Natural Language Understanding, Generation, and Machine Translation

Lecture 12: Neural Parsing

Frank Keller

Week of 7 February 2022 (week 4)

School of Informatics
University of Edinburgh
keller@inf.ed.ac.uk

The Story so Far

Neural Parsing

Potential Problems

Results

Parsing with Transformers

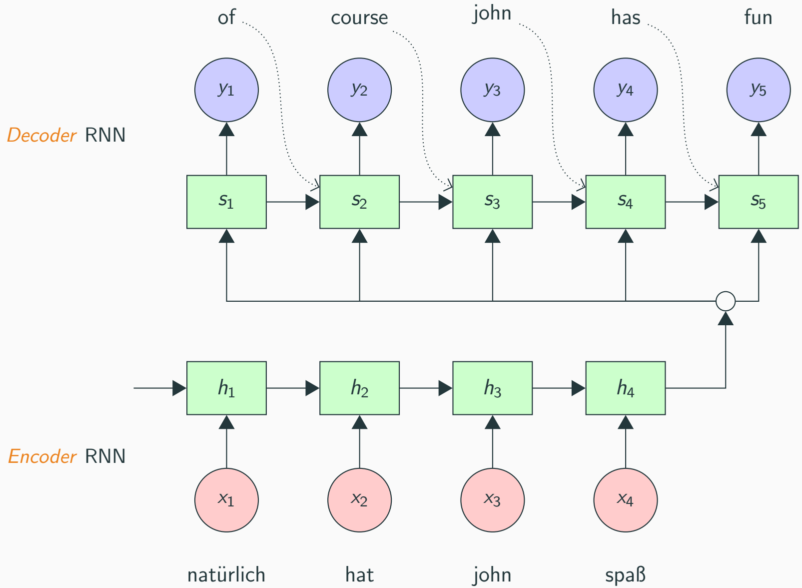
Reading: Vinyals et al. (2015)

The Story so Far

Encoder-Decoder Architecture

- So far, we have used the encoder-decoder architecture for machine translation.
- But it can be used for any task where both the input and output are sequences of symbols.
- In this lecture, we will use it for **syntactic parsing.**
- We will see an LSTM-with-attention encoder-decoder, but also discuss a transformer-based model.

Reminder: Encoder-Decoder Architecture



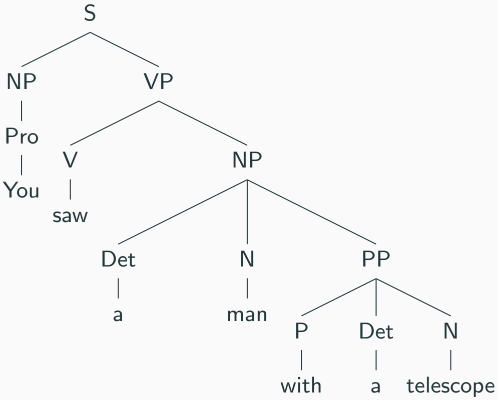
Neural Parsing

Parsing

Parsing is the task of turning a sequence of words:

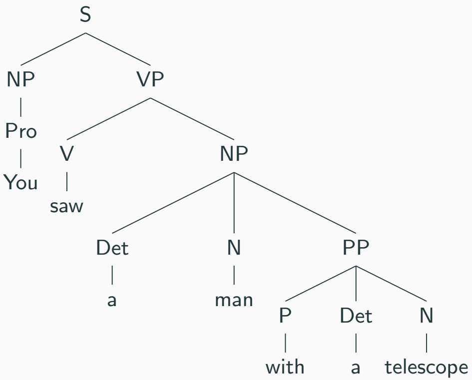
(1) You saw a man with a telescope.

into a syntax tree:



Linearizing the Input

But how can we use an encoder-decoder model for parsing? The input is a sequence, but the output is not.



We can **linearize** the syntax tree:

(S (NP (Pro You)) (VP (V saw) (NP (Det a) (N man) (PP (P with) (Det a) (N telescope)))))

Linearizing the Input

Now we have a sequence that represents the syntax tree:

(S (NP (Pro You)) (VP (V saw) (NP (Det a) (N man) (PP (P with) (Det a) (N telescope)))))

We can simplify it by stripping out the words:

