Class 10

Semantic parsing
Semantic Parsing

How many institutes are there in CNRS?

Count(\( \lambda x. \) Type(\( x \), Institute) \( \land \) Parent(\( x \), CNRS))

11
Semantic Parsing

Book me the earliest train from Nancy to Paris on Dec 12th

Semantic parsing

BookTicket(argmax(\(\lambda x.\) From\(x, \) Paris)\(\wedge\) To\(x, \) Nancy)\(\wedge\) Date\(x, 12-12-2017\), Time))

Execution

SNCF train ticket image
Semantic Parsing

[utterance] → Semantic parsing → [program] → Execution → [denotation]
Internet of Things (IoT)
Semantic Parsing

- We saw the neural method: sequence-to-sequence models. Let’s go back to some more traditional CKY-based approaches
Grammars and derivations

**Simple grammar**

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Type 1</th>
<th>Rule</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(lexicon)</td>
<td>Set</td>
<td>$\Rightarrow$ Phrase</td>
<td>Chicago $\Rightarrow$ <em>Chicago</em></td>
</tr>
<tr>
<td>(lexicon)</td>
<td>Binary</td>
<td>$\Rightarrow$ Phrase</td>
<td>PlacesLived $\Rightarrow$ <em>lived in</em></td>
</tr>
<tr>
<td>(join)</td>
<td>Set</td>
<td>$\Rightarrow$ Binary Set</td>
<td>$b.s$ $\Rightarrow$ <em>b s</em></td>
</tr>
<tr>
<td>(intersect)</td>
<td>Root</td>
<td>$\Rightarrow$ Set Set</td>
<td>$s_1 \cap s_2$ $\Rightarrow$ <em>s_1 s_2</em></td>
</tr>
</tbody>
</table>
Grammars and derivations

Simple grammar

(lexicon) Set ⇒ Phrase  
Chicago ⇒ Chicago
(lexicon) Binary ⇒ Phrase  
PlacesLived ⇒ lived in
(join) Set ⇒ Binary Set  
b.s ⇒ b s
(intersect) Root ⇒ Set Set  
s₁ ⊓ s₂ ⇒ s₁ s₂

Type.Person ⊓ PlaceLived.Chicago
intersec

Type.Person who PlaceLived.Chicago

lexicon

people

lived in Chicago

lexicon

lexicon
A grammar $G$ is a 4-tuple:

$$
\Sigma: \quad \text{chicago, lived} \\
\mathcal{N}: \quad \text{Binary, Set, Root} \\
\text{Root} \in \mathcal{N}: \text{start symbol} \\
\mathcal{R}: \quad \text{grammar rules}
$$
Grammar rules

Simple grammar

(lexicon) Set ⇒ Phrase
Chicago ⇒ Chicago

(lexicon) Binary ⇒ Phrase
PlacesLived ⇒ lived in

(join) Set ⇒ Binary Set
b.s ⇒ b s

(intersect) Root ⇒ Set Set
s₁ ⊓ s₂ ⇒ s₁ s₂

A rule \( r \in \mathcal{R} \) has:

\[ A \in \mathcal{N} : \quad \text{left-hand-side (LHS) non-terminal} \]
\[ \alpha \in (\mathcal{N} \cup \Sigma)^+ : \quad \text{right-hand-side} \]
\[ f : \quad \text{semantic function for building derivations} \]

Phrase is a compact way to write all \( \alpha \in \Sigma^+ \)

Semantic functions are the key component!
A derivation tree $d^A_{i:j}$ over a span $x = (w_i, \ldots, w_j)$:

- Has category $d.c = A \in \mathcal{N}$
- Has logical form $d.z$
- Has children that are derivations or in $\Sigma^+$
Semantic functions

A function $f : D^k \rightarrow 2^D$

Example:

- $r = A \rightarrow B C [f]$
- $D^A_{i:j} = f(d^B_{i:k}, d^C_{k:j})$

Lexicon function:

$$\text{Lex}(\text{lincoln}) = \left\{ \begin{array}{c}
\text{Entity: AbeLincoln, lincoln,}
\text{Entity: LincolnFilm, lincoln,} \\
\end{array} \right\}$$
Semantic functions

Join function:

\[
\text{Join}(\text{Entity: AbeLincoln, Binary: PlaceOfBirthOf}) = \{\text{Entity: AbeLincoln, Binary: PlaceOfBirthOf}\}
\]

Merge function:

\[
\text{Intersect}(\text{Set: Type.City, Set: ReleaseDateOf.LincolnFilm}) = \{}
\]

Return sets (unlike CFGs)
Semantic function

IdentityFn  Copy logical form from only child
SelectFn(i)  Select logical form from child $i$ (skip words)
DateFn      Handle date and time language expressions
Semantic function

IdentityFn  Copy logical form from only child
SelectFn(i)  Select logical form from child i (skip words)
DateFn  Handle date and time language expressions
ContextFn  Integrate arbitrary context

What city was abraham lincoln born?
How about his wife?
Semantic function

IdentityFn  Copy logical form from only child
SelectFn(i) Select logical form from child $i$ (skip words)
DateFn     Handle date and time language expressions
ContextFn  Integrate arbitrary context

What city was abraham lincoln born?
How about his wife?

Flexible! arbitrary logic can be used
Outline

• Parsing
  – CKY
  – Approximations
  – Some tricks
  – Learning agenda-based parsers
Parsing

We need to compute

\[ \text{argmax}_{d \in D(x)} \quad p_\theta(d \mid x) \]

Inference: Find best tree given model
Parsing

We need to compute

\[ \text{argmax}_{d \in \mathcal{D}(x)} p_{\theta}(d | x) \]

**Inference:** Find best tree given model

\[ E_{q_{\theta}(d|x)}[\phi(x, d)] \]
\[ E_{p_{\theta}(d|x)}[\phi(x, d)] \]

**Learning:** Computing gradient

\[ \nabla O(\theta) = E_{q_{\theta}(d|x)}[\phi(x, d)] - E_{p_{\theta}(d|x)}[\phi(x, d)] \]
Outline

• Parsing
  – CKY
  – Approximations
  – Some tricks
  – Learning agenda-based parsers
Finding best tree

Can we use CKY?

Only if \( f(x, y) = \sum_{r \in d} g(x, r)^\top \theta \)

\( r = (A, B, C, i, j, k) \) is a rule production

Why not? is \( g \) sufficient?
Finding best tree

Maybe we can only have ”type” features?

Root : WonElection(Type(Country))

Binary : WonElection
Set : Type(Country)

won

country

Root : WonAward(Type(Country))

Binary : WonAward
Set : Type(Country)

won

country

**Type features**: binary-unary type match
Finding best tree

Maybe we can only have "type" features?

