. 2015. Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. *SIGDIAL*

SC-LSTM Wen, T. H. et al. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. *EMNLP*

SimpleNLG Gatt, A. and Reiter, E. 2009. SimpleNLG: A realisation engine for practical applications. *ENLG*

SPoT Walker, M. et al. 2001. SPoT: A trainable sentence planner. *NAACL*

StuMaBa Bohnet, B. et al. 2010. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. *COLING*

TectoMT Dušek, O. et al. 2015. New language pairs in TectoMT. *WMT*

TGEN1 Dušek, O. and Jurčíček, F. 2015. Training a natural language generator from unaligned data. *ACL-IJCNLP*

TGEN2 Dušek, O. and Jurčíček, F. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. *ACL* (to appear)

Textbook Reiter, E. and Dale, R. 2000. *Building natural language generation systems*. Cambridge Univ. Press

Treex Popel, M. and Žabokrtský, Z. 2010. TectoMT: modular NLP framework. *IceTAL* http://ufal.cz/treex

WASP^{-1} Wong, Y. W. and Mooney, R. 2007. Generation by inverting a semantic parser that uses statistical machine translation. *NAACL-HLT*

Further Links

C. DiMarco's slides: https://cs.uwaterloo.ca/~jchampai/CohenClass.en.pdf
F. Mairesse's slides: http://people.csail.mit.edu/francois/research/papers/ART-NLG.pdf
J. Moore's NLG course: http://www.inf.ed.ac.uk/teaching/courses/nlg/
NLG Systems Wiki: http://www.nlg-wiki.org
Wikipedia: http://en.wikipedia.org/wiki/Natural_language_generation