

EE-608: Deep Learning for Natural Language Processing: Syntactic Parsing

James Henderson



DLNLP, Lecture 6

Outline

Syntactic Structure

Syntactic Dependency Parsing

Transition-based Neural Network Parsing

Graph-based Neural Network Parsing

Iterative-Refinement Neural Network Parsing

Outline

Syntactic Structure

Syntactic Dependency Parsing

Transition-based Neural Network Parsing

Graph-based Neural Network Parsing

Iterative-Refinement Neural Network Parsing

1. Two views of linguistic structure: Constituency = phrase structure grammar = context-free grammars (CFGs)

Phrase structure organizes words into nested constituents

Starting unit: words

the, cat, cuddly, by, door
Det N Adj P N

a
the
this
that
every

Words combine into phrases

NP [the cuddly cat,] [by [the door] NP] PP

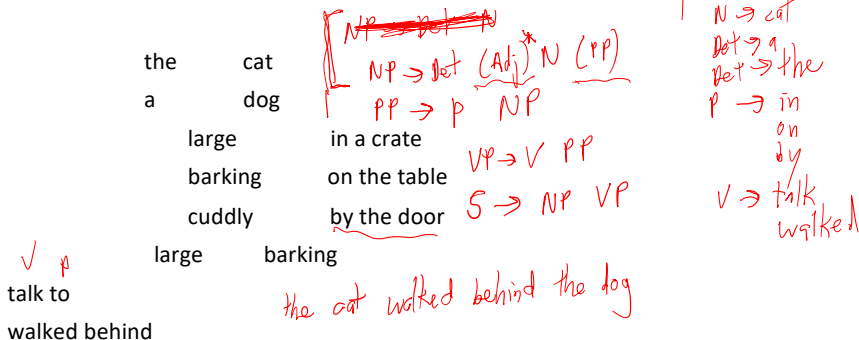
Phrases can combine into bigger phrases

the cuddly cat by the door

NP PP

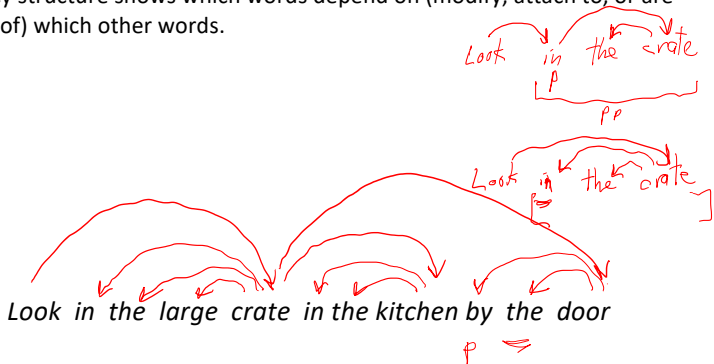
Two views of linguistic structure: Constituency = phrase structure grammar = context-free grammars (CFGs)

Phrase structure organizes words into nested constituents.



Two views of linguistic structure: Dependency structure

- Dependency structure shows which words depend on (modify, attach to, or are arguments of) which other words.



Why do we need sentence structure?

Humans communicate complex ideas by composing words together into bigger units to convey complex meanings

Listeners need to work out what modifies [attaches to] what

A model needs to understand sentence structure in order to be able to interpret language correctly

Prepositional phrase attachment ambiguity

The image shows a screenshot of a BBC News article. The headline is "San Jose cops kill man with knife". Handwritten red annotations illustrate the ambiguity of the prepositional phrase "with knife":

- A bracket labeled "subj" (subject) spans "San Jose cops".
- A bracket labeled "obj" (object) spans "kill man".
- A bracket labeled "obl" (oblique) spans "with knife".
- A red line connects the "obl" bracket to the "kill" verb, indicating one possible attachment (the knife used for killing).
- Another red line connects the "obl" bracket to the "man" noun, indicating another possible attachment (the man who has a knife).

The page includes the BBC logo, a "Sign in" button, and navigation links for News, Sport, Weather, Shop, Reel, and Travel. Below the headline is a red banner with the word "NEWS" and a list of categories: Home, Video, World, US & Canada, UK, Business, Tech, Science, and Stories. The "Science" category is highlighted, and the sub-header "Science & Environment" is visible.

Scientists count whales from space

By Jonathan Amos
BBC Science Correspondent

Prepositional phrase attachment ambiguity

Scientists count whales from space



Scientists count whales from space



PP attachment ambiguities multiply

- A key parsing decision is how we 'attach' various constituents
 - PPs, adverbial or participial phrases, infinitives, coordinations,

The board approved [its acquisition] [by Royal Trustco Ltd.]
[of Toronto]
[for \$27 a share]
[at its monthly meeting].

- Catalan numbers: $C_n = (2n)! / ((n+1)!n!)$
 - An exponentially growing series, which arises in many tree-like contexts:
 - E.g., the number of possible triangulations of a polygon with $n+2$ sides
- Turns up in triangulation of probabilistic graphical models (CS228)....

Coordination scope ambiguity

coord

[Shuttle veteran] and [longtime NASA executive Fred Gregory] appointed to board

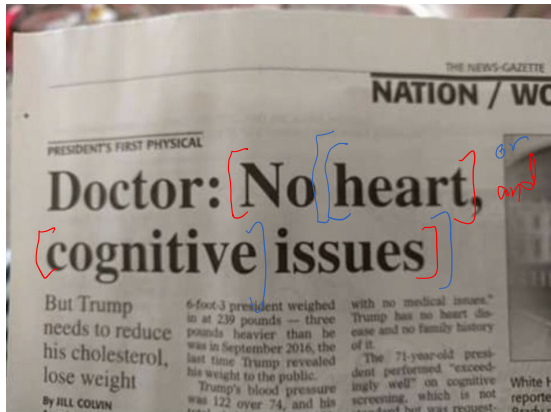
2 people

nmod

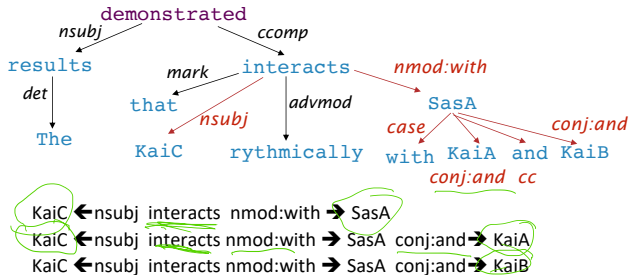
[Shuttle veteran and longtime NASA executive] Fred Gregory appointed to board

1 person

Coordination scope ambiguity



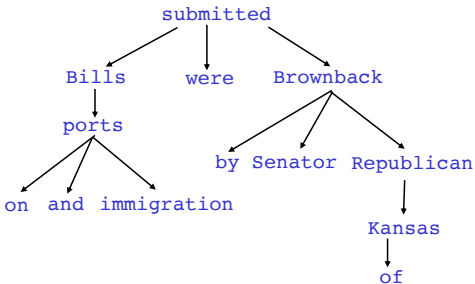
Dependency paths help extract semantic interpretation – simple practical example: extracting protein-protein interaction



[Erkan et al. EMNLP 07, Fundel et al. 2007, etc.]

2. Dependency Grammar and Dependency Structure

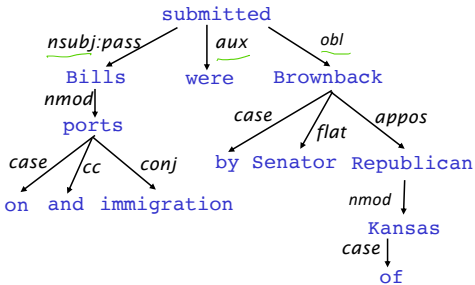
Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called **dependencies**



Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called **dependencies**

The arrows are commonly **typed** with the name of grammatical relations (subject, prepositional object, apposition, etc.)