Root: \( \text{WonElection(Type(Country))} \)

Binary: \( \text{WonElection} \)

Set: \( \text{Type(Country)} \)

\( \text{won} \)

\( \text{country} \)

Root: \( \text{WonAward(Type(Country))} \)

Binary: \( \text{WonAward} \)

Set: \( \text{Type(Country)} \)

\( \text{won} \)

\( \text{country} \)

**Type features**: binary-unary type match

We can get exact decoding if augment state:

- Binary-Person-Country
- Binary-Person-Award
- Set-Country
- Set-Award

**Explosion in grammar size!**

If number of types is \( T \), then \( \mathcal{N} \) grows by at least \( T^2 \)
Finding best tree

In practice many features depend on the logical form itself

- Bridging features

If we could hold the logical form as part of the state we could use dynamic programming, but usually that is not possible
Logical form features

\[ E_{q_\theta}(d|x)[\phi(x, d)] \]

\[ = \sum_{d \in D(x)} \frac{\exp(\phi(x, d)^\top \theta) \cdot R(d)}{\sum_{d'} \exp(\phi(x, d')^\top \theta) \cdot R(d')} \phi(x, d) \]

\( R(d) \) depends on entire logical form - too large!
Logical form features

\[ E_{q_\theta}(d|x)[\phi(x, d)] = \sum_{d \in \mathcal{D}(x)} \frac{\exp(\phi(x, d) \top \theta) \cdot R(d)}{\sum_{d'} \exp(\phi(x, d') \top \theta) \cdot R(d')} \phi(x, d) \]

\( R(d) \) depends on entire logical form - too large!

Do approximate inference!
Outline

- Parsing
  - CKY
  - Approximations
  - Some tricks
  - Learning agenda-based parsers
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

abraham lincoln born in
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

```
|    | 20 | abraham | lincoln |
|----|----|---------|
|    |    | born    |
|    |    | in      |
```
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

$K$

abraham lincoln born in
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

![Beam Parsing Diagram](image)
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

```
abraham lincoln born in
K K
```

Diagram:

```
K  K
abraham lincoln born in
```
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

![chart diagram]

- Abraham Lincoln born in
- Chart cell combinations

$$K \quad K \quad K$$

```
abraham
lincoln
born
in
```
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

\[ K \quad K \quad K \quad 508 \]

\[ \text{abraham lincoln born in} \]
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

```
abraham lincoln born in K K K K
```
Beam parsing

- Keep top-\(K\) derivations in every chart cell
- Combine chart cells in all possible ways
Beam parsing

• Keep top-\(K\) derivations in every chart cell
• Combine chart cells in all possible ways

\[
\begin{array}{cccc}
K & & & \\
K & K & & \\
K & K & K & \\
abraham & lincoln & born & in \\
\end{array}
\]
Beam parsing

• Keep top-$K$ derivations in every chart cell

• Combine chart cells in all possible ways
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

\[
\begin{array}{cccc}
K & K & K & K \\
K & K & K & K \\
abraham & lincoln & born & in
\end{array}
\]
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

```
abraham lincoln born in
```

\[\begin{array}{cccc}
  K & K & K & K \\
  K & K & K & K \\
  abraham & lincoln & born & in \\
\end{array}\]
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

\[ \begin{array}{cccc}
  & K & K & K \\
  K & & & \\
  K & K & & \\
  K & K & K & K \\
\end{array} \]

\textit{abraham} \hspace{1em} \textit{lincoln} \hspace{1em} \textit{born} \hspace{1em} \textit{in}
Beam parsing

• Keep top-$K$ derivations in every chart cell
• Combine chart cells in all possible ways
Beam parsing

• Keep top-$K$ derivations in every chart cell

• Combine chart cells in all possible ways
Beam parsing

• Keep top-\(K\) derivations in every chart cell
• Combine chart cells in all possible ways

\[ \text{abraham lincoln born in} \]

\[
\begin{array}{cccc}
K & K & K & K \\
K & K & K & K \\
K & K & K & K \\
\end{array}
\]
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

```
abraham lincoln born in
```

```
K K K K
K K K
K K
K2
8
```
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

```
abraham lincoln born in
K K K K
K K K
K K
K
```

![Diagram](attachment:image.png)
Beam parsing

• Keep top-$K$ derivations in every chart cell
• Combine chart cells in all possible ways

Building a chart cell is now $O(K^2 + \mathcal{L})$ and not $O(1)$
Beam parsing

- Keep top-$K$ derivations in every chart cell
- Combine chart cells in all possible ways

Assume root beam $\hat{D}(x) \sim D(x)$

$$E_{p_{\theta}}[\phi(x, d)] = \sum_{d \in \hat{D}(x)} p_{\theta}(d) \phi(x, d)$$

$$E_{q_{\theta}}[\phi(x, d)] = \sum_{d \in \hat{D}(x)} q_{\theta}(d) \phi(x, d)$$
Beam Parsing variants

- Keep top-$K$ for every span $(i, j)$ and category $A$
- Keep top-$K$ for every span $(i, j)$
Beam Parsing variants

- Keep top-$K$ for every span $(i, j)$ and category $A$
- Keep top-$K$ for every span $(i, j)$

Prune cell with probability ratio
Beam Parsing variants

• Keep top-$K$ for every span $(i,j)$ and category $A$

• Keep top-$K$ for every span $(i,j)$

Prune cell with probability ratio

for $\alpha = 10$:

$[0.6, 0.2, 0.1, 0.05, 0.03, 0.02]$
Beam Parsing variants

• Keep top-$K$ for every span $(i, j)$ and category $A$

• Keep top-$K$ for every span $(i, j)$

Prune cell with probability ratio

for $\alpha = 10$:

$[0.6, 0.2, 0.1]$
Beam Parsing variants

- Keep top-$K$ for every span $(i, j)$ and category $A$
- Keep top-$K$ for every span $(i, j)$

Prune cell with probability ratio

for $\alpha = 10$:

$[0.6, 0.2, 0.1]$

Learn a classifier that predict $K$ for every chart cell, [Bodenstab et al., 2011]
Outline

- Parsing
  - CKY
  - Approximations
  - Some tricks
  - Learning agenda-based parsers
Coarse pruning

Most of parsing time is in building logical forms and extracting features

- Extracting lexical items
- Executing logical forms
- Suggesting binary predicates for bridging

Do 2-pass parsing. First only recognizing chart cells (without feature extraction), and then again pruning unnecessary chart cells
Coarse pruning