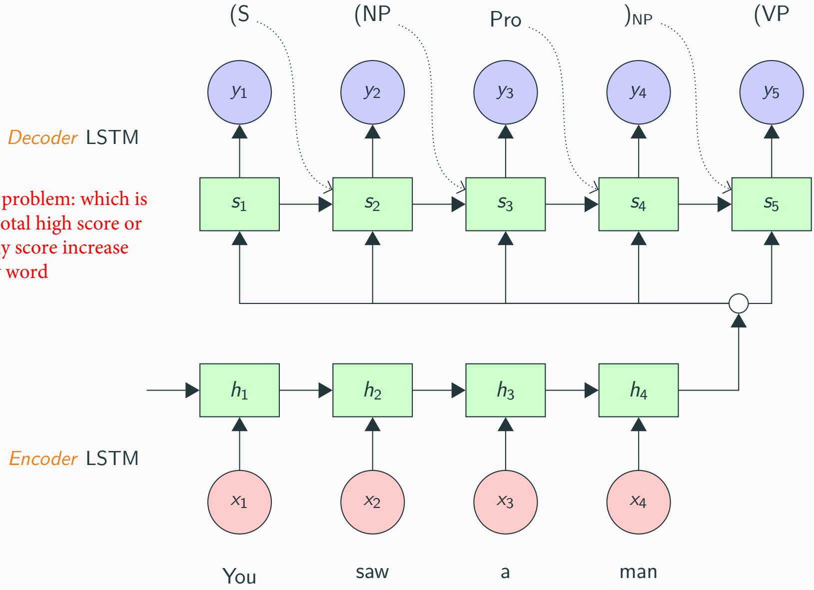
(S (NP Pro) (VP V (NP Det N (PP P Det N))))

And we can make it easier to process by annotating also the closing brackets:

(S (NP Pro $)_{NP}$ (VP V (NP Det N (PP P Det N $)_{PP}$ $)_{NP}$ $)_{VP}$ $)_{S}$

give the contextual information

An Encoder-Decoder for Parsing



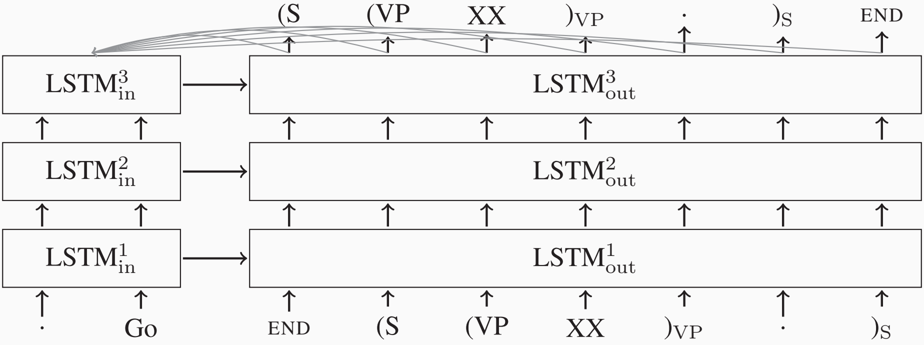
seq2seq problem: which is better? total high score or gradually score increase word by word

An Encoder-Decoder for Parsing

Essentially, that is our parsing model! But in order for this to work properly, we also need to:

- Add an end-of-sequence symbol, as output sequences can vary in length.
- Reverse the input string: results in small performance gain.
- Make the network deeper. Vinyals et al. (2015) use three LSTM layers for both encoder and decoder.
- Add attention. This essentially works like the encoder-decoder with attention we saw for MT (lecture 7).
- Use pre-trained word embeddings as input (here: word2vec).
- Get lots of training data. Vinyals et al. (2015) use an existing parser (the Berkeley parser) to parse a large amount of text, which they then use as training data.

This is how they draw their model architecture:



Potential Problems

Potential Problems

But wait! Can this really work? What about:

- How do we make sure that opening and closing brackets match? Else we won't have a well-formed tree!
- How do we associate the words in the input with the leaves of the tree in the output?
- The output sequence can be longer than the input sequence, isn't this a problem? single word has many barckets
- How can I make sure that the model outputs the best overall sequence, not just the best symbol at each time step?

Potential Problems

How to deal with these problems:

- This is really rare (0.8–1.5% of sentences). And if it occurs, just fix the brackets in post-processing (add brackets to beginning or end of the sequence).
- You could just associate each input word with a PoS in the output, in sequence order. But in practice: only the tree is evaluated. Vinyals et al. (2015) replace all PoS tags with XX.
- Not a problem, also happens in MT. And only the tree is evaluated.
- Use beam search to generate the output (as in MT). However, in practice, beam size has very little impact on performance.

Results

Training Corpora

Training corpora used:

- Wall Street Journal (WSJ): treebank with 40k manually annotated sentences.
- BerkeleyParser corpus: 90k sentences from WSJ and several other treebanks, and 11M sentences parsed with Berkeley Parser.
- High-confidence corpus: 90k sentences from WSJ from several treebanks, and 11M sentences for which two parsers produce the same tree (length resampled).

Results

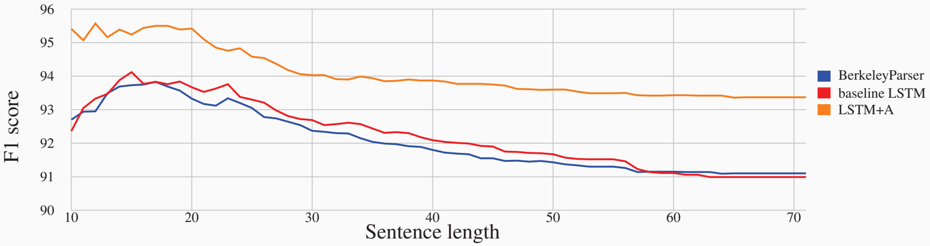
Results reported by Vinyals et al. (2015):

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**

The second half of the table lists results for various versions of the Berkeley parser.