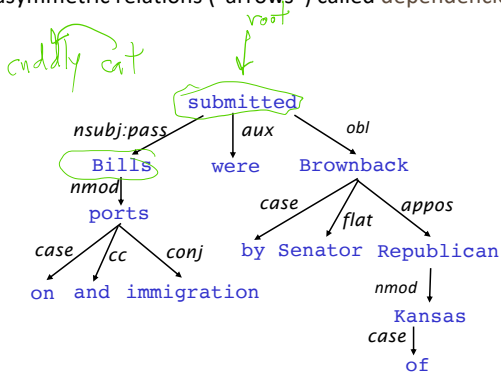


Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called dependencies

An arrow connects a **head** (governor, superior, regent) with a **dependent** (modifier, inferior, subordinate)

Usually, dependencies form a tree (a connected, acyclic, single-root graph)



Pāṇini's grammar (c. 5th century BCE)

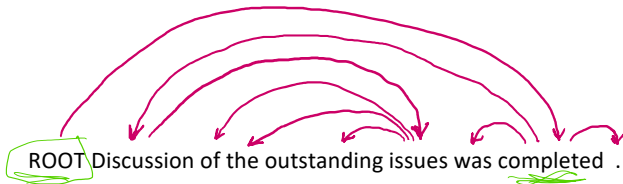


Gallery: <http://wellcomeimages.org/indexplus/image/L0032691.html>
CC BY 4.0 File: Birch bark MS from Kashmir of the Rupavatra Wellcome L0032691.jpg

Dependency Grammar/Parsing History

- The idea of dependency structure goes back a long way
 - To Pāṇini's grammar (c. 5th century BCE)
 - Basic approach of 1st millennium Arabic grammarians
- Constituency/context-free grammar is a new-fangled invention
 - 20th century invention (R.S. Wells, 1947; then Chomsky 1953, etc.)
- Modern dependency work is often sourced to Lucien Tesnière (1959)
 - Was dominant approach in "East" in 20th Century (Russia, China, ...)
 - Good for free-er word order, inflected languages like Russian (or Latin!)
- Used in some of the earliest parsers in NLP, even in the US:
 - David Hays, one of the founders of U.S. computational linguistics, built early (first?) dependency parser (Hays 1962) and published on dependency grammar in *Language*

Dependency Grammar and Dependency Structure



- Some people draw the arrows one way; some the other way!
 - Tesnière had them point from head to dependent – we follow that convention
- We usually add a fake ROOT so every word is a dependent of precisely 1 other node

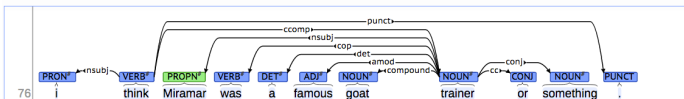
The rise of annotated data & Universal Dependencies treebanks

Brown corpus (1967; PoS tagged 1979); Lancaster-IBM Treebank (starting late 1980s);

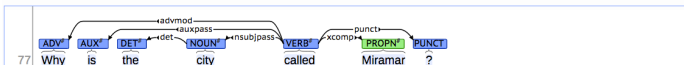
Marcus et al. 1993, The Penn Treebank, *Computational Linguistics*;

Universal Dependencies: <http://universaldependencies.org/>

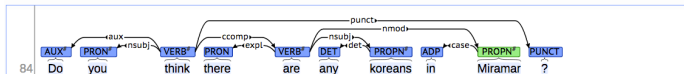
[context] [conllu]



[context] [conllu]



[context] [conllu]



The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

But a treebank gives us many things

- Reusability of the labor
 - Many parsers, part-of-speech taggers, etc. can be built on it
 - Valuable resource for linguistics
- Broad coverage, not just a few intuitions
- Frequencies and distributional information
- A way to evaluate NLP systems

Dependency Conditioning Preferences

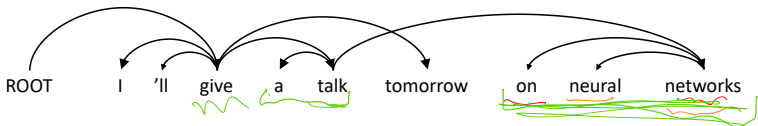
What are the sources of information for dependency parsing?

1. Bilexical affinities The dependency [discussion → issues] is plausible
2. Dependency distance Most dependencies are between nearby words
3. Intervening material Dependencies rarely span intervening verbs or punctuation
4. Valency of heads How many dependents on which side are usual for a head?



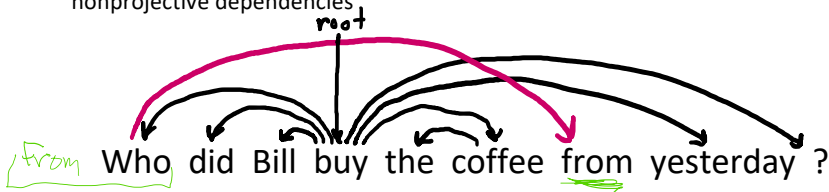
Dependency Parsing

- A sentence is parsed by choosing for each word what other word (including ROOT) it is a dependent of
- Usually some constraints:
 - Only one word is a dependent of ROOT
 - Don't want cycles $A \rightarrow B, B \rightarrow A$
- This makes the dependencies a tree
- Final issue is whether arrows can cross (be **non-projective**) or not



Projectivity

- Definition of a **projective parse**: There are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words
- Dependencies corresponding to a CFG tree must be **projective**
 - I.e., by forming dependencies by taking 1 child of each category as head
- Most syntactic structure is projective like this, but dependency theory normally does allow non-projective structures to account for displaced constituents
 - You can't easily get the semantics of certain constructions right without these nonprojective dependencies



3. Methods of Dependency Parsing

1. Dynamic programming

Eisner (1996) gives a clever algorithm with complexity $O(n^3)$, by producing parse items with heads at the ends rather than in the middle

2. Graph algorithms

You create a Minimum Spanning Tree for a sentence

McDonald et al.'s (2005) MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)

Neural graph-based parser: Dozat and Manning (2017) et seq. – very successful!

3. Constraint Satisfaction

Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

4. “Transition-based parsing” or “deterministic dependency parsing”

Greedy choice of attachments guided by good machine learning classifiers

E.g., MaltParser (Nivre et al. 2008). Has proven highly effective.

Summary of Syntactic Structure

- ▶ Syntactic structure is a fundamental property of all natural languages
- ▶ Syntax is an interface between the meanings of words and the meaning of text
- ▶ Two modern traditions for syntactic structure:
 - ▶ Constituency structure:
group words into phrases, recursively
 - ▶ Dependency structure:
link words by their roles, in a tree

Outline

Syntactic Structure

Syntactic Dependency Parsing

Transition-based Neural Network Parsing

Graph-based Neural Network Parsing

Iterative-Refinement Neural Network Parsing

Greedy transition-based parsing [Nivre 2003]



- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom-up actions
 - Roughly like “shift” or “reduce” in a shift-reduce parser, but the “reduce” actions are specialized to create dependencies with head on left or right
- The parser has:
 - a stack σ , written with top to the right
 - which starts with the ROOT symbol
 - a buffer β , written with top to the left
 - which starts with the input sentence
 - a set of dependency arcs A
 - which starts off empty
 - a set of actions

Basic transition-based dependency parser

Start: $\sigma = [\text{ROOT}]$, $\beta = w_1, \dots, w_n$, $A = \emptyset$

1. **Shift** $\sigma, w_i | \beta, A \rightarrow \sigma | w_i, \beta, A$

2. **Left-Arc_r** $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_j, \beta, A \cup \{r(w_j, w_i)\}$

3. **Right-Arc_r** $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_i, \beta, A \cup \{r(w_i, w_j)\}$

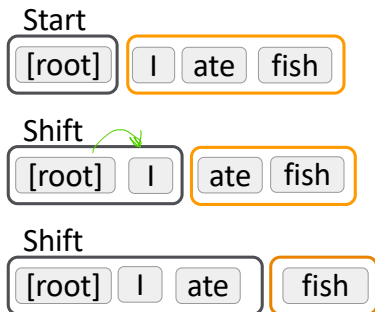
Finish: $\sigma = [w]$, $\beta = \emptyset$

↑
relation

Arc-standard transition-based parser

(there are other transition schemes ...)