1. Build derivations without logical forms and features (one derivation per rule application) - fast!
Coarse pruning

1. Build derivations without logical forms and features (one derivation per rule application) - fast!

2. Record cell reachability for every rule application: \((A, i, j) \rightarrow (B, i, k), (C, k, j)\)
Coarse pruning

1. Build derivations without logical forms and features (one derivation per rule application) - fast!

2. Record cell reachability for every rule application: \((A, i, j) \rightarrow (B, i, k), (C, k, j)\)

3. Collect all chart cells that are reachable from \((S, 1, n)\)
Coarse pruning

1. Build derivations without logical forms and features (one derivation per rule application) - fast!

2. Record cell reachability for every rule application: \((A, i, j) \rightarrow (B, i, k), (C, k, j)\)

3. Collect all chart cells that are reachable from \((S, 1, n)\)

4. Parse again ignoring unreachable states
Coarse pruning

1. Build derivations without logical forms and features (one derivation per rule application) - fast!

2. Record cell reachability for every rule application: \((A, i, j) \rightarrow (B, i, k), (C, k, j)\)

3. Collect all chart cells that are reachable from \((S, 1, n)\)

4. Parse again ignoring unreachable states

We know unreachable states do not contribute to root derivations over sentence
Coarse parsing

Root 0.0273
Root 0.007
Root 0.162
Binary 0.9
Set 0.7

won

Binary 0.1
Set 0.3
Mod 1.0
Conj 1.0

country
Coarse parsing

\[
\begin{align*}
\pi(0, 2, \text{Root}) & \rightarrow (\pi(0, 1, \text{Binary}), \pi(1, 2, \text{Set})) \\
& \quad (\pi(0, 1, \text{Set}), \pi(1, 2, \text{Binary})) \\
& \quad (\pi(0, 1, \text{Set}), \pi(1, 2, \text{Mod}))
\end{align*}
\]

\[
\begin{align*}
\pi(0, 1, \text{Binary}) & \rightarrow \\
\pi(0, 1, \text{Set}) & \rightarrow \\
\pi(1, 2, \text{Binary}) & \rightarrow \\
\pi(1, 2, \text{Set}) & \rightarrow \\
\pi(1, 2, \text{Mod}) & \rightarrow \\
\pi(1, 2, \text{Conj}) & \rightarrow \\
\end{align*}
\]
Unary rules in CKY

Unary rules are convenient when designing grammars

- Set → Entity
- Set → Unary
Unary rules in CKY

Unary rules are convenient when designing grammars

- Set $\rightarrow$ Entity
- Set $\rightarrow$ Unary

Assume:

rules $A \rightarrow B$ and $B \rightarrow C$

A derivation $d^C_{i:j}$

Computing $\pi(i, j, A)$ before $\pi(i, j, B)$

- Why this fails?
Unary rules

Solution:

Construct an acyclic (!) directed graph $G = (V, E)$:

$V = \mathcal{N}$

$E$ are unary rules
Unary rules

Solution:

Construct an acyclic (!) directed graph $G = (V, E)$:

$V = N$

$E$ are unary rules

Sort topologically edges and apply rules from end to start after binary rules

- Guaranteed to apply $B \rightarrow C$ before $A \rightarrow B$
Unary rules

Solution:

Construct an acyclic (!) directed graph $G = (V, E)$:

$V = \mathcal{N}$

$E$ are unary rules

Sort topologically edges and apply rules from end to start after binary rules

- Guaranteed to apply $B \rightarrow C$ before $A \rightarrow B$

Need to verify grammar has no unary cycles
Outline

- Learning
  - Overview
  - Details
Outline

• Learning
  – Overview
  – Details
Supervision in syntactic parsing

Input:

They play football

Output:
Supervision in semantic parsing

Input:

**Heavy supervision**

*How tall is Lebron James?*
*HeightOf.LebronJames*

*What is Steph Curry’s daughter called?*
*ChildrenOf.StephCurry \sqcap Gender.Female*

*Youngest player of the Cavaliers*
*arg\,min(\text{PlayerOf.Cavaliers, BirthDateOf})*

...

**Light supervision**

*How tall is Lebron James?*
*203cm*

*What is Steph Curry’s daughter called?*
*Riley Curry*

*Youngest player of the Cavaliers*
*Kyrie Irving*

...
Supervision in semantic parsing

Input:

Heavy supervision

How tall is Lebron James?
HeightOf.LebronJames

What is Steph Curry’s daughter called?
ChildrenOf.StephCurry ⊓ Gender.Female

Youngest player of the Cavaliers
arg min(PlayerOf.Cavaliers, BirthDateOf)

... 

Light supervision

How tall is Lebron James?
203cm

What is Steph Curry’s daughter called?
Riley Curry

Youngest player of the Cavaliers
Kyrie Irving

...

Output:

Clay Thompson’s weight ➔ ClayThompson’s weight ➔ 205 lbs
Learning in a nutshell

*utterance*

0. Define model for derivations
Learning in a nutshell

0. Define model for derivations
1. Generate candidate derivations (later)
Learning in a nutshell

0. Define model for derivations
1. Generate candidate derivations (later)
2. Label as correct and incorrect
0. Define model for derivations
1. Generate candidate derivations (later)
2. Label as correct and incorrect
3. Update model to favor correct trees
Training intuition

Where did Mozart tupress?

Vienna
Training intuition

Where did Mozart tupress?

PlaceOfBirth.WolfgangMozart
PlaceOfDeath.WolfgangMozart
PlaceOfMarriage.WolfgangMozart

Vienna
Training intuition

Where did Mozart tupress?

PlaceOfBirth.WolfgangMozart ⇒ Salzburg
PlaceOfDeath.WolfgangMozart ⇒ Vienna
PlaceOfMarriage.WolfgangMozart ⇒ Vienna

Vienna
Training intuition

Where did Mozart typress?

PlaceOfBirth.WolfgangMozart ⇒ Salzburg
PlaceOfDeath.WolfgangMozart ⇒ Vienna
PlaceOfMarriage.WolfgangMozart ⇒ Vienna

Vienna
Training intuition

Where did Mozart tupress?
PlaceOfBirth.WolfgangMozart ⇒ Salzburg
PlaceOfDeath.WolfgangMozart ⇒ Vienna
PlaceOfMarriage.WolfgangMozart ⇒ Vienna

Vienna

Where did Hogarth tupress?
Training intuition

Where did Mozart tupress?

PlaceOfBirth.WolfgangMozart ⇒ Salzburg
PlaceOfDeath.WolfgangMozart ⇒ Vienna
PlaceOfMarriage.WolfgangMozart ⇒ Vienna

Vienna

Where did Hogarth tupress?

PlaceOfBirth.WilliamHogarth
PlaceOfDeath.WilliamHogarth
PlaceOfMarriage.WilliamHogarth

London
Training intuition

*Where did Mozart tuple*?

