Decoding and Inference with Syntactic Translation Models

Machine Translation Lecture 15

Instructor: Chris Callison-Burch
TAs: Mitchell Stern, Justin Chiu

Website: mt-class.org/penn
CFGs

S → NP VP
VP → NP V
V → tabeta
NP → jon-ga
NP → ringo-o

Output: jon-ga ringo-o tabeta
Synchronous CFGs

S → NP VP
VP → NP V
V → tabeta
NP → jon-ga
NP → ringo-o
Synchronous CFGs

\[
S \rightarrow \text{NP VP} \quad \begin{array}{c}
1 \\
2
\end{array} \\
\text{(monotonic)}
\]

\[
\text{VP} \rightarrow \text{NP V} \quad \begin{array}{c}
2 \\
1
\end{array} \\
\text{(inverted)}
\]

\[
V \rightarrow \text{tabeta} \quad \text{ate}
\]

\[
\text{NP} \rightarrow \text{jon-ga} \quad \text{John}
\]

\[
\text{NP} \rightarrow \text{ringo-o} \quad \text{an apple}
\]
Output: (jon-ga ringo-o tabeta : *John ate an apple*)
Translation as parsing

Parse source

Project to target

S
/   \
/    /
NP    VP
/  \
/   /
jon-ga ringo-o tabeta

S
/   \
/    /
NP    VP
/  \
/   /
John   ate

NP
/   \
/    
an apple

S
/   \
/    /
NP    VP
/  \
/   /
John    ate

NP
/   \
/    
an apple
A closer look at parsing

- Parsing is usually done with dynamic programming
- Share common computations and structure
- Represent exponential number of alternatives in polynomial space
- With SCFGs there are two kinds of ambiguity
  - source parse ambiguity
  - translation ambiguity
- parse forests can represent both!
A closer look at parsing

• Any monolingual parser can be used (most often: CKY or variants on the CKY algorithm)

• Parsing complexity is $O(|n^3|)$
  • cubic in the length of the sentence ($n^3$)
  • cubic in the number of non-terminals ($|G|^3$)
    • adding nonterminal types increases parsing complexity substantially!

• With few NTs, exhaustive parsing is tractable
Parsing as deduction

“If $A$ and $B$ are true with weights $u$ and $v$, and phi is also true, then $C$ is true with weight $w$.”
Example: CKY

Inputs:
\[ f = \langle f_1, f_2, \ldots, f_L \rangle \]

\[ G \] Context-free grammar in Chomsky normal form.

Item form:
\[ [X, i, j] \] A subtree rooted with NT type \( X \) spanning \( i \) to \( j \) has been recognized.
Example: CKY

Goal:

\[ [S, 0, \ell] \]

Axioms:

\[
[X, i - 1, i] : w \quad (X \overset{w}{\rightarrow} f_i) \in G
\]

Inference rules:

\[
[X, i, k] : u \quad [Y, k, j] : v
\]

\[
[Z, i, j] : u \times v \times w \quad (Z \overset{w}{\rightarrow} XY) \in G
\]
S → PRP VP
VP → V NP
VP → V SBAR
SBAR → PRP V
NP → PRP NN
V → saw
NN → duck
V → duck
PRP → I
PRP → her

I saw her duck
I saw her duck.
I saw her duck.
I saw her duck.
I saw her duck.
I saw her duck.
S → PRP VP
VP → V NP
VP → V SBAR
SBAR → PRP V
NP → PRP NN
V → saw
NN → duck
V → duck
PRP → I
PRP → her

I saw her duck
I saw her duck.
I saw her duck.
S → PRP VP

VP → V NP

VP → V SBAR

SBAR → PRP V

NP → PRP NN

V → saw

NN → duck

V → duck

PRP → I

PRP → her

NP → PRP NN
I saw her duck.
I saw her duck.
Semantics of hypergraphs

- Generalization of directed graphs
- Special node designated the “goal”
- Every edge has a single head and 0 or more tails (the arity of the edge is the number of tails)
- Node labels correspond to LHS’s of CFG rules
- A derivation is the generalization of the graph concept of path to hypergraphs
- Weights multiply along edges in the derivation, and add at nodes (cf. semiring parsing)
Edge labels