Analysis of “I ate fish”



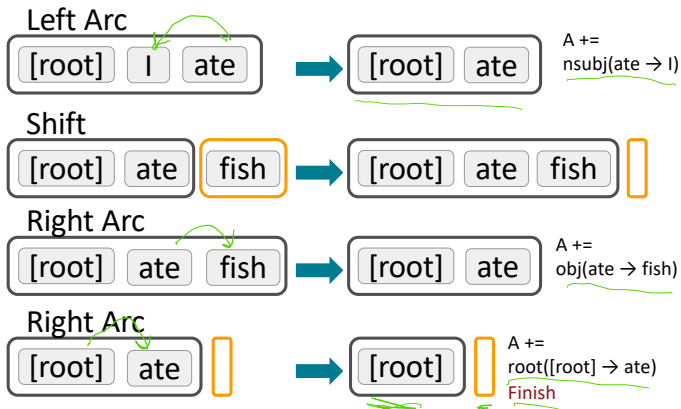
Start: $\sigma = [\text{ROOT}]$, $\beta = w_1, \dots, w_n$, $A = \emptyset$

1. Shift $\sigma, w_i | \beta, A \rightarrow \sigma | w_i, \beta, A$
2. Left-Arc, $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_j, \beta, AU\{r(w_i, w_j)\}$
3. Right-Arc, $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_i, \beta, AU\{r(w_i, w_j)\}$

Finish: $\sigma = [w]$, $\beta = \emptyset$

Arc-standard transition-based parser

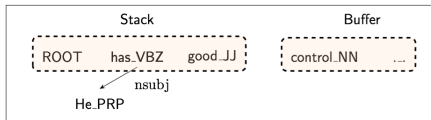
Analysis of “I ate fish”



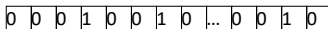
MaltParser [Nivre and Hall 2005]

- We have left to explain how we choose the next action 🤖
 - Answer: Stand back, I know machine learning!
- Each action is predicted by a discriminative classifier (e.g., softmax classifier) over each legal move
 - Max of 3 untyped choices; max of $|R| \times 2 + 1$ when typed
 - Features: top of stack word, POS; first in buffer word, POS; etc.
- There is NO search (in the simplest form)
 - But you can profitably do a beam search if you wish (slower but better): You keep k good parse prefixes at each time step
- The model's accuracy is *fractionally* below the state of the art in dependency parsing, but
- It provides **very fast linear time parsing**, with high accuracy – great for parsing the web

Conventional Feature Representation



binary, sparse
dim = $10^6 - 10^7$



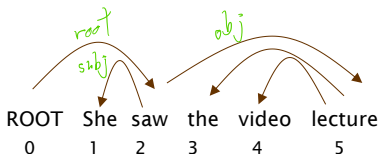
Feature templates: usually a combination of 1–3 elements from the configuration

Indicator features

1/0 {

- $s1.w = \text{good} \wedge s1.t = \text{JJ}$
- $s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$
- $lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$
- $lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$

Evaluation of Dependency Parsing: (labeled) dependency accuracy



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

$$\text{UAS} = 4 / 5 = 80\%$$

$$\text{LAS} = 2 / 5 = 40\%$$

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed

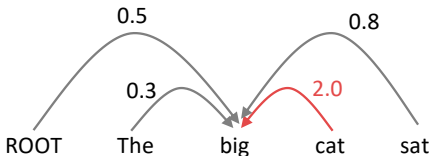
1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Handling non-projectivity

- The arc-standard algorithm we presented only builds projective dependency trees
- Possible directions to head:
 1. Just declare defeat on nonprojective arcs 🙅
 2. Use dependency formalism which only has projective representations
 - A CFG only allows projective structures; you promote head of projectivity violations
 3. Use a postprocessor to a projective dependency parsing algorithm to identify and resolve nonprojective links
 4. Add extra transitions that can model at least most non-projective structures (e.g., add an extra SWAP transition, cf. bubble sort)
 5. Move to a parsing mechanism that does not use or require any constraints on projectivity (e.g., the graph-based MSTParser or Dozat and Manning (2017))
 6. Add dependency edges which projectivise local nonprojective structures

Graph-based dependency parsers

- Compute a score for every possible dependency for each word
 - Doing this well requires good “contextual” representations of each word token, which we will develop in coming lectures
 - And repeat the same process for each other word



e.g., picking the head for “big”

Graph-based dependency parsers

- ▶ Given $O(n^2)$ scores $\text{score}(i, j|x)$ (for first-order algorithms), find:

$$\operatorname{argmax}_{\hat{y} \in Y(x)} \sum_i \text{score}(i, \hat{y}_i|x)$$

- ▶ where $Y(x)$ is the set of trees over x
- ▶ using an algorithm like Minimum Spanning Tree ($O(n^3)$)

Summary of Syntactic Dependency Parsing

- ▶ Syntactic parsing is a benchmark task for ML models of structured prediction
- ▶ Dependency parsing finds the syntactic dependencies for a sentence
- ▶ Two ways to decompose the score of a parse:
 - ▶ Graph-based:
estimate scores independently, then choose the optimal combination with dynamic programming (MST)
 - ▶ Transition-based:
estimate scores conditioned on history, and search greedily (or with beam search)
- ▶ Transition-based parsing tends to be faster, but graph-based is more accurate

Outline

Syntactic Structure

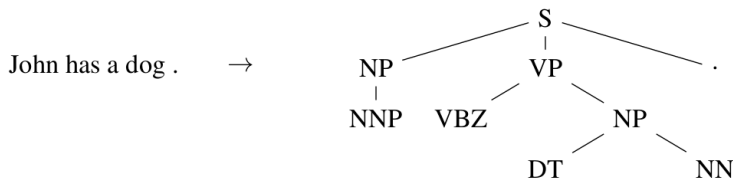
Syntactic Dependency Parsing

Transition-based Neural Network Parsing

Graph-based Neural Network Parsing

Iterative-Refinement Neural Network Parsing

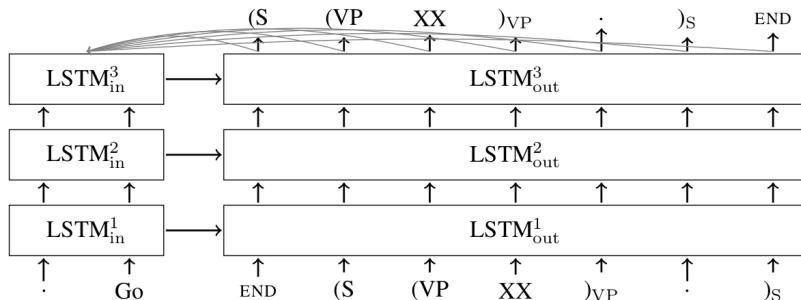
Sequence-to-Sequence Parsing (Vinyals et al. NIPS 2015)



John has a dog . $\rightarrow$ (S (NP NNP)_{NP} (VP VBZ (NP DT NN)_{NP})_{VP} .)_S

- ▶ Parses can be represented as bracketed sequences
- ▶ Neural networks can generate sequences
- ▶ Why not do parsing like machine translation?