PlaceOfBirth.WolfgangMozart \(\rightarrow\) Salzburg

PlaceOfDeath.WolfgangMozart \(\rightarrow\) Vienna

PlaceOfMarriage.WolfgangMozart \(\rightarrow\) Vienna

Vienna

*Where did Hogarth tuple*?

PlaceOfBirth.WilliamHogarth \(\rightarrow\) London

PlaceOfDeath.WilliamHogarth \(\rightarrow\) London

PlaceOfMarriage.WilliamHogarth \(\rightarrow\) Paddington

London
Training intuition

Where did Mozart tupress?

PlaceOfBirth.WolfgangMozart ⇒ Salzburg
PlaceOfDeath.WolfgangMozart ⇒ Vienna
PlaceOfMarriage.WolfgangMozart ⇒ Vienna

Vienna

Where did Hogarth tupress?

PlaceOfBirth.WilliamHogarth ⇒ London
PlaceOfDeath.WilliamHogarth ⇒ London
PlaceOfMarriage.WilliamHogarth ⇒ Paddington

London
Training intuition

Where did Mozart tupress?

PlaceOfBirth.WolfgangMozart ⇒ Salzburg

PlaceOfDeath.WolfgangMozart ⇒ Vienna

PlaceOfMarriage.WolfgangMozart ⇒ Vienna

Vienna

Where did Hogarth tupress?

PlaceOfBirth.WilliamHogarth ⇒ London

PlaceOfDeath.WilliamHogarth ⇒ London

PlaceOfMarriage.WilliamHogarth ⇒ Paddington

London
Outline

• Learning
  – Overview
  – Details
Constructing derivations

Type.Person \(\cap\) PlaceLived.Chicago

**intersect**

Type.Person \(\sqcap\) PlaceLived.Chicago

**join**

Type.Person \(\cap\) PlaceLived.Chicago

**intersect**

Type.Person \(\cap\) PlaceLived.Chicago

**join**

Type.Person \(\sqcap\) PlaceLived.Chicago

**intersect**

Type.Person \(\cap\) PlaceLived.Chicago

**intersect**

Type.Person \(\sqcap\) PlaceLived.Chicago

**intersect**
Many possible derivations!

\[ x = \textit{people who have lived in Chicago} \]

set of candidate derivations \( \mathcal{D}(x) \)
Feature vector and parameters in $\mathbb{R}^F$:

\[
\phi(x, d) \quad \theta \quad \leftarrow \text{learned}
\]

apply join $1 \quad 1.2$
apply intersect $1 \quad 0.6$
apply lexicon $3 \quad 2.1$

\textit{lived} maps to \textit{PlacesLived} $1 \quad 3.1$

\textit{lived} maps to \textit{PlaceOfBirth} $0 \quad -0.4$

\textit{born} maps to \textit{PlaceOfBirth} $0 \quad 2.7$

... $\quad ... \quad ...$
$x$: utterance

d: derivation

Feature vector and parameters in $\mathbb{R}^F$:

$$\phi(x, d) \theta \leftarrow \text{learned}$$

- apply join 1 1.2
- apply intersect 1 0.6
- apply lexicon 3 2.1
- $lived$ maps to $\text{PlacesLived}$ 1 3.1
- $lived$ maps to $\text{PlaceOfBirth}$ 0 -0.4
- $born$ maps to $\text{PlaceOfBirth}$ 0 2.7

... ... ...

$$\text{Score}_\theta(x, d) = \phi(x, d) ^\top \theta =$$

$$1.2 \cdot 1 + 0.6 \cdot 1 + 2.1 \cdot 3 + 3.1 \cdot 1 + -0.4 \cdot 0 + 2.7 \cdot 0 + \ldots$$

Deep learning alert!

The feature vector $\phi(x, d)$ is constructed by hand.
Deep learning alert!

The feature vector $\phi(x, d)$ is constructed by hand

Constructing good features is hard
Deep learning alert!

The feature vector $\phi(x,d)$ is constructed by hand

Constructing good features is hard

Algorithms are likely to do it better
Deep learning alert!

The feature vector \( \phi(x, d) \) is constructed by hand.

Constructing good features is hard.

Algorithms are likely to do it better.

Perhaps we can train \( \phi(x, d) \)

\[ \phi(x, d) = F_\psi(x, d), \text{ where } \psi \text{ are the parameters} \]
Log-linear model

Candidate derivations: $\mathcal{D}(x)$

Model: distribution over derivations $d$ given utterance $x$

$$p_\theta(d \mid x) = \frac{\exp(Score_\theta(x,d))}{\sum_{d' \in \mathcal{D}(x)} \exp(Score_\theta(x,d'))}$$
Log-linear model

Candidate derivations: $D(x)$

Model: distribution over derivations $d$ given utterance $x$

$$p_\theta(d \mid x) = \frac{\exp(\text{Score}_\theta(x,d))}{\sum_{d' \in D(x)} \exp(\text{Score}_\theta(x,d'))}$$

score$_\theta(x, d)$ $\left[1, 2, 3, 4\right]$
Log-linear model

Candidate derivations: \( D(x) \)

Model: distribution over derivations \( d \) given utterance \( x \)

\[
p_\theta(d \mid x) = \frac{\exp(\text{Score}_\theta(x,d))}{\sum_{d' \in D(x)} \exp(\text{Score}_\theta(x,d'))}
\]

\( \text{score}_\theta(x, d) \)

\[
p_\theta(d \mid x) = \left[ \frac{e}{e+e^2+e^3+e^4}, \frac{e^2}{e+e^2+e^3+e^4}, \frac{e^3}{e+e^2+e^3+e^4}, \frac{e^4}{e+e^2+e^3+e^4} \right]
\]

Parsing: find the top-\( K \) derivation trees \( D_\theta(x) \)
Features

Dense features:

- intersection=0.67
- ent-popularity:HIGH
- denoation-size:1
Features

Dense features:

- intersection=0.67
- ent-popularity:HIGH
- denoation-size:1

Sparse features:

- bridge-binary:STUDY
- born:PlaceOfBirth
- city:Type.Location
Features

Dense features:
- intersection=0.67
- ent-popularity:HIGH
- denotation-size:1

Sparse features:
- bridge-binary:STUDY
- born:PlaceOfBirth
- city:Type.Location

Syntactic features:
- ent-pos:NNP NNP
- join-pos:V NN
- skip-pos:IN
Features

Dense features:
• intersection=0.67
• ent-popularity:HIGH
• denotation-size:1

Sparse features:
• bridge-binary:STUDY
• born:PlaceOfBirth
• city:Type.Location

Syntactic features:
• ent-pos:NNP NNP
• join-pos:V NN
• skip-pos:IN

Grammar features:
• Binary->Verb
Learning $\theta$: marginal maximum-likelihood

Training data:

