Training a Natural Language Generator from Unaligned Data

Ondřej Dušek and Filip Jurčíček

Institute of Formal and Applied Linguistics
Charles University in Prague

July 27, 2015
Introduction

• NLG = meaning representation $\rightarrow$ sentence
  • (for use in dialogues)
Introduction

- NLG = meaning representation → sentence
  - (for use in dialogues)
- Typical NLG system training:
  a) requires alignments of MR elements and words/phrases
  b) uses a separate alignment step

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

```
X is an italian restaurant in the riverside area .
```

```
text
```

```
alignment
```

```
MR
```

```
be
```

```
be
```

```
italian
```

```
restaurant
```

```
riverside
```

```
area
```

```
X
```

```
n:in+X
```

```
automatic analysis in Treex
```

```
sentence plan
deep syntax tree
```

```
2/17 Ondřej Dušek and Filip Jurčíček
```

Training a Natural Language Generator from Unaligned Data
Introduction

• NLG = meaning representation $\rightarrow$ sentence
  • (for use in dialogues)
• Typical NLG system training:
  a) requires alignments of MR elements and words/phrases
  b) uses a separate alignment step
• Our generator 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
• Our generator learns alignments jointly
  • training from pairs: **MR + sentence**
  • with sentence planning (MR $\rightarrow$ deep syntax trees)

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

**text**

X is an Italian restaurant in the riverside area.
Why learn alignments jointly?

- No need for manual annotation
  - faster/cheaper for larger domains
Why learn alignments jointly?

- No need for manual annotation
  - faster/cheaper for larger domains
- Avoiding errors of automatic preprocessing
  - errors may add up
Why learn alignments jointly?

- No need for manual annotation
  - faster/cheaper for larger domains
- Avoiding errors of automatic preprocessing
  - errors may add up
- No hard alignments forced on the generator, alignment is latent
  - MR elements ↔ words/phrases may not always be 1 : 1
Introduction

Motivation

Why learn alignments jointly?

- No need for manual annotation
  - faster/cheaper for larger domains
- Avoiding errors of automatic preprocessing
  - errors may add up
- No hard alignments forced on the generator, alignment is latent
  - MR elements $\leftrightarrow$ words/phrases may not always be 1 : 1

```
inform(name=X-name, type=placetoeat, area=centre, eattype=restaurant, near=X-near)
The X restaurant is conveniently located near X, right in the city center.
```

```
inform(name=X-name, type=placetoeat, foodtype=Chinese_takeaway)
X serves Chinese food and has a takeaway possibility.
```

```
inform(name=X-name, type=placetoeat, pricerange=cheap)
Prices at X are quite cheap.
```
Overall workflow of our generator

A two-step setup:
Overall workflow of our generator

A two-step setup:

- *Input*: a meaning representation
Overall workflow of our generator

A two-step setup:

- **Input**: a meaning representation
- **1. sentence planning**
  - statistical, our main focus
  - expanding + ranking candidate sentence plans
  - $A^*$-like search
Overall workflow of our generator

A two-step setup:

- **Input**: a meaning representation
- **1. sentence planning**
  - statistical, our main focus
  - expanding + ranking candidate sentence plans
  - A*-like search
- **Intermediate**: sentence plan (deep syntax trees)

MR

Sentence planner

![Diagram of sentence planner]

- **Sentence planner**
- **candidate generator**
- **scorer**
- **A* search**
- **sentence plan (deep syntax tree)**
Overall workflow of our generator

A two-step setup:

- **Input**: a meaning representation
- **1. – sentence planning**
  - statistical, our main focus
  - expanding + ranking candidate sentence plans
  - A*-like search
- **Intermediate**: sentence plan (deep syntax trees)
- **2. – surface realization**
  - reusing Treex/TectoMT realizer
  - (mostly) rule-based pipeline
Overall workflow of our generator

A two-step setup:

- **Input**: a meaning representation

1. **sentence planning**
   - statistical, our main focus
   - expanding + ranking candidate sentence plans
   - A*-like search

- **Intermediate**: sentence plan (deep syntax trees)

2. **surface realization**
   - reusing *Treex/TectoMT* realizer
   - (mostly) rule-based pipeline

- **Output**: plain text sentence
Data formats

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

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

```
X is an Italian restaurant in the riverside area.
```
Data formats

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

• **Sentence plan**: deep-syntax dependency trees
  - nodes for content words only (nouns, verbs, adjectives, adverbs)
  - two attributes per tree node: *t-lemma* + *formeme*
  - using surface word order