Sequence-to-Sequence Parsing (Vinyals et al. NIPS 2015)



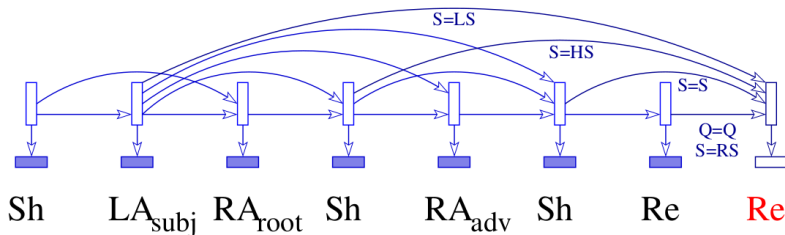
- ▶ Use a 2015-SoA neural machine translation system
- ▶ stacked LSTM encoder, stacked LSTM decoder with attention to encoder
- ▶ Train it to generate valid bracketed sequences

Sequence-to-Sequence Parsing (Vinyals et al. NIPS 2015)

Parser	Training Set	WSJ 22	WSJ 23
baseline LSTM+D	WSJ only	< 70	< 70
LSTM+A+D	WSJ only	88.7	88.3
LSTM+A+D ensemble	WSJ only	90.7	90.5
baseline LSTM	BerkeleyParser corpus	91.0	90.5
LSTM+A	high-confidence corpus	92.8	92.1
Petrov et al. (2006) [12]	WSJ only	91.1	90.4
Zhu et al. (2013) [13]	WSJ only	N/A	90.4
Petrov et al. (2010) ensemble [14]	WSJ only	92.5	91.8
Zhu et al. (2013) [13]	semi-supervised	N/A	91.3
Huang & Harper (2009) [15]	semi-supervised	N/A	91.3
McClosky et al. (2006) [16]	semi-supervised	92.4	92.1

- ▶ Given a sentence, generate a bracketed string
- ▶ Convert the string into a parse tree
- ▶ Measure accuracy of the resulting parses
- ▶ (Optionally, constrain predictions to valid sequences at test time)

Output-Dependent Model Structure



- ▶ Need a bias towards learning correlations which are structurally local
- ▶ Structure is determined by the parser output
- ▶ $\Rightarrow$ Model structure can be incrementally constructed based on output structure

Neural Network Augmentations of Symbolic Models

(Henderson, NAACL 2003; Henderson and Titov, JMLR 2011)

- ▶ Recurrent neural networks (RNN)
 - ▶ powerful probability estimators
 - ▶ induced history representations
 - ▶ inductive bias in model structure
- ▶ Symbolic models
 - ▶ unbounded generalisation
 - ▶ domain-appropriate structural locality bias
 - ▶ efficient decoding strategies
- ▶ Incremental Neural Networks (INN) (a.k.a. SSN, ISBN)
 - ▶ RNN estimates derivation decision probabilities
 - ▶ with incrementally-specified model structure
 - ▶ and efficient beam search decoding

This is the ground breaking work on integrating decoding and structured prediction with neural networks.

Neural Network Augmentations of Symbolic Models

(Henderson, NAACL 2003; Henderson and Titov, JMLR 2011)

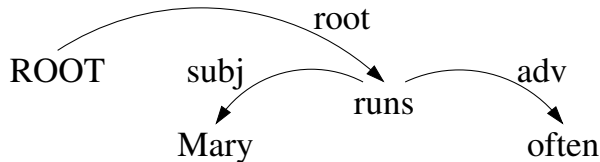
- ▶ Recurrent neural networks (RNN)
 - ▶ powerful probability estimators
 - ▶ induced history representations
 - ▶ inductive bias in model structure
- ▶ Symbolic models
 - ▶ unbounded generalisation
 - ▶ domain-appropriate structural locality bias
 - ▶ efficient decoding strategies
- ▶ Incremental Neural Networks (INN) (a.k.a. SSN, ISBN)
 - ▶ RNN estimates derivation decision probabilities
 - ▶ with incrementally-specified model structure
 - ▶ and efficient beam search decoding

This is the ground breaking work on integrating decoding and structured prediction with neural networks.

Syntactic parsing with INNs

- ▶ Constituency parser with left-corner derivations (Henderson, NAACL 2003)
- ▶ Dependency parser with arc-eager derivations (Titov and Henderson, IWPT 2007)
- ▶ Faster beam search with discriminative training (Yazdani and Henderson, CoNLL 2015)

The worlds best parser in 2017 (from Google) was based on the design from (Titov and Henderson, IWPT 2007).



INN dependency parsing example

(Titov and Henderson, IWPT 2007)

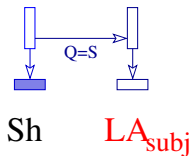
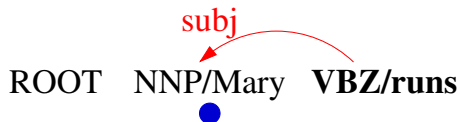
ROOT **NNP/Mary** **VBZ/runs**



Sh

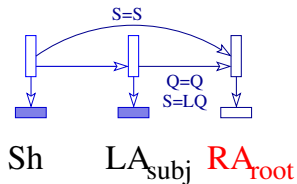
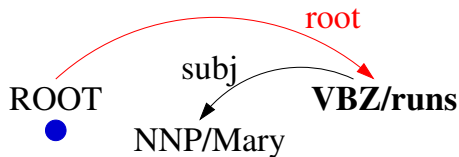
INN dependency parsing example

(Titov and Henderson, IWPT 2007)



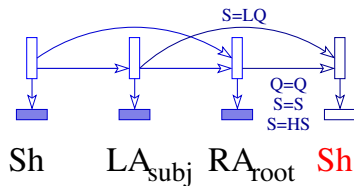
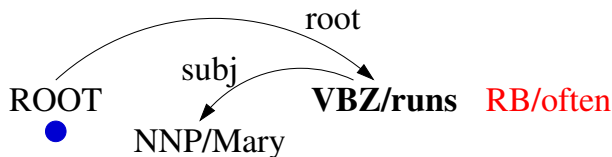
INN dependency parsing example

(Titov and Henderson, IWPT 2007)



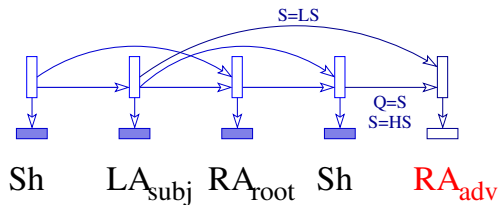
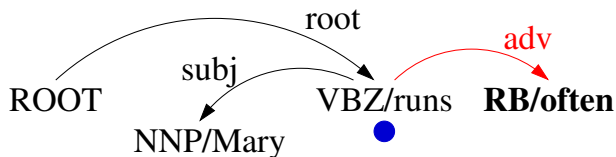
INN dependency parsing example

(Titov and Henderson, IWPT 2007)



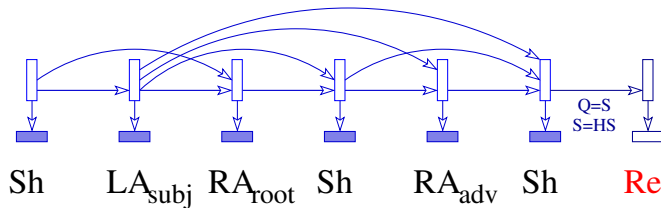
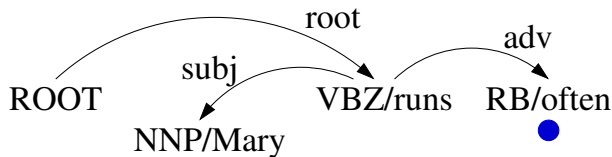
INN dependency parsing example

(Titov and Henderson, IWPT 2007)



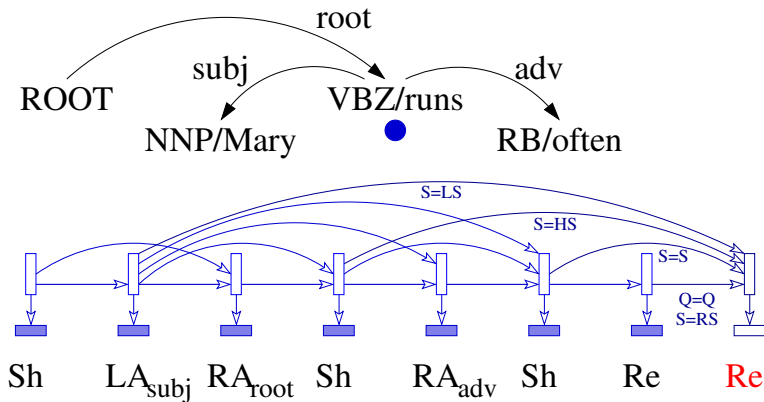
INN dependency parsing example

(Titov and Henderson, IWPT 2007)