<table>
<thead>
<tr>
<th>What’s Bulgaria’s capital?</th>
<th>Sofia</th>
</tr>
</thead>
</table>

| What movies has Tom Cruise been in? | TopGun, VanillaSky,… |

<table>
<thead>
<tr>
<th>What’s Bulgaria’s capital?</th>
<th>CapitalOf.Bulgaria</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>What movies has Tom Cruise been in?</th>
<th>Type.Movie ⊓ HasPlayed.TomCruise</th>
</tr>
</thead>
</table>

...
Learning $\theta$: marginal maximum-likelihood

Training data:

<table>
<thead>
<tr>
<th>What’s Bulgaria’s capital?</th>
<th>Sofia</th>
</tr>
</thead>
<tbody>
<tr>
<td>What movies has Tom Cruise been in?</td>
<td>TopGun, VanillaSky,…</td>
</tr>
</tbody>
</table>

... | ... |

<table>
<thead>
<tr>
<th>What’s Bulgaria’s capital?</th>
<th>CapitalOf.Bulgaria</th>
</tr>
</thead>
<tbody>
<tr>
<td>What movies has Tom Cruise been in?</td>
<td>Type.Movie $\sqcap$ HasPlayed.TomCruise</td>
</tr>
</tbody>
</table>

\[
\arg\max_{\theta} \sum_{i=1}^{n} \log p_{\theta}(y^{(i)} \mid x^{(i)}) = \arg\max_{\theta} \sum_{i=1}^{n} \log \sum_{d^{(i)}} p_{\theta}(d^{(i)} \mid x^{(i)}) R(d^{(i)})
\]
Learning $\theta$: marginal maximum-likelihood

Training data:

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What's Bulgaria's capital?</td>
<td>Sofia</td>
</tr>
<tr>
<td>What movies has Tom Cruise been in?</td>
<td>TopGun, VanillaSky,...</td>
</tr>
</tbody>
</table>

\[
\arg \max_\theta \sum_{i=1}^n \log p_\theta(y^{(i)} \mid x^{(i)}) =
\]

\[
\arg \max_\theta \sum_{i=1}^n \log \sum_{d^{(i)}} p_\theta(d^{(i)} \mid x^{(i)}) R(d^{(i)})
\]

\[
R(d) = \begin{cases} 
1 & d.z = z^{(i)} \\
0 & o/w 
\end{cases}
\]

\[
R(d) = \begin{cases} 
1 & [d.z]_K = y^{(i)} \\
0 & o/w 
\end{cases}
\]

\[
R(d) = F_1([d.z]_K, y^{(i)})
\]
Optimization: stochastic gradient descent

For every example:

\[ O(\theta) = \log \sum_d p_\theta(d \mid x) R(d) \]
\[ \nabla O(\theta) = E_{q_\theta(d \mid x)}[\phi(x, d)] - E_{p_\theta(d \mid x)}[\phi(x, d)] \]
\[ p_\theta(d \mid x) \propto \exp(\phi(x, d)^\top \theta) \]
\[ q_\theta(d \mid x) \propto \exp(\phi(x, d)^\top \theta) \cdot R(d) \]
Optimization: stochastic gradient descent

For every example:

\[ O(\theta) = \log \sum_d p_\theta(d \mid x) R(d) \]
\[ \nabla O(\theta) = E_{q_\theta(d \mid x)}[\phi(x, d)] - E_{p_\theta(d \mid x)}[\phi(x, d)] \]
\[ p_\theta(d \mid x) \propto \exp(\phi(x, d)^\top \theta) \]
\[ q_\theta(d \mid x) \propto \exp(\phi(x, d)^\top \theta) \cdot R(d) \]

\[ p_\theta(\mathcal{D}(x)) = [0.2, 0.1, 0.1, 0.6] \]
\[ R(\mathcal{D}(x)) = [1, 0, 0, 1] \]
Optimization: stochastic gradient descent

For every example:

\[
O(\theta) = \log \sum_d p_\theta(d \mid x) R(d) \\
\nabla O(\theta) = E_{q_\theta(d \mid x)}[\phi(x, d)] - E_{p_\theta(d \mid x)}[\phi(x, d)] \\
p_\theta(d \mid x) \propto \exp(\phi(x, d) \top \theta) \\
q_\theta(d \mid x) \propto \exp(\phi(x, d) \top \theta) \cdot R(d)
\]

\[
p_\theta(\mathcal{D}(x)) = [0.2, 0.1, 0.1, 0.6] \\
R(\mathcal{D}(x)) = [1, 0, 0, 1] \\
q_\theta(\mathcal{D}(x)) = [0.25, 0, 0, 0.75] \\
q_\theta = \frac{p_\theta}{p_\theta R}
\]
Optimization: stochastic gradient descent

For every example:

\[ O(\theta) = \log \sum_d p_\theta(d \mid x) R(d) \]
\[ \nabla O(\theta) = E_{q_\theta(d \mid x)}[\phi(x, d)] - E_{p_\theta(d \mid x)}[\phi(x, d)] \]
\[ p_\theta(d \mid x) \propto \exp(\phi(x, d) \top \theta) \]
\[ q_\theta(d \mid x) \propto \exp(\phi(x, d) \top \theta) \cdot R(d) \]

\[ p_\theta(\mathcal{D}(x)) = [0.2, 0.1, 0.1, 0.6] \]
\[ R(\mathcal{D}(x)) = [1, 0, 0, 1] \]
\[ q_\theta(\mathcal{D}(x)) = [0.25, 0, 0, 0.75] \]
\[ q_\theta = \frac{p_\theta}{p_\theta R} \]

Gradient:

\[ 0.05 \cdot \phi(x, d_1) - 0.1 \cdot \phi(x, d_2) - 0.1 \cdot \phi(x, d_3) + 0.15 \cdot \phi(x, d_4) \]
Training

Input: \( \{x_i, y_i\}_{i=1}^{n} \)

Output: \( \theta \)
Training

Input: \( \{x_i, y_i\}_{i=1}^{n} \)

Output: \( \theta \)

\(\theta \leftarrow 0\)
Training

Input: $\{x_i, y_i\}_{i=1}^{n}$

Output: $\theta$

$\theta \leftarrow 0$

for iteration $\tau$ and example $i$

$D(x_i) \leftarrow \arg \max^K (p_{\theta}(d | x_i))$
Training

Input: \( \{x_i, y_i\}_{i=1}^n \)

Output: \( \theta \)

\( \theta \leftarrow 0 \)

for iteration \( \tau \) and example \( i \)

\( \mathcal{D}(x_i) \leftarrow \text{arg max}_K (p_{\theta}(d \mid x_i)) \)

\( \theta \leftarrow \theta + \eta_{\tau,i} (E_{q_{\theta}(d\mid x_i)}[\phi(x_i,d)] - E_{p_{\theta}(d\mid x_i)}[\phi(x_i,d)]) \)
Training

Input: \( \{x_i, y_i\}_{i=1}^{n} \)

Output: \( \theta \)

\[ \theta \leftarrow 0 \]

for iteration \( \tau \) and example \( i \)

\[ D(x_i) \leftarrow \arg \max_K (p_\theta(d|x_i)) \]

\[ \theta \leftarrow \theta + \eta_{\tau,i} (E_{q_\theta(d|x_i)}[\phi(x_i, d)] - E_{p_\theta(d|x_i)}[\phi(x_i, d)]) \]

\( \eta_{\tau,i} \): learning rate

Regularization often added (L2, L1, ...)

