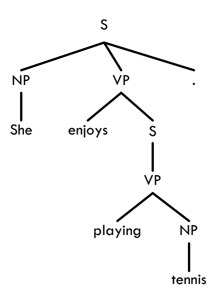
Neural Constituency Parsing

Dan Klein
CS 288

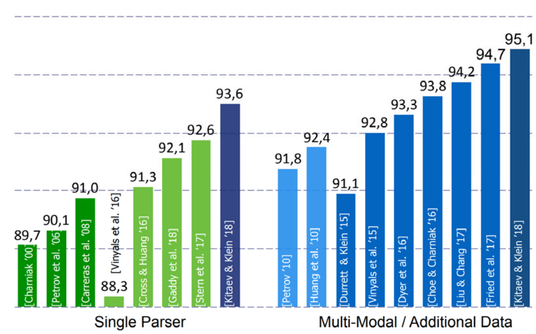
Syntactic Parsing

She enjoys playing tennis.

Syntactic Parsing

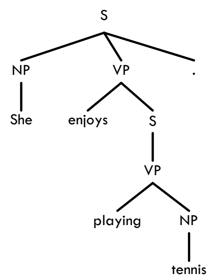


Historical Trends



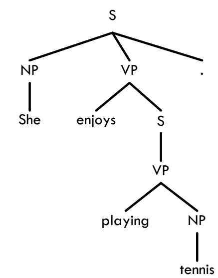
[Slide from Slav Petrov]

Output Correlations



Grammars

S → NP VP



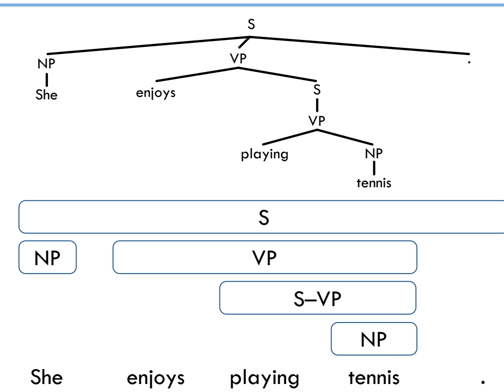
VP[*enjoys*] : S[*playing*]

NP^S → *she*

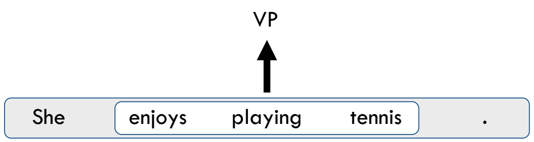
Input-Output Correlations

She enjoys playing tennis.