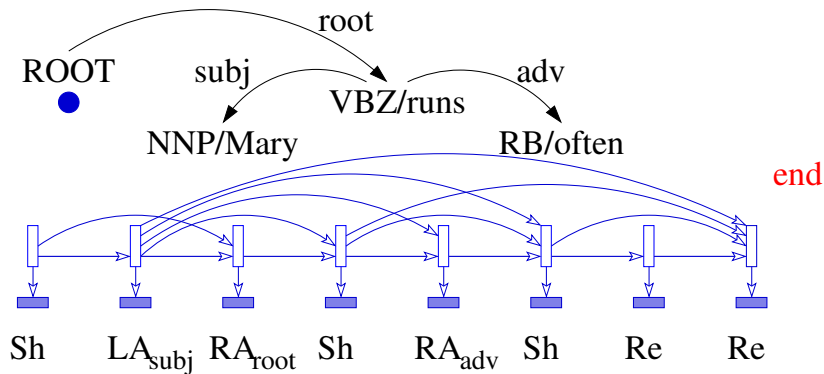
INN dependency parsing example

(Titov and Henderson, IWPT 2007)



INN dependency parsing example

(Titov and Henderson, IWPT 2007)



Discriminative modelling of beam search

(Yazdani and Henderson, CoNLL 2015)

Replace the word predictions of the generative model with a ***Correctness Probability*** output

- ▶ Trained discriminatively to **distinguish between correct and incorrect parse prefixes**
- ▶ Allows beam search with pruning just at *Shift* actions

Better feature parametrisation:

- ▶ Low-frequency word-role pair features are factorised into word vectors multiplied by role matrices

Discriminative modelling of beam search

(Yazdani and Henderson, CoNLL 2015)

Replace the word predictions of the generative model with a ***Correctness Probability*** output

- ▶ Trained discriminatively to **distinguish between correct and incorrect parse prefixes**
- ▶ Allows beam search with pruning just at *Shift* actions

Better feature parametrisation:

- ▶ Low-frequency word-role pair features are factorised into word vectors multiplied by role matrices

Results on English

better than (Yazdani and Henderson, CoNLL 2015)

CoNLL 2009 English syntactic dependencies:

Model	LAA	wrds/sec
<i>MALT</i> _{AE}	86.0	7549
Chen&Manning	86.5	9589
MST	87.1	290
Generative INN , beam 1	77.8	1122
Generative INN , beam 10	87.7	107
Generative INN , large beam	88.7	NA
DINN , beam 1	89.0	4890
DINN , beam 10	89.8	544

- Discriminative parser with a beam of 1 is **50 times faster** than the generative model with beam 10, with higher accuracy.

Multilingual Results

Diverse CoNLL 2009 languages, syntactic dependencies:

Model	German	Spanish	Czech
Chen&Manning	81.9	81.5	58.5
MALT	80.7	82.4	67.3
MST	84.1	82.7	73.4
DINN	86.0	85.4	77.5

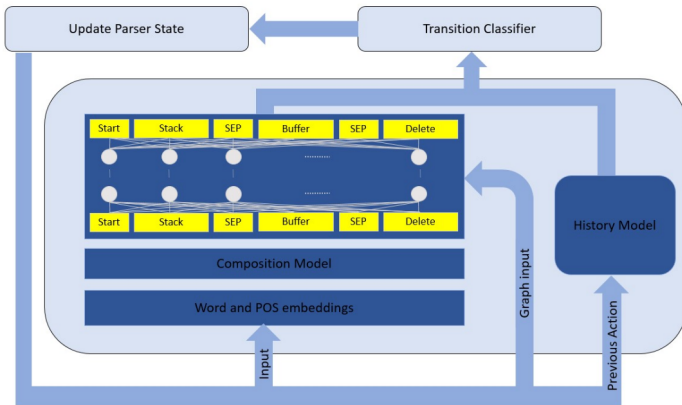
- Better at generalising to new languages without feature engineering.

Stack-LSTM Dependency Parser

“Transition-Based Dependency Parsing with Stack Long Short-Term Memory”, Dyer, Ballesteros, Ling, Matthews and Smith. ACL 2015.

- ▶ Model structure is defined in terms of the parser's data structures
- ▶ General mechanism for applying LSTMs to stack data structures

Transformer Transition-Based Dependency Parsing



“Graph-to-Graph Transformer for Dependency Parsing”,
Mohammadshahi and Henderson, 2020

- ▶ Transformer computes token embeddings for parser state
- ▶ Additional inputs to reflect parser state and history
- ▶ Parser actions predicted from relevant tokens (top of stack)

- Motivation
- Graph-to-Graph Transformer
 - Transition-based Dependency Parsing
 - Graph-based Iterative Refinement
 - Semantic Role Labelling
- Compression of Massively Multilingual Models
 - Impacts on different biases
 - Distillation of these models for low-resource languages
- Conclusion and Future Directions

Graph-to-Graph Transformer for Transition-based Dependency Parsing



■ PhD Oral Exam

Slide from Alireza Mohammadshahi

Findings of EMNLP 2020

Authors:

Alireza Mohammadshahi
James Henderson

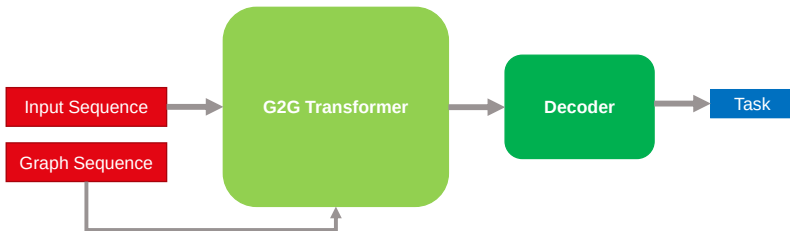
Previous Work

- Add a Graph Encoder on top of sequence encoder (Ji et al,2019)



- Add a hard bias toward graph structure (strubell et al,2018)
- Recursively use neural networks (Dyer et al,2015)

Graph-to-Graph Transformer



Graph-to-Graph Transformer

- ❖ Input arbitrary graph in addition to input sequence
- ❖ Output a graph for the downstream task
- ❖ Combines both sequence encoder and graph encoder into one general encoder
- ❖ Add a soft bias toward attention heads (no hard coding)

Graph-to-Graph Transformer

Compute output representations of input sequence by stack of multi-head self-attention layers

Each Layer:

- ❖ Multi-head attention layer
- ❖ Feed-forward position-wise NN

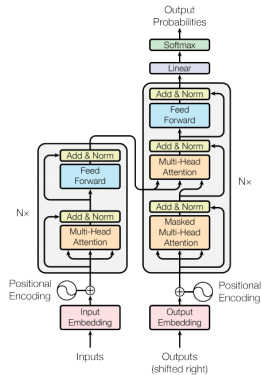


Photo by Vaswani et al, 2017

Graph-to-Graph Transformer

We have input sequence X , Transformer finds Output representation Z :

$$z_i = \sum_j \alpha_{ij} (x_j W^v)$$

Attention weights are calculated as:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}, \quad e_{ij} = \frac{(x_i W^Q)(x_j W^K)}{\sqrt{d}}$$