15
Training (structured perceptron)

Input: $\{x_i, y_i\}_{i=1}^n$

Output: $\theta$
Training (structured perceptron)

Input: $\{x_i, y_i\}_{i=1}^n$

Output: $\theta$

$\theta \leftarrow 0$
Training (structured perceptron)

Input: \( \{x_i, y_i\}_{i=1}^{n} \)

Output: \( \theta \)

\( \theta \leftarrow 0 \)

for iteration \( \tau \) and example \( i \)

\( \hat{d} \leftarrow \arg \max (p_{\theta}(d \mid x_i)) \)

\( d^* \leftarrow \arg \max (q_{\theta}(d \mid x_i)) \)
Training (structured perceptron)

Input: \( \{x_i, y_i\}_{i=1}^{n} \)

Output: \( \theta \)

\( \theta \leftarrow 0 \)

for iteration \( \tau \) and example \( i \)

\( \hat{d} \leftarrow \arg \max(p_{\theta}(d \mid x_i)) \)

\( d^* \leftarrow \arg \max(q_{\theta}(d \mid x_i)) \)

if \( [d^*]_\mathcal{K} \neq [\hat{d}]_\mathcal{K} \)

\( \theta \leftarrow \theta + \phi(x_i, d^*) - \phi(x_i, \hat{d}) \)
Training (structured perceptron)

Input: \( \{x_i, y_i\}_{i=1}^n \)

Output: \( \theta \)

\[ \theta \leftarrow 0 \]

for iteration \( \tau \) and example \( i \)

\[ \hat{d} \leftarrow \arg \max(p_\theta(d | x_i)) \]

\[ d^* \leftarrow \arg \max(q_\theta(d | x_i)) \]

if \[ [d^*]_\mathcal{K} \neq [\hat{d}]_\mathcal{K} \]

\[ \theta \leftarrow \theta + \phi(x_i, d^*) - \phi(x_i, \hat{d}) \]

Regularization often added with weight averaging
Training

Other simple variants exist:

• E.g., cost-sensitive max-margin training

• That is, find pairs of good and bad derivations that look different but have similar scores and update on those
Introduction

• Traditional semantic parsers used grammar based parsing algorithms

• Neural semantic parsing has moved towards using encoder decoder models

• Decoders are usually recurrent neural networks that produce sequences

• But they can produce outputs that are not valid (syntactically or semantically)!
Example from WikiTableQuestions

<table>
<thead>
<tr>
<th>Athlete</th>
<th>Nation</th>
<th>Olympics</th>
<th>Medals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gillis Grafström</td>
<td>Sweden (SWE)</td>
<td>1920–1932</td>
<td>4</td>
</tr>
<tr>
<td>Evgeni Plushenko</td>
<td>Russia (RUS)</td>
<td>2002–2014</td>
<td>4</td>
</tr>
<tr>
<td>Karl Schäfer</td>
<td>Austria (AUT)</td>
<td>1928–1936</td>
<td>2</td>
</tr>
<tr>
<td>Katarina Witt</td>
<td>East Germany (GDR)</td>
<td>1984–1988</td>
<td>2</td>
</tr>
<tr>
<td>Tenley Albright</td>
<td>United States (USA)</td>
<td>1952-1956</td>
<td>2</td>
</tr>
<tr>
<td>Kim Yu-na</td>
<td>South Korea (KOR)</td>
<td>2010–2014</td>
<td>2</td>
</tr>
<tr>
<td>Patrick Chan</td>
<td>Canada (CAN)</td>
<td>2014</td>
<td>2</td>
</tr>
</tbody>
</table>

Question:
Which athlete was from South Korea after 2010?

((reverse athlete)
(and
(nation south_korea)
(year ((reverse date) (>= 2010-mm-dd))))

[Pasupat and Liang, 2015]
Seq2Seq Output Space

<table>
<thead>
<tr>
<th>Athlete</th>
<th>Nation</th>
<th>Olympics</th>
<th>Medals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kim Yu-na</td>
<td>South Korea (KOR)</td>
<td>2010–2014</td>
<td>2</td>
</tr>
<tr>
<td>Tenley Albright</td>
<td>United States (USA)</td>
<td>1952-1956</td>
<td>2</td>
</tr>
</tbody>
</table>

LSTM

reverse athlete argmax and
2010.mm.dd nation
(south_korea)
......
Seq2Seq Output Space

<table>
<thead>
<tr>
<th>Athlete</th>
<th>Nation</th>
<th>Olympics</th>
<th>Medals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kim Yu-na</td>
<td>South Korea (KOR)</td>
<td>2010–2014</td>
<td>2</td>
</tr>
<tr>
<td>Tenley Albright</td>
<td>United States (USA)</td>
<td>1952-1956</td>
<td>2</td>
</tr>
</tbody>
</table>
Seq2Seq Output Space

<table>
<thead>
<tr>
<th>Athlete</th>
<th>Nation</th>
<th>Olympics</th>
<th>Medals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kim Yu-na</td>
<td>South Korea (KOR)</td>
<td>2010–2014</td>
<td>2</td>
</tr>
<tr>
<td>Tenley Albright</td>
<td>United States (USA)</td>
<td>1952-1956</td>
<td>2</td>
</tr>
</tbody>
</table>
Example from ATIS

*flights from Dallas leaving after 4 in the afternoon*

\[
\text{\texttt{(lambda } \$0 \texttt{ e (and (> (departure\_time } \$0 \texttt{) 1600:ti) (from } \$0 \texttt{ dallas:ci)))}}
\]

[Hemphill et al., 1990; Dahl et al., 1994]
Constrained Decoding

- Constrain the output space to selections that matter
- **Inference:** Avoid invalid parses
- **Training:** Do not waste modeling power in distinguishing invalid parses from valid ones!

**Token-based Decoding:**
The output space is tokens, but they are constrained to be relevant at each time step.

**Grammar-based Decoding:**
The output space is production rules, and a grammar defines the constraints.
Constrained Decoding

- Constrain the output space to selections that matter
- **Inference:** Avoid invalid parses
- **Training:** Do not waste modeling power in distinguishing invalid parses from valid ones!