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

```
t-tree
X-name
n:subj
be
v:fin
italian
adj:attr
restaurant
n:obj
riverside
n:attr
area
n:in+X
```

```
XisanItalianrestaurantintheriversidearea.
```
Our generator

Introduction

Data formats

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

- **Sentence plan**: deep-syntax dependency trees
  - nodes for content words only (nouns, verbs, adjectives, adverbs)
  - two attributes per tree node: *t-lemma* + *formeme*
  - using surface word order

- **Output**: plain text sentence

```
X is an Italian restaurant in the riverside area.
```

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

```

X-name
n:subj
be
v:fin
italian
adj:attr
restaurant
n:obj
riverside
n:attr
area
n:in+X
```
Sentence planner

- A*-style search
  - “finding the path” from empty tree to full sentence plan tree
  - expand the most promising candidate sentence plan in each step
  - stop when candidates don't improve for a while
Sentence planner

- A*-style search
  - “finding the path” from empty tree to full sentence plan tree
  - expand the most promising candidate sentence plan in each step
  - 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)
Sentence planner

- A*-style search
  - “finding the path” from empty tree to full sentence plan tree
  - expand the most promising candidate sentence plan in each step
  - 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
Sentence planner

- **A*-style search**
  - “finding the path” from empty tree to full sentence plan tree
  - expand the most promising candidate sentence plan in each step
  - 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

- Training data = MR + sentence plan tree pairs
  - trees obtained by automatic parsing in *Treex*
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)

• Combinations explode even for small trees

• Limiting “possible places”
  • a few simple rules
  • based on context (elements of current MR, parent node)
Scorer/Ranker

- a function:

  \[
  \text{sentence plan tree} + \text{MR} \rightarrow \text{real-valued score}
  \]

- describes the fitness of tree for MR
Scorer/Ranker

- a function:
  \[ \text{sentence plan tree} + \text{MR} \rightarrow \text{real-valued score} \]
  - describes the fitness of tree for MR

Linear perceptron scorer (Collins & Duffy, 2002)

- **score** = weights \cdot features (from tree and MR)
  - features – elements of tree and MR
  - presence of nodes, slots, values + combination
  - tree size and shape, parent-child
Scorer/Ranker

- a function:
  
  \[
  \text{sentence plan tree + MR} \rightarrow \text{real-valued score}
  \]

  - describes the fitness of tree for MR

Linear perceptron scorer (Collins & Duffy, 2002)

- \textbf{score} = \text{weights} \cdot \text{features (from tree and MR)}
  
  - features – elements of tree and MR
  - presence of nodes, slots, values + combination
  - tree size and shape, parent-child

- \textbf{training} loop:
  
  - given MR, generate the best tree with current weights
  - update weights if generated tree ranks better than gold tree
Scorer/Ranker

- a function:
  \[ \text{sentence plan tree} + \text{MR} \rightarrow \text{real-valued score} \]
  - describes the fitness of tree for MR

Linear perceptron scorer (Collins & Duffy, 2002)

- **score** = weights \cdot features (from tree and MR)
  - features – elements of tree and MR
  - presence of nodes, slots, values + combination
  - tree size and shape, parent-child

- **training** loop:
  - given MR, generate the best tree with current weights
  - update weights if generated tree ranks better than gold tree

- **update** = \( \alpha \cdot \) difference in features (gold—generated)
  - want gold to score better next time
Scoring problem

- Features are global over the whole sentence plan tree → bigger trees tend to score better
Scoring problem

- Features are global over the whole sentence plan tree → bigger trees tend to score better
- But we score incomplete trees during the A* search
  - bigger incomplete trees are not always right
  - we need to promote “promising” incomplete trees
Scoring problem

- Features are global over the whole sentence plan tree → bigger trees tend to score better
- But we score incomplete trees during the A* search
  - bigger incomplete trees are not always right
  - we need to promote “promising” incomplete trees
- Scoring accuracy affects which paths are explored
Scoring problem

- Features are global over the whole sentence plan tree → bigger trees tend to score better
- But we score incomplete trees during the A* search
  - bigger incomplete trees are not always right
  - we need to promote “promising” incomplete trees
- Scoring accuracy affects which paths are explored

Our improvements to the scorer

- Differing tree updates
- Future promise
Differing subtree updates

- Additional perceptron update
  - performed with the regular one
  - using pairs of differing subtrees of gold and generated tree (starting from common subtree)
  - promoting promising paths, demoting dead-ends
Differing subtree updates