- Edge labels may be a mix of terminals and substitution sites (non-terminals)
- In translation hypergraphs, edges are labeled in both the source and target languages
- The number of substitution sites must be equal to the arity of the edge and must be the same in both languages
- The two languages may have different orders of the substitution sites
- There is no restriction on the number of terminal symbols
Edge labels

\{ \text{la lectura de ayer : yesterday's reading}, \text{la lectura de ayer : reading from yesterday} \}
Inference algorithms

- Viterbi \( O(|E| + |V|) \)
  - Find the maximum weighted derivation
  - Requires a partial ordering of weights
- Inside - outside \( O(|E| + |V|) \)
  - Compute the marginal (sum) weight of all derivations passing through each edge/node
- k-best derivations \( O(|E| + |D_{max}|k \log k) \)
  - Enumerate the k-best derivations in the hypergraph
- See IWPT paper by Huang and Chiang (2005)
Things to keep in mind

Bound on the number of edges:
\[|E| \in O(n^3|G|^3)\]

Bound on the number of nodes:
\[|V| \in O(n^2|G|)\]
Decoding Again

- Translation hypergraphs are a “lingua franca” for translation search spaces
- Note that FST lattices are a special case
- Decoding problem: how do I build a translation hypergraph?
Representational limits

Consider this very simple SCFG translation model:

“Glue” rules:

\[ S \rightarrow S \ S \ S : \ 1 \ 2 \]
\[ S \rightarrow S \ S \ S : \ 2 \ 1 \]
Representational limits

Consider this very simple SCFG translation model:

“Glue” rules:
\[
\begin{align*}
S & \rightarrow \ S \ S \ S : 1 \ 2 \\
S & \rightarrow \ S \ S \ S : 2 \ 1
\end{align*}
\]

“Lexical” rules:
\[
\begin{align*}
S & \rightarrow \ \text{tabeta} : \text{ate} \\
S & \rightarrow \ \text{jon-ga} : \text{John} \\
S & \rightarrow \ \text{ringo-o} : \text{an apple}
\end{align*}
\]
Representational limits

• Phrase-based decoding runs in exponential time

• All permutations of the source are modeled (traveling salesman problem!)

• Typically distortion limits are used to mitigate this

• But parsing is polynomial...what’s going on?
Binary SCFGs cannot model this (however, ternary SCFGs can):
Representational limits

Binary SCFGs cannot model this (however, ternary SCFGs can):  

![Diagram](image)

But can’t we binarize *any* grammar?
Representational limits

Binary SCFGs cannot model this (however, ternary SCFGs can):

But can’t we binarize any grammar?

No. Synchronous CFGs cannot generally be binarized!
Does this matter?

• The “forbidden” pattern is observed in real data (Melamed, 2003)

• Does this matter?

  • Learning

    • Phrasal units and higher rank grammars can account for the pattern

    • Sentences can be simplified or ignored

• Translation

  • The pattern does exist, but how often must it exist (i.e., is there a good translation that doesn’t violate the SCFG matching property)?
Tree-to-string

- How do we generate a hypergraph for a tree-to-string translation model?
  - Simple linear-time (given a fixed translation model) top-down matching algorithm
  - Recursively cover “uncovered” sites in tree
  - Each node in the input tree becomes a node in the translation forest
  - For details, Huang et al. (AMTA, 2006) and Huang et al. (EMNLP, 2010)
S($x_1$:NP $x_2$:VP) → $x_1$ $x_2$
VP($x_1$:NP $x_2$:V) → $x_2$ $x_1$

\{ 
  \text{tabeta} \rightarrow \text{ate} \\
  \text{ringo-o} \rightarrow \text{an apple} \\
  \text{jon-ga} \rightarrow \text{John} 
\} 

Tree-to-string grammar
S(x₁:NP  x₂:VP) → x₁  x₂
VP(x₁:NP  x₂:V) → x₂  x₁
  tabeta → ate
  ringo-o → an apple
  jon-ga → John
John ate an apple.