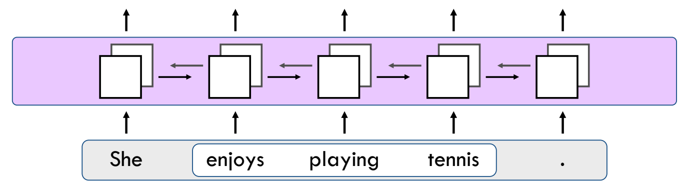
Span-Based Parsing



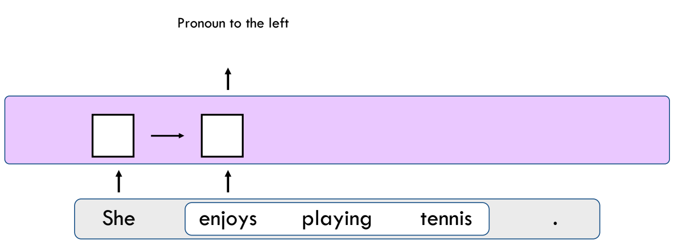
Parsing as Span Classification



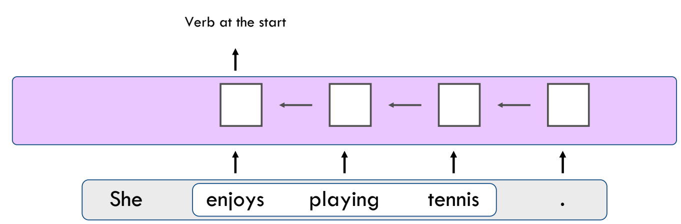
Routing with LSTMs



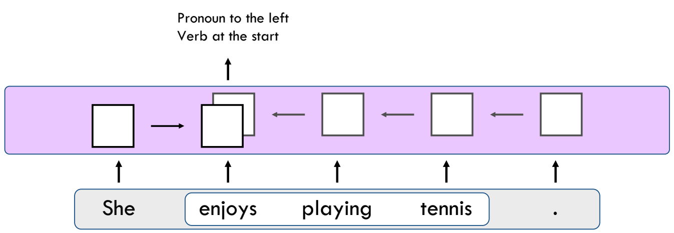
Routing with LSTMs



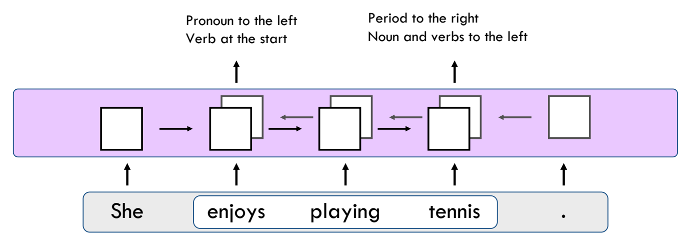
Routing with LSTMs



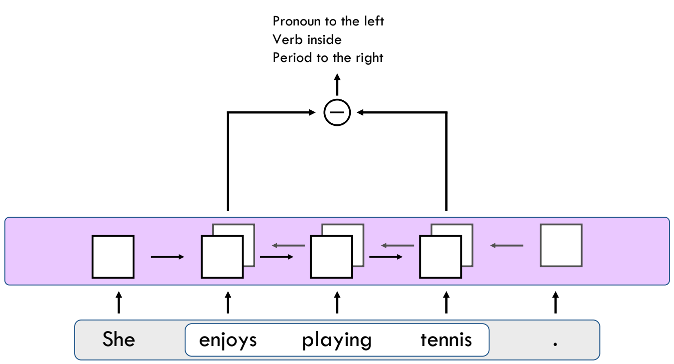
Routing with LSTMs



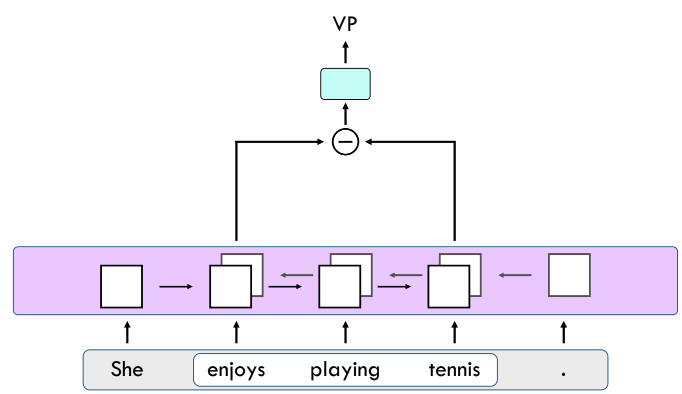
Routing with LSTMs



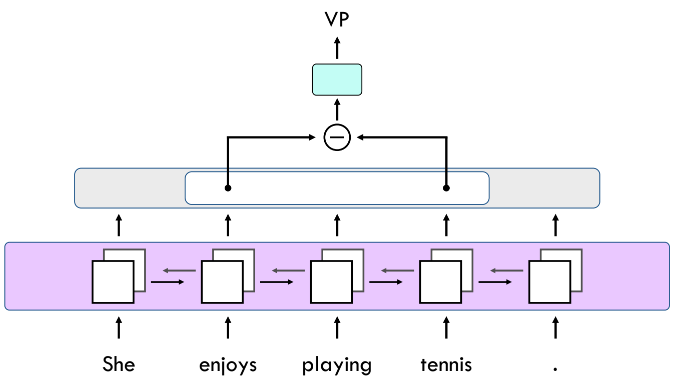
Span Classification



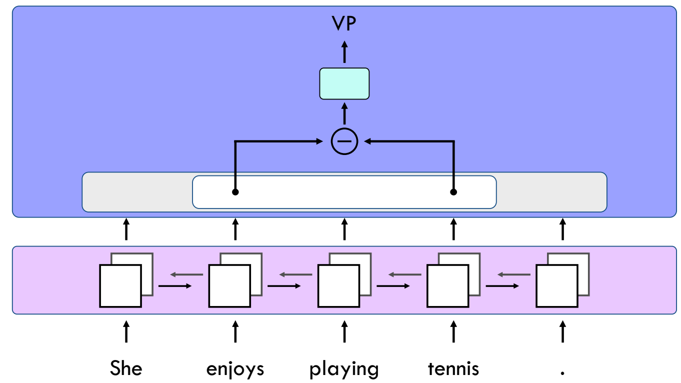
Span Classification



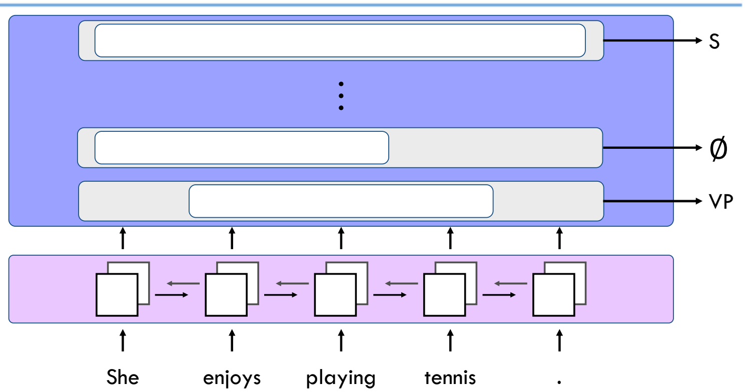
Span Classification



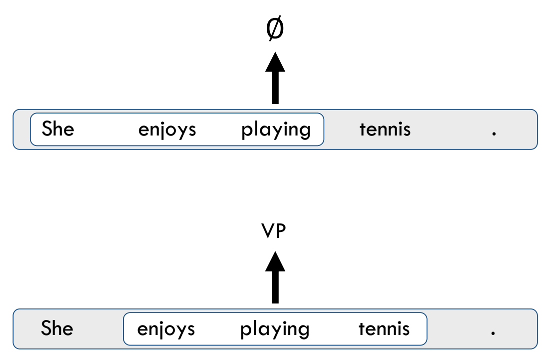
Span Classification



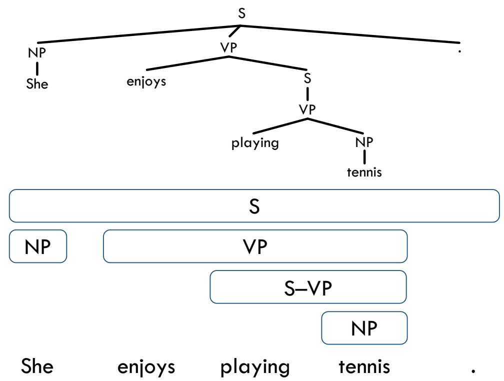
Span Classification



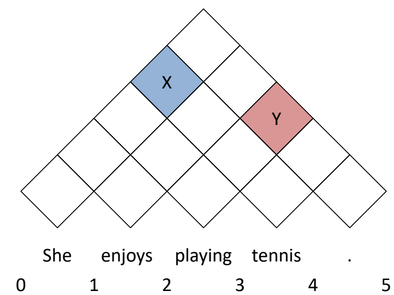
Non-Constituents