- Additional perceptron update
  - performed with the regular one
  - using pairs of differing subtrees of gold and generated tree (starting from common subtree)
  - promoting promising paths, demoting dead-ends
Differing subtree updates

- Additional perceptron update
  - performed with the regular one
  - using pairs of differing subtrees of gold and generated tree (starting from common subtree)
  - promoting promising paths, demoting dead-ends
Differing subtree updates

- Additional perceptron update
  - performed with the regular one
  - using pairs of differing subtrees of gold and generated tree (starting from common subtree)
  - promoting promising paths, demoting dead-ends
Differing subtree updates

- Additional perceptron update
  - performed with the regular one
  - using pairs of differing subtrees of gold and generated tree (starting from common subtree)
  - promoting promising paths, demoting dead-ends

- Common subtree
Differing subtree updates

• Additional perceptron update
  • performed with the regular one
  • using pairs of differing subtrees of gold and generated tree (starting from common subtree)
  • promoting promising paths, demoting dead-ends
Differing subtree updates

- Additional perceptron update
  - performed with the regular one
  - using pairs of differing subtrees of gold and generated tree (starting from common subtree)
  - promoting promising paths, demoting dead-ends
Future promise estimate

• Further score boost for incomplete trees
Future promise estimate

- Further score boost for incomplete trees
- Using the *expected number of children* of a node

```
<table>
<thead>
<tr>
<th>n:subj</th>
<th>be</th>
<th>v:fin</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-name</td>
<td>restaurant</td>
<td>n:obj</td>
</tr>
<tr>
<td>italian</td>
<td>adj:attr</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>n:subj</th>
<th>be</th>
<th>v:fin</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-name</td>
<td>restaurant</td>
<td>n:obj</td>
</tr>
<tr>
<td>price</td>
<td>v:attr</td>
<td></td>
</tr>
<tr>
<td>(priced)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(moderately, cheaply...)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Future promise estimate

• Further score boost for incomplete trees
• Using the *expected number of children* of a node

**Future promise:**
“how many children are missing to meet the expectation”
  • floored at zero, summed over the whole tree
• Added to scores, used to select next expansion path
Experimental Setup

Data

- Restaurant recommendations from the *BAGEL* generator (Mairesse et al., 2010)
  - restaurant location, food type, etc.
  - 404 sentences for 202 input dialogue acts, 2 paraphrases each
  - manual alignment provided, but we don't use it
Experiments

Experimental Setup

Data

- Restaurant recommendations from the BAGEL generator (Mairesse et al., 2010)
  - restaurant location, food type, etc.
  - 404 sentences for 202 input dialogue acts, 2 paraphrases each
  - manual alignment provided, but we don't use it

Setup

- using 10-fold cross-validation
- measuring BLEU/NIST against 2 references
Experimental Setup

Data

- Restaurant recommendations from the BAGEL generator (Mairesse et al., 2010)
  - restaurant location, food type, etc.
  - 404 sentences for 202 input dialogue acts, 2 paraphrases each
  - manual alignment provided, but we don't use it

Setup

- using 10-fold cross-validation
- measuring BLEU/NIST against 2 references
## Results

<table>
<thead>
<tr>
<th>Setup</th>
<th>BLEU</th>
<th>NIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>perceptron scorer</td>
<td>54.24</td>
<td>4.643</td>
</tr>
<tr>
<td>+ differing subtree updates</td>
<td>58.70*</td>
<td>4.876</td>
</tr>
<tr>
<td>+ future promise</td>
<td>59.89*</td>
<td>5.231</td>
</tr>
</tbody>
</table>

* both improvements statistically significant
<table>
<thead>
<tr>
<th>Setup</th>
<th>BLEU</th>
<th>NIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>perceptron scorer</td>
<td>54.24</td>
<td>4.643</td>
</tr>
<tr>
<td>+ differing subtree updates</td>
<td>58.70*</td>
<td>4.876</td>
</tr>
<tr>
<td>+ future promise</td>
<td>59.89*</td>
<td>5.231</td>
</tr>
</tbody>
</table>

- * both improvements statistically significant
- Overall, lower scores than Mairesse et al.'s ~ 67% BLEU
Results

<table>
<thead>
<tr>
<th>Setup</th>
<th>BLEU</th>
<th>NIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>perceptron scorer</td>
<td>54.24</td>
<td>4.643</td>
</tr>
<tr>
<td>+ differing subtree updates</td>
<td>58.70*</td>
<td>4.876</td>
</tr>
<tr>
<td>+ future promise</td>
<td>59.89*</td>
<td>5.231</td>
</tr>
</tbody>
</table>

* both improvements statistically significant

Overall, lower scores than Mairesse et al.'s ~ 67% BLEU

But our problem is harder:
  - we learn alignments jointly
  - our generator has to decide when to stop
    (whether all required information is included)
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th><code>inform(name=X-name, type=placetoeat, pricerange=moderate, eattype=restaurant)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a restaurant that offers moderate price range.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a restaurant in the moderate price range.</td>