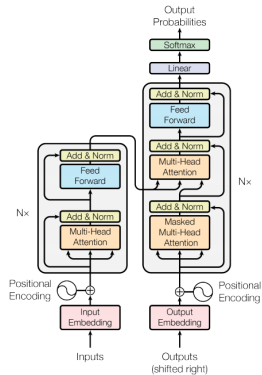


Photo by Vaswani et al, 2017

Graph-to-Graph Transformer

To input a graph, we modify equations of original Transformer:

$$z_i = \sum_j \alpha_{ij} (x_j W^v + p_{ij} W_L^2)$$
$$e_{ij} = \frac{(x_i W^Q)(x_j W^K + p_{ij} W_L^1)}{\sqrt{d}}$$

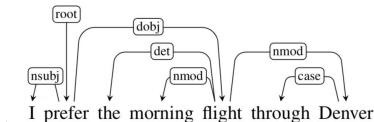
p_{ij} is the relation between token x_i and x_j

Graph-to-Graph Transformer

- Matrix P can be any input graph
- **Soft Bias.** Each attention head can easily learn to attend only to positions in a given relation, but it can also learn other structures in combination with other input
- Output value representation can have both token-level and graph-level information
- Can be applied to any NLP tasks which require to input a graph or produce a graph over the same nodes

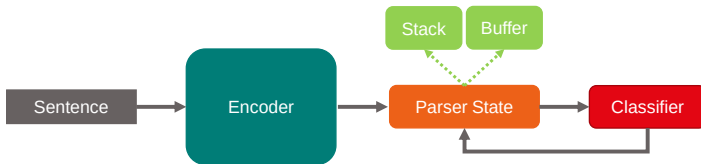
Dependency Parsing

- Extracting a dependency parse of a sentence that represents its grammatical structure
- Defines the relationships between “head” words and words, which modify those heads.



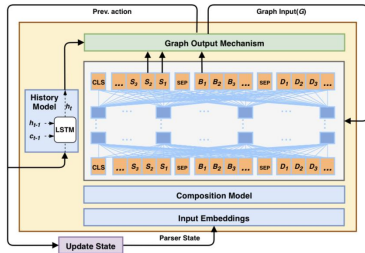
Transition-based Models

- Iteratively build the dependency graph by predicting a new transition at each step
- Transition classifier predicts the new action based on parser state

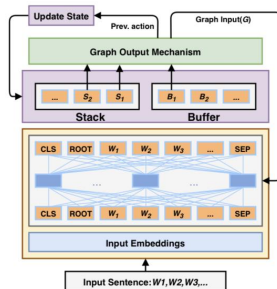


Baselines + G2G-Tr

StateTr+G2G-Tr



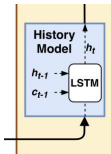
SentTr+G2G-Tr



Slide from Alireza Mohammadshahi

Baselines + G2G-Tr

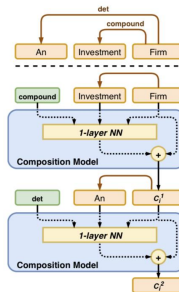
History Model



$$h^t, c^t = \text{LSTM}((h^{t-1}, c^{t-1}), a^t + l^t)$$

a^t and l^t are predicted action and dependency label.

Composition Model



Results

We evaluated our model on:

- English WSJ Penn Treebank (Marcus et al,1993)
- 13 Languages of UD Treebanks (Nivre et al,2018)

Results

- G2G-Tr integration
- BERT pre-training
- Graph output mechanism
- Replacement with composition model
- UD Benchmark

Language	Kulmizev et al. (2019)	BERT StateTr+G2GTr	BERT SentTr+G2GTr
Arabic	81.9	82.63	83.65
Basque	77.9	74.03	83.88
Chinese	83.7	85.91	87.49
English	87.8	89.21	90.35
Finnish	85.1	80.87	89.47
Hebrew	85.5	87.0	88.75
Hindi	89.5	93.13	93.12
Italian	92.0	92.6	93.99
Japanese	92.9	95.25	95.51
Korean	83.7	80.13	87.09
Russian	91.5	92.34	93.30
Swedish	87.6	88.36	90.40
Turkish	64.2	56.87	67.77
Average	84.87	84.48	88.06

	Dev Set		Test Set	
	UAS	LAS	UAS	LAS
Transition-based:				
Dyer et al. (2015)			93.10	90.90
Weiss et al. (2015)			94.26	91.42
Cross and Huang (2016)			93.42	91.36
Ballesteros et al. (2016)			93.56	92.41
Andor et al. (2016)			94.61	92.79
Kiperavasser and Goldberg (2016)			93.90	91.90
Yang et al. (2017)			94.18	92.26
Seq2Seq-based:				
Zhang et al. (2017)			93.71	91.60
Li et al. (2018)			94.11	92.08
StateTr	91.94	89.07	92.32	89.69
StateTr+G2GTr	92.53	90.16	93.07	91.08
BERT StateTr	94.66	91.94	95.18	92.73
BERT StateCLSTr	93.62	90.95	94.31	91.85
BERT StateTr+G2GTr	94.96	92.88	95.58	93.74
BERT StateTr+G2CLSTr	94.29	92.13	94.83	92.96
BERT StateTr+G2GTr+C	94.41	92.25	94.89	92.93
BERT SentTr	95.34	93.29	95.65	93.85
BERT SentTr+G2GTr	95.66	93.60	96.06	94.26
BERT SentTr+G2GTr-7 layer	95.78	93.74	96.11	94.33

Conclusion

- Propose Graph-to-Graph Transformer Architecture
- Applied to transition-based dependency parsing
- Compatible with BERT pre-training
- Outperforms previous work in the transition-based parsing

Outline

Syntactic Structure

Syntactic Dependency Parsing

Transition-based Neural Network Parsing

Graph-based Neural Network Parsing

Iterative-Refinement Neural Network Parsing

A Neural graph-based dependency parser

[Dozat and Manning 2017; Dozat, Qi, and Manning 2017]

- This paper revived interest in graph-based dependency parsing in a neural world
 - Designed a biaffine scoring model for neural dependency parsing
 - Also crucially uses a neural sequence model, something we discuss next week
- Really great results!
 - **But slower than the simple neural transition-based parsers**
 - There are n^2 possible dependencies in a sentence of length n



Method	UAS	LAS (PTB WSJ SD 3.3)
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79
Dozat & Manning 2017	95.74	94.08

LSTM-based Dependency Score Estimation

“Graph-based Dependency Parsing with Bidirectional LSTM”,
Wang and Chang. ACL 2016.

- ▶ Deep NN model computes context-dependent token embeddings
- ▶ Scores estimated from the token embedding pairs

Syntax-Aware Graph-to-Graph Transformer for Semantic Role Labelling



Slide from Alireza Mohammadshahi

Rep4NLP at ACL 2023

Authors:

Alireza Mohammadshahi
James Henderson

Motivation

Semantic Role Labelling (SRL)

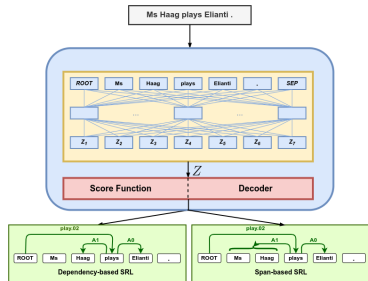
- Provides a shallow representation of the semantics in a sentence

Graph-Types:

- Dependency-based
- Span-based

Applications:

- ✓ Question Answering
- ✓ Machine Translation
- ✓ Natural Language Inference (NLI)
- ✓ ...



Slide from Alireza Mohammadshahi

Motivation

Syntactic Knowledge for SRL

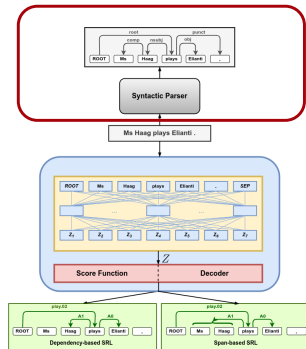


Open Questions:

- Is Syntax beneficial for SRL?
- How can one efficiently encode it?