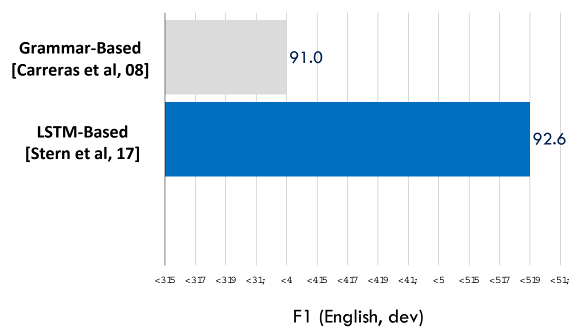
… But Will We Get a Tree Out?



Reconciliation



Does It Work?



What's Going on in There?

Neural parsers no longer have much of the model structure provided to classical parsers.

How do they perform so well without it?

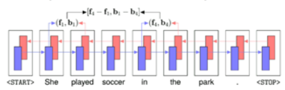
What's Going on in There?

Why don't we need a grammar?

Adjacent tree labels are redundant with LSTM features

If we can predict surrounding tree labels from our LSTM representation of the input, then this information doesn't need to be provided explicitly by grammar production rules

We find that for **92.3%** of spans, the label of the span's parent can predicted from the neural representation of the span



What's Going on in There?

Do we need tree constraints?

Not for F1

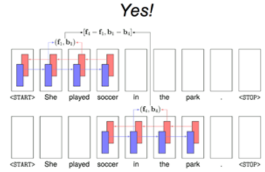
Many neural parsers no longer model output correlations with grammar rules, but still use output correlations from tree constraints

Predicting span brackets independently gives **nearly identical performance** on PTB development set F1 and produces valid trees for **94.5%** of sentences

What's Going on in There?

Is distant context important?

Yes!



Almost a full point of F1 is lost by truncating context 5 words away from span endpoints and half a point with 10 words

What's Going on in There?

Do LSTMs introduce useful inductive bias compared to feedforward networks?

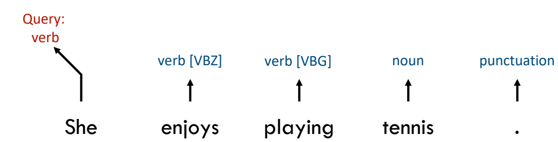
Yes!

We compare a truncated LSTM with feedforward architectures that are given the same inputs

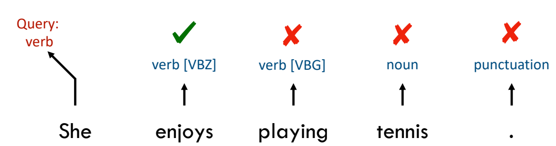
The LSTM outperformed the best feedforward by **6.5 F1**