**Token-based Decoding**

Dong and Lapata. 2016. *Language to Logical Form with Neural Attention*. In ACL.

Dong and Lapata. 2018. *Coarse-to-Fine Decoding for Neural Semantic Parsing*. In ACL.


**Grammar-based Decoding:**


Yin and Neubig. 2017. *A Syntactic Neural Model for General Purpose Code Generation*. In ACL.

Krishnamurthy, Dasigi, and Gardner. 2017. *Neural Semantic Parsing with Type Constraints for Semi-Structured Tables*. In EMNLP.
Token-based Constrained Decoding
Constraining output structure: Seq2Tree

Flights from Dallas leaving after 4 in the afternoon

(lamba $0 e
  (and
    (> (departure_time $0) 1600:ti)
    (from $0 dallas:ci))))

[Dong and Lapata, 2016]
Constraining output structure: Seq2Tree

Flights from Dallas leaving after 4 in the afternoon

(lambda $0 e <n>)

[Dong and Lapata, 2016]
Constraining output structure: Seq2Tree

Flights from Dallas leaving after 4 in the afternoon

(lambda $0 e (and <n> <n>))
Constraining output structure: Seq2Tree

Flights from Dallas leaving after 4 in the afternoon

(lambda $0 e (and (> <n> 1600:ti) <n>))

[Dong and Lapata, 2016]
Constraining output structure: Seq2Tree

Flights from Dallas leaving after 4 in the afternoon

(lambda $0 e
  (and
    (> (departure_time $0) 1600:ti)
    <n>)))

[Dong and Lapata, 2016]
Flights from Dallas leaving after 4 in the afternoon

(lambda $0 e
  (and
   (> (departure_time $0) 1600:ti)
   (from $0 dallas:ci))))

[Dong and Lapata, 2016]
Constraining output structure: Seq2Tree

Flights from Dallas leaving after 4 in the afternoon

Need not explicitly model matching parentheses
Syntactically valid trees
Allows parent feeding
Semantically valid trees

[Dong and Lapata, 2016]
Empirical Comparison with Seq2Seq on GEO and ATIS

**GEO**

- Seq2Seq: 84.6
- Seq2Tree: 87.1

**ATIS**

- Seq2Seq: 84.2
- Seq2Tree: 84.6

[Dong and Lapata, 2016]
Sketch-Constrained Seq2Tree

- Decoding in two steps:
  1. Decoder 1: Rough sketch conditioned on encoder output
  2. Decoder 2: Finer output constrained by the sketch, conditioned on the outputs of decoder 1 and encoder

[Dong and Lapata, 2018]
Grammar-based Constrained Decoding
Constraining output structure and types

<table>
<thead>
<tr>
<th>Athlete</th>
<th>Nation</th>
<th>Olympics</th>
<th>Medals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gillis Grafström</td>
<td>Sweden (SWE)</td>
<td>1920–1932</td>
<td>4</td>
</tr>
<tr>
<td>Kim Yu-na</td>
<td>South Korea (KOR)</td>
<td>2010–2014</td>
<td>2</td>
</tr>
<tr>
<td>Patrick Chan</td>
<td>Canada (CAN)</td>
<td>2014</td>
<td>2</td>
</tr>
</tbody>
</table>

Basic Types:
Row (r); Cell (c) (kim_yu_na; 2014; …); Number (n); Date (d)

Complex Types:
Column (<c,r>): athlete; nation; olympics; medals
Binary row operations (<r,<r,r>>): and; or
Reverse column operation (<<c,r>,<r,c>>): reverse

[((reverse athlete)
 (and
 (nation south_korea)
 (year ((reverse date)
   (>= 2010-mm-dd)))))

[Krishnamurthy, Dasigi and Gardner, 2017]
Note on the notation of types

• Complex types
  • Example: column: \texttt{cell} \to \texttt{row} \quad \langle\texttt{c},\texttt{r}\rangle
  • Concrete example: \texttt{(nation south_korea)}

• Currying for functions with multiple arguments
  • Example: binary row operator: \texttt{row}, \texttt{row} \to \texttt{row}
    Rewritten as: \texttt{row} \to (\texttt{row} \to \texttt{row}) \quad \langle\texttt{r},\langle\texttt{r},\texttt{r}\rangle\rangle
  • Concrete example:
    \texttt{(and (nation south_korea) (medals 4))}

• Higher order functions
  • Example: reverse: (\texttt{cell} \to \texttt{row}) \to (\texttt{row} \to \texttt{cell}) \quad \langle\langle\texttt{c},\texttt{r}\rangle,\langle\texttt{r},\texttt{c}\rangle\rangle
  • Concrete example:
    \texttt{((reverse athlete) (and (nation south_korea) (medals 4))})
Constraining output structure and types

<table>
<thead>
<tr>
<th>Athlete</th>
<th>Nation</th>
<th>Olympics</th>
<th>Medals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gillis Grafström</td>
<td>Sweden (SWE)</td>
<td>1920–1932</td>
<td>4</td>
</tr>
<tr>
<td>Kim Yu-na</td>
<td>South Korea (KOR)</td>
<td>2010–2014</td>
<td>2</td>
</tr>
<tr>
<td>Patrick Chan</td>
<td>Canada (CAN)</td>
<td>2014</td>
<td>2</td>
</tr>
</tbody>
</table>

Basic Types:
Row (r); Cell (c) (kim_yu_na; 2014; …); Number (n); Date (d)

Complex Types:
Column (<c,r>): athlete; nation; olympics; medals
Binary row operations (<r,<r,r>>): and; or
Reverse column operation (<<c,r>,<r,c>>): reverse

((reverse athlete)
(and
(nation south_korea)
(year ((reverse date)
(>= 2010-mm-dd))))

[Krishnamurthy, Dasigi and Gardner, 2017]
Constraining output structure and types

$$((\text{reverse athlete}) \land \text{(nation south_korea)} \land \text{(year ((reverse date) \geq 2010-mm-dd))))$$
Constraining output structure and types

\(((\text{reverse athlete}) (\text{and} (\text{nation south_korea}) (\text{year} ((\text{reverse date}) (\geq 2010-mm-dd)))))\)

Generate the tree as a sequence of typed actions

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

Generated Actions
START $\rightarrow$ c

Logical Form
C

Non-terminal Stack

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

Generated Actions
START → c

Logical Form

Non-terminal Stack

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

Generated Actions
START → c
  c→(<r,c> r)

Logical Form
(<r,c> r)

Non-terminal Stack

[r, c]

[ Krishnamurthy, Dasigi and Gardner, 2017 ]
Which athlete was from South Korea after the year 2010?