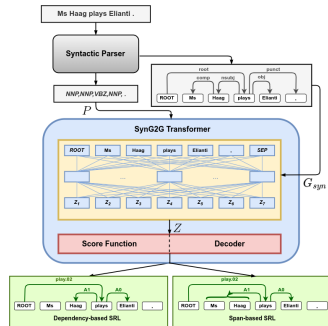
Previous Work:

- Use separate graph encoder (e.g. GNN)
- Hard coding of attention heads in Transformer
- Joint Training of Syntax and Semantic



Syntax-Aware Graph-to-Graph Transformer (SynG2G-Tr)

- Integrate the graph relations into the self-attention mechanism
 - Soft bias (the model can still learn other structures)
 - Efficient as it adds linear complexity to the computation
- ❖ In this work, we use syntactic dependency graph



Attention Mechanism

To input a graph, we modify equations of original Transformer:

$$z_i = \sum_j \alpha_{ij} (x_j W^V)$$
$$e_{ij} = \frac{1}{\sqrt{d}} [(x_i W^Q)(x_j W^K)^T + x_i W^Q (r_{ij} W^R)^T + r_{ij} W^R (x_j W^K)^T]$$

r_{ij} is the relation between token x_i and x_j

Results

Span-based SRL

Evaluation data: CoNLL 2005 benchmark

Without BERT:

- SynG2G-Tr outperforms Strubell et al. (2018)
- Benefit of soft bias instead of hard-coding

With BERT:

- Outperforms by 5.4%/8.8% F1 relative error reduction (RER) on average in both in-domain and out-of-domain settings
- Compatibility with BERT initialisation

Model	SA	WSJ (in-domain)			Brown (out-of-domain)		
		P	R	F1	P	R	F1
end-to-end							
He et al. (2017)	✗	85.0	84.3	84.6	74.9	72.4	73.6
He et al. (2018a)	✗	81.2	83.9	82.5	69.7	71.9	70.8
Strubell et al. (2018)	✓	85.53	84.45	<u>84.99</u>	75.8	73.54	<u>74.66</u>
Li et al. (2019)	✗	-	-	83.0	-	-	-
Xia et al. (2019)	✓	84.3	83.8	84.1	73.7	72.0	72.9
Xia et al. (2020)	✓	83.05	84.49	84.49	73.47	74.92	74.19
SynG2G-Tr (w/o BERT)	✓	84.48	86.46	85.45	73.92	76.65	75.26
+pre-training							
He et al. (2018a)	✗	84.8	87.2	86.0	73.9	78.4	76.1
Strubell et al. (2018)†	✓	87.13	86.67	<u>86.9</u>	79.02	77.49	<u>78.25</u>
Li et al. (2019)	✗	85.2	87.5	86.3	74.7	78.1	76.4
SynG2G-Tr	✓	86.86	88.3	87.57	80.01	81.07	80.53
given predicate							
Tan et al. (2017)	✗	84.5	85.2	84.8	73.5	74.6	74.1
He et al. (2018a)	✗	-	-	83.9	-	-	73.7
Strubell et al. (2018)†	✓	86.02	86.05	<u>86.04</u>	76.65	76.44	<u>76.54</u>
Ouchi et al. (2018)	✗	84.7	82.3	83.5	76.0	70.4	73.1
Xia et al. (2020)	✓	85.12	85.0	85.06	76.3	75.42	75.86
SynG2G-Tr (w/o BERT)	✓	86.46	86.56	86.50	77.73	77.18	77.45
+pre-training							
He et al. (2018a)	✗	-	-	87.4	-	-	80.4
Ouchi et al. (2018)	✗	88.2	87.0	87.6	79.9	77.5	78.7
Li et al. (2019)	✗	87.9	87.5	87.7	80.6	80.4	80.5
Jindal et al. (2020)	✗	87.70	88.15	87.93	81.52	81.36	81.44
Zhang et al. (2021)	✗	88.70	88.00	87.90	80.30	80.10	80.20
Jia et al. (2022)	✗	-	-	<u>88.25</u>	-	-	<u>81.90</u>
SynG2G-Tr	✓	89.11	88.74	88.93	83.93	82.50	83.21

Slide from Alireza Mohammadshahi

Results

Dependency-based SRL

Evaluation data: CoNLL 2009 benchmark

Without BERT:

- Outperforms previous work in both in-domain and out-of-domain settings

With BERT:

- Significantly outperforms previous work in end-to-end setting with 3.2%/10.4% F1 RER in both in-domain and out-of-domain
- Has competitive performance in given-predicate setting
- Outperforms Fei et al. (2021), which shows the benefit of SynG2G-Tr compared to GNN

Model	SA	WSJ (in-domain)			Brown (out-of-domain)		
		P	R	F1	P	R	F1
end-to-end							
He et al. (2018b)	✓	83.9	82.7	83.3	-	-	-
Cai et al. (2018)	✗	84.7	85.2	85.0	-	-	<u>72.5</u>
Li et al. (2019)	✗	-	-	<u>85.1</u>	-	-	-
SynG2G-Tr (w/o BERT)	✓	84.10	87.07	85.59	73.66	72.56	73.11
+pre-training							
Li et al. (2019)	✗	84.5	86.1	<u>85.3</u>	74.6	73.8	<u>74.2</u>
SynG2G-Tr	✓	86.38	89.78	88.05	80.35	83.57	81.93
given predicate							
Marcheggiani et al. (2017)	✗	88.7	86.8	87.7	79.4	76.2	77.7
M&T(2017)	✓	89.1	86.8	88.0	78.5	75.9	77.2
He et al. (2018b)	✓	89.7	89.3	89.5	81.9	76.9	79.3
Cai et al. (2018)	✗	89.9	89.2	<u>89.6</u>	79.8	78.3	79.0
Cai and Lapata (2019c)	✓	90.5	88.6	<u>89.6</u>	80.5	78.2	<u>79.4</u>
Kasai et al. (2019)	✓	89.0	88.2	<u>88.6</u>	78.0	77.2	77.6
SynG2G-Tr (w/o BERT)	✓	89.78	90.28	90.03	81.32	82.15	81.73
+pre-training							
Li et al. (2019)	✗	89.6	91.2	90.4	81.7	81.4	81.5
Kasai et al. (2019)	✓	90.3	90.0	90.2	81.0	80.5	80.8
Lyu et al. (2019)	✗	-	-	<u>90.99</u>	-	-	82.18
Chen et al. (2019)	✗	90.74	91.38	91.06	82.66	82.78	82.72
He et al. (2019)	✓	90.41	91.32	90.86	86.15	86.70	86.42
Cai and Lapata (2019a)	✓	91.1	90.4	90.7	82.1	81.3	81.6
Munir et al. (2021)	✓	91.2	90.6	90.9	83.1	82.6	82.8
SynG2G-Tr	✓	91.31	91.16	91.23	86.40	86.47	86.43
gold syntax							
Fei et al. (2021)	✓	92.5	92.5	<u>92.5</u>	85.6	85.3	<u>85.4</u>
SynG2G-Tr+Gold	✓	92.71	93.37	93.03	88.27	88.31	88.29

Slide from Alireza Mohammadshahi

Conclusion

- Propose SynG2G-Tr model for encoding the dependency parsing graph in the SRL task
- Evaluated our model on CoNLL 2005 and CoNLL 2009 datasets and outperform previous comparable models in most cases of both in-domain and out-of-domain sets

Outline

Syntactic Structure

Syntactic Dependency Parsing

Transition-based Neural Network Parsing

Graph-based Neural Network Parsing

Iterative-Refinement Neural Network Parsing

Recursive Non-Autoregressive Graph-to-Graph Transformer for Dependency Parsing with Iterative Refinement



Slide from Alireza Mohammadshahi

Transactions of ACL 2021

Authors:

Alireza Mohammadshahi
James Henderson

Motivation

Need an architecture to find the graph structure:

Model types { Autoregressive
Non-Autoregressive ✓

Non-Autoregressive Structure Prediction:

Predict all edges of the graph in parallel

Problem: Not considering between-edge information!