Routing with Transformers

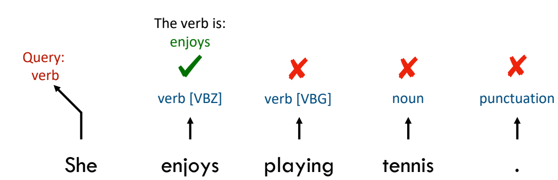
Routing with Transformers



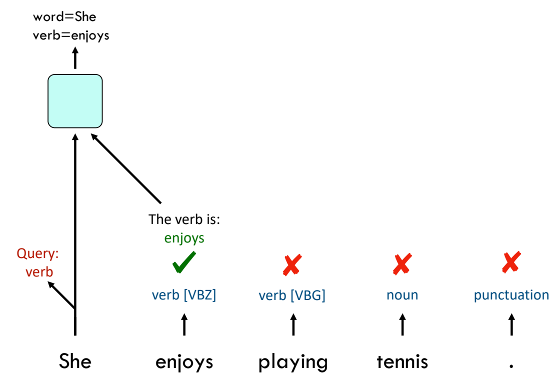
Routing with Transformers



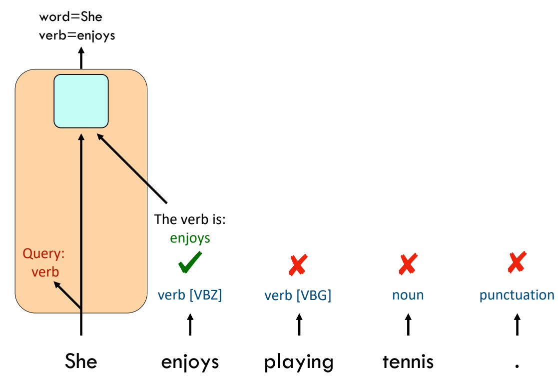
Routing with Transformers



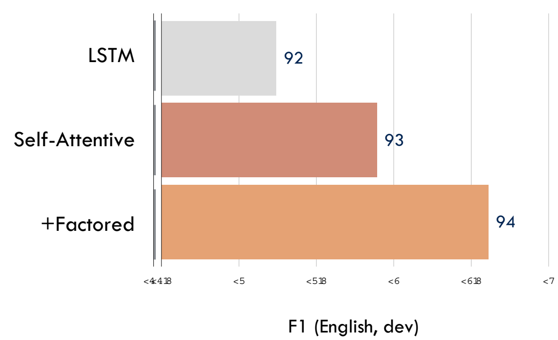
Routing with Transformers



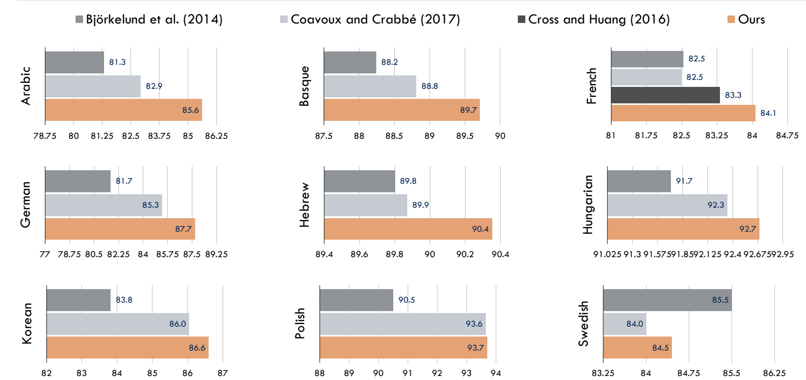
Routing with Transformers



What Helps?



Results: Multilingual



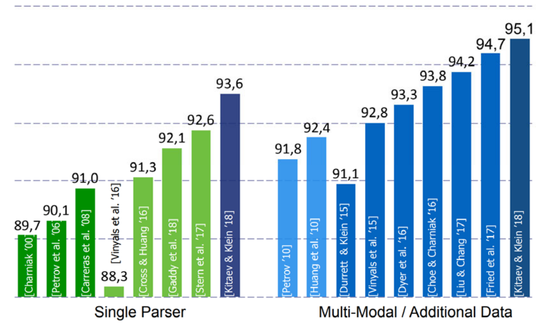
Pre-Training

Problem: Input has more variation than output

Need to handle:

- Rare words not seen during training
- Word forms in morphologically rich languages
- Contextual paraphrase / lexical variation

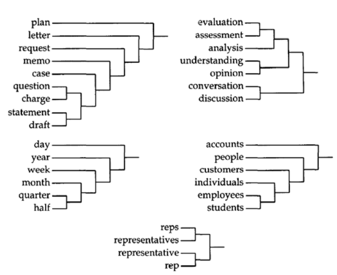
Historical Trends

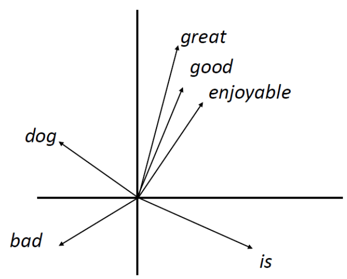


[Slide from Slav Petrov]