[S(x₁:NP x₂:VP) → x₁ x₂]

[VP(x₁:NP x₂:V) → x₂ x₁]

\textit{tabeta} → \textit{ate}

\textit{ringo-o} → \textit{an apple}

\textit{jon-ga} → \textit{John}
S($x_1$:NP $x_2$:VP) → $x_1$ $x_2$

VP($x_1$:NP $x_2$:V) → $x_2$ $x_1$

tabeta → ate

ringo-o → an apple

jon-ga → John
S(x₁:NP x₂:VP) → x₁ x₂
VP(x₁:NP x₂:V) → x₂ x₁

\text{tabeta} → \text{ate}
\text{ringo-o} → \text{an apple}
\text{jon-ga} → \text{John}
\[
S(x_1:NP \ x_2:VP) \rightarrow x_1 \ x_2
\]

\[
VP(x_1:NP \ x_2:V) \rightarrow x_2 \ x_1
\]

\[
\text{tabeta} \rightarrow \text{ate}
\]

\[
\text{ringo-o} \rightarrow \text{an apple}
\]

\[
\text{jon-ga} \rightarrow \text{John}
\]
\[ S(x_1:NP \ x_2:VP) \rightarrow x_1 \ x_2 \]
\[ VP(x_1:NP \ x_2:V) \rightarrow x_2 \ x_1 \]
\[ \text{tabeta} \rightarrow \text{ate} \]
\[ \text{ringo-o} \rightarrow \text{an apple} \]
\[ \text{jon-ga} \rightarrow \text{John} \]
Language Models
Hypergraph review

Source label

Target label

Goal node
Hypergraph review

Substitution sites / variables / non-terminals
Hypergraph review

For LM integration, we ignore the source!
Hypergraph review

For LM integration, we ignore the source!
Hypergraph review

How can we add the LM score to each string derived by the hypergraph?
LM Integration

- If LM features were purely local ...
  - “Unigram” model
- ... integration would be a breeze
  - Add an “LM feature” to every edge
- But, LM features are non-local!
Why is it hard?

Two problems:

1. What is the content of the variables?
Why is it hard?

Two problems:

1. What is the content of the variables?
Why is it hard?

Two problems:

1. What is the content of the variables?
Why is it hard?

Two problems:

1. What is the content of the variables?
Why is it hard?

Two problems:

1. What is the content of the variables?

2. What will be the left context when this string is substituted somewhere?
Why is it hard?

Two problems:

1. What is the content of the variables?
2. What will be the left context when this string is substituted somewhere?
Why is it hard?

Two problems:

1. What is the content of the variables?

2. What will be the left context when this string is substituted somewhere?
Why is it hard?

Two problems:

1. What is the content of the variables?

2. What will be the left context when this string is substituted somewhere?
Naive solution

- Extract the all (k-best?) translations from the translation model
- Score them with an LM
- What’s the problem with this?
Outline of DP solution

• Use $n$-order Markov assumption to help us
  • In an $n$-gram LM, words more than $n$ words away will not affect the local (conditional) probability of a word in context
  • This is not generally true, just the Markov assumption!

• General approach
  • Restructure the hypergraph so that LM probabilities decompose along edges.
  • Solves both “problems”
    • we will not know the full value of variables, but we will know “enough”.
    • defer scoring of left context until the context is established.
Hypergraph restructuring

• Note the following three facts:

• If you know $n$ or more consecutive words, the conditional probabilities of the $n$th, $(n+1)$th, ... words can be computed.
  • Therefore: add a feature weight to the edge for words.

• $(n-1)$ words of context to the left is enough to determine the probability of any word
  • Therefore: split nodes based on the $(n-1)$ words on the right side of the span dominated by every node

• $(n-1)$ words on the left side of a span cannot be scored with certainty because the context is not known
  • Therefore: split nodes based on the $(n-1)$ words on the left side of the span dominated by every node
Hypergraph restructuring

• Note the following three facts:

• If you know $n$ or more consecutive words, the conditional probabilities of the $n$th, $(n+1)$th, ... words can be computed.

• Therefore: add a feature weight to the edge for words.