Our Proposal

We propose **Recursive Non-Autoregressive Graph-to-Graph Transformer (RNGTr)** architecture:

- **Refines arbitrary graphs** in an **iterative non-autoregressive** manner with conditioning on the complete graph.
- Defines a general Transformer-based encoder to input both sequences and graphs.
- Evaluated on dependency parsing and achieved SOTA on UD and Penn Treebanks.

RNGTr Model

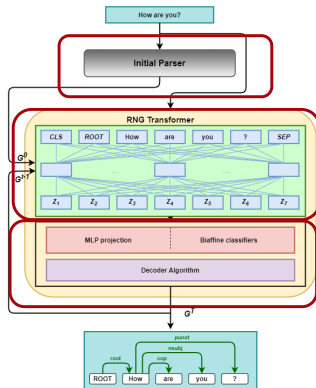
- Start with an initial parser (G^0)

- Refinement mechanism:

$$Z^t = E^{RNG}(W, P, G^{t-1})$$

$$G^t = D^{RNG}(Z^t)$$

- $t = 1, \dots, T$
- G^T is the final graph



Attention Mechanism

To input a graph, we modify equations of original Transformer:

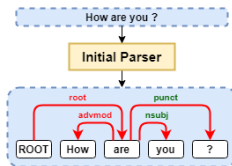
$$z_i = \sum_j \alpha_{ij} (x_j W^v + r_{ij} W_L^2)$$
$$e_{ij} = \frac{(x_i W^q)(x_j W^k + \text{LN}(r_{ij} W_L^1))}{\sqrt{d}}$$

r_{ij} is the relation between token x_i and x_j

Attention Mechanism

For labelled dependency graph:

$$(i \rightarrow^l j)_t: \begin{cases} r_{ij}^t = ind_l \\ r_{ji}^t = ind_l + |L| \end{cases}$$



	CLS	ROOT	How	are	you	?	SEP
CLS	0	0	0	0	0	0	0
ROOT	0	0	0	id_{root}	0	0	0
How	0	0	0	$id_{advmod} + L $	0	0	0
are	0	$id_{root} + L $	id_{advmod}	0	id_{nsubj}	id_{punct}	0
you	0	0	0	$id_{nsubj} + L $	0	0	0
?	0	0	0	$id_{punct} + L $	0	0	0
SEP	0	0	0	0	0	0	0

Self-Attention Mechanism

Initial Parsers

- Syntactic Transformer:
 - RNGTr model without the graph input

$$Z^0 = E^{RNG}(W, P)$$

$$G^0 = D^{RNG}(Z^0)$$

- For UD Treebanks: UDify
- For Penn Treebank: Biaffine parser
- Empty parser

UD Treebanks

- Integration with UDify
- Integration with SynTr
- Empty Parser

Language	Train Size	Mono [1]	Multi UDPipe	Multi UDify	Multi+Mono UDify+RNGTr	Mono SynTr	Mono SynTr+RNGTr	Mono Empty+RNGTr
Arabic	6.1K	81.8	82.94	82.88	85.93 (+17.81%)	86.23	86.31 (+0.58%)	86.05
Basque	5.4K	79.8	82.86	80.97	87.55 (+34.57%)	87.49	88.2 (+5.68%)	87.96
Chinese	4K	83.4	80.5	83.75	89.05 (+32.62%)	89.53	90.48 (+9.08%)	89.82
English	12.5K	87.6	86.97	88.5	91.23 (+23.74%)	91.41	91.52 (+1.28%)	91.23
Finnish	12.2K	83.9	87.46	82.03	91.87 (+54.76%)	91.80	91.92 (+1.46%)	91.78
Hebrew	5.2K	85.9	86.86	88.11	90.80 (+22.62%)	91.07	91.32 (+2.79%)	90.56
Hindi	13.3K	90.8	91.83	91.46	93.94 (+29.04%)	93.95	94.21 (+4.3%)	93.97
Italian	13.1K	91.7	91.54	93.69	94.65 (+15.21%)	95.08	95.16 (+1.62%)	94.96
Japanese	7.1K	92.1	93.73	92.08	95.41 (+42.06%)	95.66	95.71 (+1.16%)	95.56
Korean	4.4K	84.2	84.24	74.26	89.12 (+57.73%)	89.29	89.45 (+1.5%)	89.1
Russian	48.8K	91.0	92.32	93.13	94.51 (+20.09%)	94.60	94.47 (-2.4%)	94.31
Swedish	4.3K	86.9	86.61	89.03	92.02 (+27.26%)	92.03	92.46 (+5.4%)	92.40
Turkish	3.7K	64.9	67.56	67.44	72.07 (+14.22%)	72.52	73.08 (+2.04%)	71.99
Average	—	84.9	85.81	85.18	89.86	90.05	90.33	89.98

Slide from Alireza Mohammadshahi

Penn Treebank

- Integration with Biaffine
- Integration with SynTr

Model	Type	English (PTB)		Chinese (CTB)		German (CoNLL)	
		UAS	LAS	UAS	LAS	UAS	LAS
Chen and Manning (2014)	T	91.8	89.6	83.9	82.4	—	—
Dyer et al. (2015)	T	93.1	90.9	87.2	85.7	—	—
Ballesteros et al. (2016)	T	93.56	91.42	87.65	86.21	88.83	86.10
Cross and Huang (2016)	T	93.42	91.36	86.35	85.71	—	—
Weiss et al. (2015)	T	94.26	92.41	—	—	—	—
Andor et al. (2016)	T	94.61	92.79	—	—	90.91	89.15
Mohammadshahi and Henderson (2020)	T	96.11	94.33	—	—	—	—
Ma et al. (2018)	T	95.87	94.19	90.59	89.29	93.65	92.11
Fernández-González and Gómez-Rodríguez (2019)	T	96.04	94.43	—	—	—	—
Kiperwasser and Goldberg (2016)	G	93.1	91.0	86.6	85.1	—	—
Wang and Chang (2016)	G	94.08	91.82	87.55	86.23	—	—
Cheng et al. (2016)	G	94.10	91.49	88.1	85.7	—	—
Kuncoro et al. (2016)	G	94.26	92.06	88.87	87.30	91.60	89.24
Ma and Hovy (2017)	G	94.88	92.98	89.05	87.74	92.58	90.54
Ji et al. (2019)	G	95.97	94.31	—	—	—	—
Li et al. (2020)+ELMo	G	96.37	94.57	90.51	89.45	—	—
Li et al. (2020)+BERT	G	96.44	94.63	90.89	89.73	—	—
Biaffine (Dozat and Manning, 2016)	G	95.74	94.08	89.30	88.23	93.46	91.44
Biaffine+RNGTr	G	96.44	94.71	91.85	90.12	94.68	93.30
SynTr	G	96.60	94.94	92.42	90.67	95.11	93.98
SynTr+RNGTr	G	96.66	95.01	92.98	91.18	95.28	94.02

Conclusion

- We proposed a **general structure refinement model**, based on Transformer architecture.
- We evaluated the model on the dependency parsing task.
- We achieved **state-of-the-art** results on UD and Penn Treebanks.

Summary of Neural Network Syntactic Parsing

- ▶ Deep NNs are very good at encoding the context of a parsing decision
- ▶ Deep NNs are very good at scoring dependency relations
- ▶ Iterative refinement with Graph2Graph Transformer improves accuracy on complex graphs

Summary of Graph Processing in Transformers

Transformers are latent graph processing models
(not only sequence-to-sequence models)

- ▶ self-attention does latent graph processing
- ▶ observed graph relations can be output with attention-like functions
- ▶ observed graph relations can be input as features in the attention function
- ▶ predicted graphs relations can be iteratively predicted and encoded

G2G Transformers can jointly encode latent, observed and predicted graphs in one set-of-vectors embedding