Knowledge Modularity

- Knowledge modularity: Learn domain-general knowledge from one data source and use it solve specific problems elsewhere

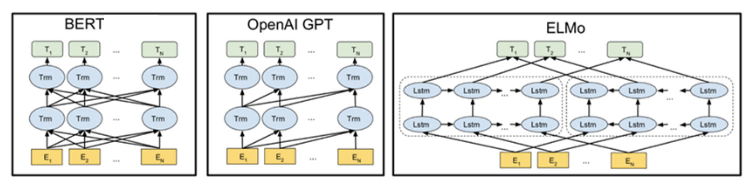




Context Embeddings and Pretraining

Key Idea: Embed contexts, not words. Use these embeddings for other tasks.

Example: BERT (Devlin et al., 2019) -- bidirectional Transformer trained on masked language modeling and next-sentence prediction



Explosion of Pretraining Work

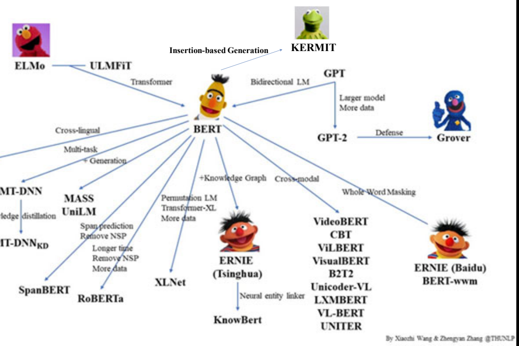
Model	URL	Score
ALBERT (Ensemble)		89.4
ALICE v2 large ensemble (Alibaba DAMO NLP)	↗	89.0
FreeLB-RoBERTa (ensemble)	↗	88.8
RoBERTa	↗	88.5
XLNet-Large (ensemble)	↗	88.4
MT-DNN-ensemble	↗	87.6
GLUE Human Baselines	↗	87.1
Snorkel MeTaL	↗	83.2
XLM (English only)	↗	83.1
SemBERT	↗	82.9
SpanBERT (single-task training)	↗	82.8
BERT + BAM	↗	82.3
Span-Extractive BERT on STILTs	↗	82.3
BERT on STILTs	↗	82.0
RGLM-Base (Huawei Noah's Ark Lab)		81.3
BERT: 24-layers, 16-heads, 1024-hidden	↗	80.5
BERT + Single-task Adapters	↗	80.2
Macaron Net-base	↗	79.7
SesameBERT-Base		78.6
MobileBERT		78.5
StackingBERT-Base	↗	78.4
TinyBERT	↗	75.4
BiLSTM+ELMo+Attn	↗	70.0

GLUE SoTA (ICLR 2020)