• $(n-1)$ words of context to the left side of the span dominated by every node

• $(n-1)$ words on the left side of a span cannot be scored with certainty because the context is not known

• Therefore: split nodes based on the $(n-1)$ words on the left side of the span dominated by every node

• Split nodes by the $(n-1)$ words on both sides of the convergent edges.
Hypergraph restructuring

- Algorithm ("cube intersection"):
  - For each node $v$ (proceeding in topological order through the nodes)
  - For each edge $e$ with head-node $v$, compute the $(n-1)$ words on the left and right; call this $q_e$
    - Do this by substituting the $(n-1)x2$ word string from the tail node corresponding to the substitution variable
    - If node $vq_e$ does not exist, create it, duplicating all outgoing edges from $v$ so that they also proceed from $vq_e$
    - Disconnect $e$ from $v$ and attach it to $vq_e$
  - Delete $v$
Hypergraph restructuring

![Graph diagram]

- The man: 0.6
- The husband: 0.4
- La mancha: 0.1
- The stain: 0.7
- The gray stain: 0.2

Edges with weights:
- 2's 1: 0.6
- 1 from 2: 0.4
Hypergraph restructuring

-LM Viterbi:
the stain’s the man
Hypergraph restructuring

Let's add a bi-gram language model!
Hypergraph restructuring

Let's add a bi-gram language model!
Hypergraph restructuring

\[ p(\text{mancha}|\text{la}) \]

\[
\begin{align*}
0.6 & \quad \text{the man} \\
0.4 & \quad \text{the husband}
\end{align*}
\]
Hypergraph restructuring

p(mancha|la)

\[
p(\text{mancha}|\text{la})
\]

0.1 \quad \text{la mancha}

0.7 \quad \text{the stain}

0.2 \quad \text{the gray stain}

0.6 \quad \text{the man}

0.4 \quad \text{the husband}

\[
\begin{align*}
X & \quad \rightarrow \quad X \\
\text{X} & \quad \rightarrow \quad \text{X} \\
2 & \quad \rightarrow \quad 1 \\
1 & \quad \text{from} \quad 2
\end{align*}
\]

0.6

0.4


Hypergraph restructuring

\[ p(\text{stain}|\text{the}) \]

- 0.6: the man
- 0.4: the husband
- 0.1: la mancha
- 0.7: the stain
- 0.2: the gray stain

\[ X \quad \text{2's 1} \quad 0.6 \]

\[ X \quad \text{la mancha} \quad \text{1 from 2} \quad 0.4 \]
Hypergraph restructuring

\[ p(\text{stain}|\text{the}) \]

\[
\begin{align*}
0.6 & \quad \text{the man} \\
0.4 & \quad \text{the husband} \\
0.1 & \quad \text{la mancha} \\
0.7 & \quad \text{the stain} \\
0.2 & \quad \text{the gray stain}
\end{align*}
\]
Hypergraph restructuring

\[
p(\text{gray|the}) \times p(\text{stain|gray})
\]

- the man
- the husband
- la mancha
- the stain
- the gray stain

2's 1

0.6
0.4
0.7
0.1
0.2

1 from 2

0.6
0.4
Hypergraph restructuring

$\text{la mancha}$

the stain

the gray stain

$p(\text{gray} | \text{the}) \times p(\text{stain} | \text{gray})$

0.6

0.4

0.1

0.7

0.2

the man

the husband

la mancha

1 from 2 0.4

2's 1 0.6
Hypergraph restructuring
Hypergraph restructuring

0.6  the man
0.4  the husband

0.1  la mancha
0.7  the stain
0.2  the gray stain

2 's 1  0.6
1 from 2  0.4
Hypergraph restructuring

the man
the husband
la mancha
the stain
the gray stain
Hypergraph restructuring

Every node “remembers” enough for edges to compute LM costs
Complexity

- What is the run-time of this algorithm?
Complexity

- What is the run-time of this algorithm?

\[ O(|V||E||Σ|^{2(n-1)}) \]

Going to longer n-grams is exponentially expensive!
Cube pruning

• Expanding every node like this exhaustively is impractical
  • Polynomial time, but really, really big!

• Cube pruning: minor tweak on the above algorithm
  • Compute the k-best expansions at each node
  • Use an estimate (usually a unigram probability) of the unscored left-edge to rank the nodes
Cube pruning

• Widely used for phrase-based and syntax-based MT

• May be applied in conjunction with a bottom-up decoder, or as a second “rescoring” pass

• Nodes may also be grouped together (for example, all nodes corresponding to a certain source span)

• Requirement for topological ordering means translation hypergraph may not have cycles
Reading

- Chapter 11 from the textbook
- Research papers listed in the syllabus