Phrase Structure

Generated Actions

\[
\begin{align*}
\text{START} & \rightarrow c \\
c & \rightarrow (<r,c> \ r)
\end{align*}
\]

Logical Form

\[
(<r,c> \ r)
\]

Non-terminal Stack

\[
\begin{array}{c}
<r,c> \\
r
\end{array}
\]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

Generated Actions
START → c
  c→(<r,c> r)
  <r,c>→(<<c,r>,<r,c>>, <c,r>)

Logical Form
((<<c,r>,<r,c>>, <c,r>) r)

Non-terminal Stack

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

**Generated Actions**

<table>
<thead>
<tr>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>(r,c) r</td>
</tr>
<tr>
<td>r,c</td>
<td>(c,r) r,c</td>
</tr>
</tbody>
</table>

**Logical Form**

\[
((<<c,r>,<r,c>>, <c,r>) r)
\]

**Non-terminal Stack**

![Diagram of non-terminal stack]

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

Generated Actions
START → c
  c→(<r,c> r)
  <r,c>→(<<c,r>,<r,c>> <c,r>)
  <<c,r>,<r,c>> → reverse

Logical Form
((reverse <c,r>) r)

Non-terminal Stack

[c,r]
r

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

Generated Actions
START → c
c → (<r, c> r)
<r, c> → (<<c, r>, <r, c>> <c, r>)
<<c, r>, <r, c>> → reverse
<c, r> → athlete

Logical Form
((reverse athlete) r)

Non-terminal Stack

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

Generated Actions

START $\rightarrow$ c

c$\rightarrow$ (<r,c> r)

<r,c> $\rightarrow$ (<c,r>,<r,c> <c,r>)

<c,r>,<r,c> $\rightarrow$ reverse

<c,r> $\rightarrow$ athlete

r $\rightarrow$ (<r,<r,r>> r r)

<r,<r,r>> $\rightarrow$ and

Logical Form

((reverse athlete)
(and r r))

Non-terminal Stack

r

r

[r, Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

**Generated Actions**
- START $\rightarrow$ c
- c $\rightarrow$ (r,c) r
- r $\rightarrow$ (c,r) r
- c $\rightarrow$ nation
- r $\rightarrow$ reverse
- c,r $\rightarrow$ athlete
- <<c,r>,<r,c>> $\rightarrow$ and

**Logical Form**
$$((\text{reverse} \ \text{athlete}) \ \text{and} \ (\text{nation} \ c \ r))$$

**Non-terminal Stack**

[ Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

Generated Actions
START → c
  c → (r, c) r
  (r, c) → (<<c, r>, <r, c>> c, r)
  <<c, r>, <r, c>> → reverse
  c, r → athlete
  r → (r, <r, r>> r r)
  <r, <r, r>> → and
  r → (c, r) c
  c, r → nation
  c → south_korea

Logical Form
((reverse athlete)
 (and (nation south_korea) r))

Non-terminal Stack

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Which athlete was from South Korea after the year 2010?

**Generated Actions**
- START $\rightarrow$ c
- c $\rightarrow$ (r,c) r
- r $\rightarrow$ (r,<r,r>> r r)
- c $\rightarrow$ south_korea
- r $\rightarrow$ (c,r) c
- c $\rightarrow$ year
- (<<c,r>,<r,c>>,r,c) $\rightarrow$ reverse
- (<<c,r>,<r,c>>,r,c) $\rightarrow$ and

**Logical Form**

$((\text{reverse athlete})
\ (\text{and (nation south_korea)}\ (\text{year c})))$
Which athlete was from South Korea after the year 2010?

**Generated Actions**

<table>
<thead>
<tr>
<th>Action</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>c</td>
</tr>
<tr>
<td>c→(&lt;r,c&gt; r)</td>
<td></td>
</tr>
<tr>
<td>&lt;r,c&gt;→(&lt;c,r&gt;,&lt;r,c&gt;&gt; &lt;c,r&gt;)</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;c,r&gt;,&lt;r,c&gt;&gt;→reverse</td>
<td></td>
</tr>
<tr>
<td>&lt;c,r&gt;→athlete</td>
<td></td>
</tr>
<tr>
<td>r→(&lt;r,&lt;r,r&gt;&gt; r r)</td>
<td></td>
</tr>
<tr>
<td>&lt;r,&lt;r,r&gt;&gt;→and</td>
<td></td>
</tr>
<tr>
<td>r→(&lt;c,r&gt; c)</td>
<td></td>
</tr>
<tr>
<td>&lt;c,r&gt;→nation</td>
<td></td>
</tr>
<tr>
<td>c→south_korea</td>
<td></td>
</tr>
<tr>
<td>r→(&lt;c,r&gt; c)</td>
<td></td>
</tr>
<tr>
<td>&lt;c,r&gt;→year</td>
<td></td>
</tr>
<tr>
<td>c→(&lt;d,c&gt; d)</td>
<td></td>
</tr>
<tr>
<td>&lt;d,c&gt;→(&lt;c,d&gt;,&lt;d,c&gt;&gt; &lt;c,d&gt;)</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;c,d&gt;,&lt;d,c&gt;&gt;→reverse</td>
<td></td>
</tr>
</tbody>
</table>

**Logical Form**

\[
((\text{reverse} \ \text{athlete})
\land (\text{nation} \ \text{south\_korea})
\land (\text{year} ((\text{reverse} \ \text{date})
\geq \text{2010-mm-dd})))
\]

[Krishnamurthy, Dasigi and Gardner, 2017]
Grammar-Constrained Decoding

Generated Actions

```
START → c
  c→(r,c) r
  <r,c>→(<<c,r>,<r,c>><c,r>)
  <<c,r>,<r,c>>→reverse
  <c,r>→nation
  r→(c,r)
  r→(<r,<r,r>>) r
  <r,<r,r>>→and
  r→(<c,r>)
  c→south
  r→(<c,r>)
  <c,r>→year
  c→(<d,c>d)
  <d,c>→(<<c,d>,<d,c>><c,d>)
  <<c,d>,<d,c>>→reverse
```

Logical Form

```
((reverse athlete)
 (and (nation south_korea)
   (year ((reverse date)
     (>= 2010.mm.dd)))))
```

Need not explicitly model matching parentheses
Syntactically valid trees
Semantically valid trees

Non-terminal Stack

[Krishnamurthy, Dasigi and Gardner, 2017]
Empirical Comparison with Seq2Seq and Seq2Tree on WikiTableQuestions

[Krishnamurthy, Dasigi and Gardner, 2017]
Summary

• Constraining output forces decoder to generate only valid outputs

• Impose hard constraints instead of hoping the model would learn them

• Various hard constraints depending on output space
  • Token-level decoding (Seq2Tree, sketches, etc)
  • Grammar-based constraints