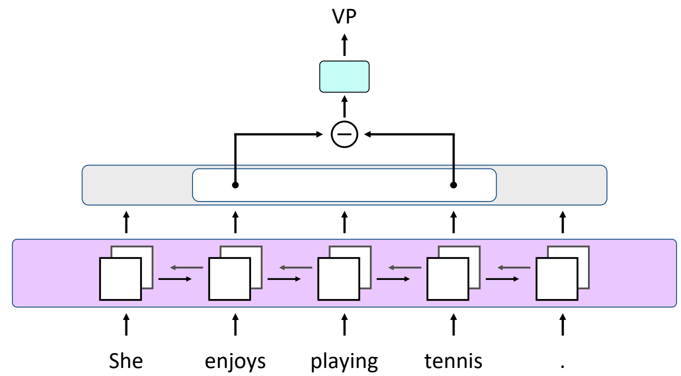
Human

BERT

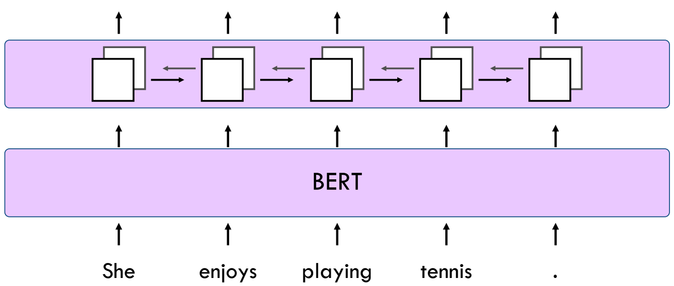
GLUE Baseline (ICLR 2019)



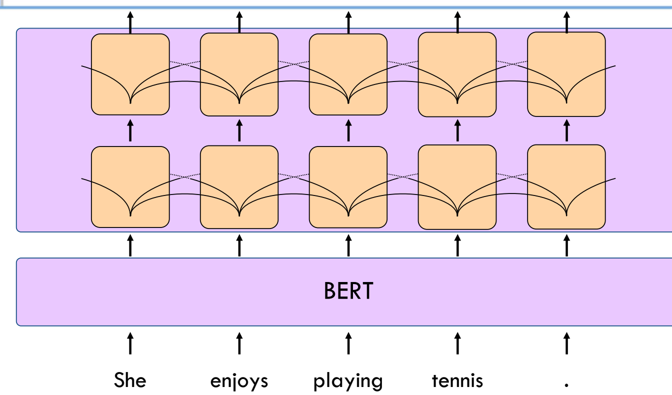
Parsing as Span Classification



Pretraining



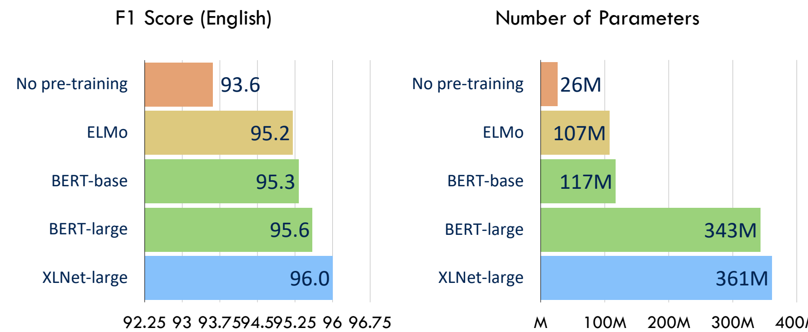
Architecture



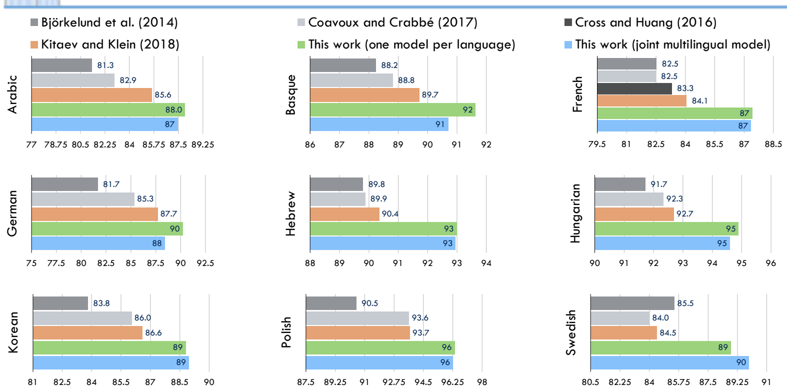
Encoder Architectures

	LSTM	Self-Attention
No pre-training	92.08 F1 [Gaddy+ 2018]	93.55 F1 [Kitaev & Klein 2018]
Pre-training	95.13 F1 (with ELMo) [Kitaev & Klein 2018]	95.60 F1 (with BERT) [Kitaev et al 2019]

Encoder Architectures



Results: Multilingual



Does Structure Help?