</tr>
</tbody>
</table>

- Mostly fluent and relevant
- Sometimes identical to reference, more often original
- Problems in some cases:
  - Information missing/repeated/superfluous

---

Ondřej Dušek and Filip Jurčíček

Training a Natural Language Generator from Unaligned Data
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>inform(name=X-name, type=placetoeat, pricerange=moderate, eattype=restaurant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a restaurant that offers moderate price range.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a restaurant in the moderate price range.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input DA</th>
<th>inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a moderately priced restaurant in X.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a restaurant in the X area.</td>
</tr>
</tbody>
</table>
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>inform(name=X-name, type=placetoeat, pricerange=moderate, eattype=restaurant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a restaurant that offers moderate price range.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a restaurant in the moderate price range.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input DA</th>
<th>inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a moderately priced restaurant in X.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a restaurant in the X area.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input DA</th>
<th>inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a French restaurant on the riverside.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a French restaurant in the riverside area which serves French food.</td>
</tr>
</tbody>
</table>
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>Reference</th>
<th>Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>inform(name=X-name, type=placetoeat, pricerange=moderate, eattype=restaurant)</td>
<td>X is a restaurant that offers moderate price range.</td>
<td>X is a restaurant in the moderate price range.</td>
</tr>
<tr>
<td>inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)</td>
<td>X is a moderately priced restaurant in X.</td>
<td>X is a restaurant in the X area.</td>
</tr>
<tr>
<td>inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French)</td>
<td>X is a French restaurant on the riverside.</td>
<td>X is a French restaurant in the riverside area which serves French food.</td>
</tr>
</tbody>
</table>

- Mostly fluent and relevant
  - sometimes identical to reference, more often original
### Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>Reference</th>
<th>Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>inform(name=X-name, type=placetoeat, pricerange=moderate, eattype=restaurant)</td>
<td>X is a restaurant that offers moderate price range.</td>
<td>X is a restaurant in the moderate price range.</td>
</tr>
<tr>
<td>inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)</td>
<td>X is a moderately priced restaurant in X.</td>
<td>X is a restaurant in the X area.</td>
</tr>
<tr>
<td>inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French)</td>
<td>X is a French restaurant on the riverside.</td>
<td>X is a French restaurant in the riverside area which serves French food.</td>
</tr>
</tbody>
</table>

- Mostly fluent and relevant
  - sometimes identical to reference, more often original
- Problems in some cases:
  - information missing / repeated / superfluous
Our NLG system – summary

- learns from unaligned MR–sentence pairs
Our NLG system – summary

- learns from unaligned MR-sentence pairs
- two-step (sentence planning, surface realization)
- deep syntax trees for sentence plans
Our NLG system – summary

- learns from unaligned MR-sentence pairs
- two-step (sentence planning, surface realization)
- deep syntax trees for sentence plans
- A*-style search, expand & score sentence plans
- perceptron scoring + improvements
Conclusion

Our NLG system – summary

• learns from unaligned MR–sentence pairs
• two-step (sentence planning, surface realization)
• deep syntax trees for sentence plans
• A*-style search, expand & score sentence plans
• perceptron scoring + improvements

Conclusion

• Learning sentence planning from unaligned data is feasible
• Promising results, but lower than previous with manual alignment (Mairesse et al.)
Future work

- Refine feature set
- Replace it with a neural network
- Try 1-step with surface dependency trees
Conclusion

Future work

• Refine feature set
• Replace it with a neural network
• Try 1-step with surface dependency trees
• Other suggestions?
Future work

- Refine feature set
- Replace it with a neural network
- Try 1-step with surface dependency trees
- Other suggestions?

Thank you for your attention

Contact us
Ondřej Dušek & Filip Jurčíček
Charles University in Prague
odusek@ufal.mff.cuni.cz

See the paper
More details there

Check out our code
https://github.com/UFAL-DSG/tgen

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