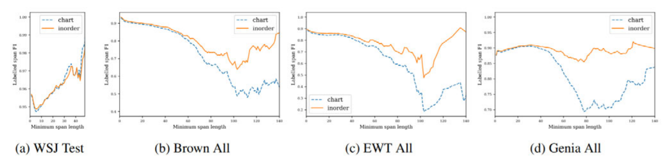


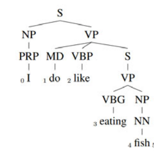
Figure 1: Labelled bracketing F1 versus minimum span length for the English corpora. F1 scores for the In-Order parser with BERT (orange) and the Chart parser with BERT (cyan) start to diverge for longer spans.

Out of Domain Parsing

	Berkeley		BLLIP		In-Order		Chart	
	F1	Δ Err.	F1	Δ Err.	F1	Δ Err.	F1	Δ Err.
WSJ Test	90.06	+0.0%	91.48	+0.0%	91.47	+0.0%	93.27	+0.0%
Brown All	84.64	+54.5%	85.89	+65.6%	85.60	+68.9%	88.04	+77.7%
Genia All	79.11	+110.2%	79.63	+139.1%	80.31	+130.9%	82.68	+157.4%
EWT All	77.38	+127.6%	79.91	+135.8%	79.07	+145.4%	82.22	+164.2%

Neural parsers improve out-of-domain numbers, but not more than in-domain numbers

Other Neural Constituency Parsers



steps	structural action	label action	stack after	bracket
1–2	sh(I/PRP)	label-NP	$_0\triangle_1$	$_0NP_1$
3–4	sh(do/MD)	nolabel	$_0\triangle_1\triangle_2$	
5–6	sh(like/VBP)	nolabel	$_0\triangle_1\triangle_2\triangle_3$	
7–8	comb	nolabel	$_0\triangle_1\triangle_3$	
9–10	sh(eating/VBG)	nolabel	$_0\triangle_1\triangle_3\triangle_4$	
11–12	sh(fish/NN)	label-NP	$_0\triangle_1\triangle_3\triangle_4\triangle_5$	$_4NP_5$
13–14	comb	label-S-VP	$_0\triangle_1\triangle_3\triangle_5$	$_3S_5$, $_3VP_5$
15–16	comb	label-VP	$_0\triangle_1\triangle_5$	$_1VP_5$
17–18	comb	label-S	$_0\triangle_5$	$_0S_5$

- Back to at least Henderson 1998!
- Recent directions:
  - Shift-Reduce, eg Cross and Huang 2016
  - SR/Generative, eg Dyer et al 2016 (RNNG)
  - In-Order Generative, eg Liu and Zhang 2017

Open Source Release

Code and models are publicly available at: `github.com/nikitakit/self-attentive-parser`

Sample Usage (with spaCy integration)

```
>>> import spacy
>>> from benepar.spacy_plugin import BeneparComponent
>>> nlp = spacy.load('en')
>>> nlp.add_pipe(BeneparComponent("benepar_en"))
>>> doc = nlp(u"Short cuts make long delays.")
>>> sent = list(doc.sents)[0]
>>> print(sent._.parse_string)
(S (NP (JJ Short) (NNS cuts)) (VP (VBP make) (NP (JJ
long) (NNS delays))) (. .))
>>> sent._.labels
('S',)
>>> list(sent._.children)[0]
Short cuts
```

Sample Usage (with NLTK integration)

```
>>> import benepar
>>> parser = benepar.Parser("benepar_en")
>>> tree = parser.parse("Short cuts make long delays.")
>>> print(tree)
(S
  (NP (JJ Short) (NNS cuts))
  (VP (VBP make) (NP (JJ long) (NNS delays)))
  (. .))
```