Current state of the art: 95–96.

Results



96
95
94
93
92
91
90
F1 score
10
20
30
40
50
60
70
Sentence length
BerkeleyParser
baseline LSTM
LSTM+A

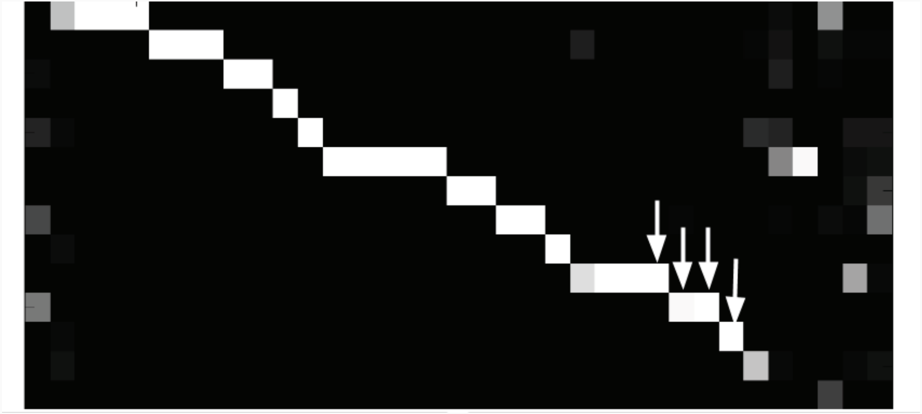
Results

Summary of results:

- The encoder-decoder model only works if attention is used.
- Ensembling models helps. Which is almost always the case.
- If we have a lot of training data, we get a large boost. And even the simple model without attention starts to work.
- A simple encoder-decoder model can match the performance of the Berkeley parser (a probabilistic chart parser; $O(n^3)$ complexity).
- Even though we use an LSTM, performance by sentence length is same or better than the Berkeley parser.

Analyzing the Attention

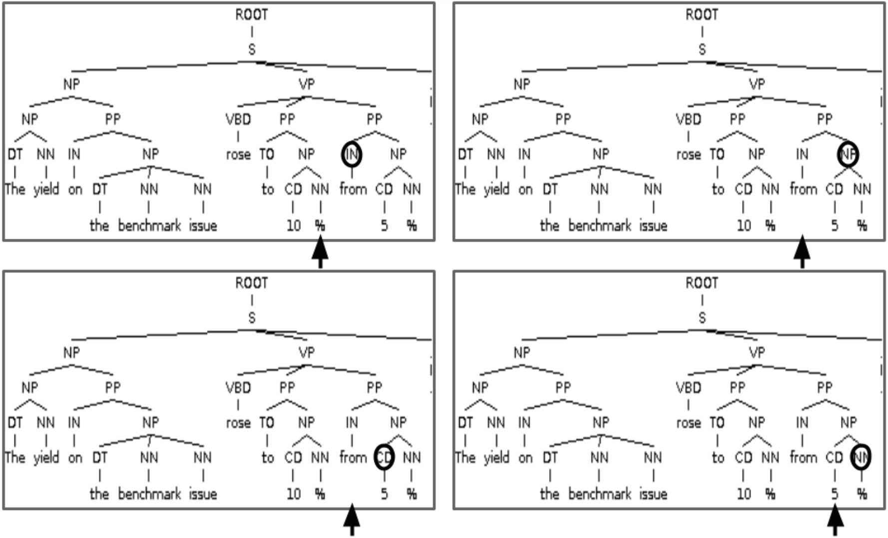
Attention matrix: columns are attention vectors over inputs:



Analyzing the Attention

Arrows: where the model attends.

Circles: current output being decoded in the tree.



Analyzing the Attention

- Attention matrix shows that the model focuses on one word as it produces the parse tree.
- It moves through the input sequence monotonically (left to right).
- Model learns stack-like behavior when producing the output.
- Note that if model focuses on position i, that state has information for all words after i (input is reversed).
- In some cases, the model skips words.

Parsing with Transformers

Parsing with Transformers

Could we use transformers for parsing, wouldn't that work even better? Yes! We will briefly look at Kitaev and Klein (2018):

- They use a transformer to encode the input.
- This results in a "context aware summary vector" (embedding) for each input word.
- The embedding encodes word, PoS tag, and position information.
- The embedding layers are combined to obtain **span scores.**
- The attention blocks, and the attention heads, are just like in Vaswani et al. (2017).
- But they also try **factored attention heads**, which separate position and content information.

Parsing with Transformers

The **decoder** of Kitaev and Klein (2018) works as follows: A real-valued score $s(T)$ is assigned to a tree T:

$$s(T) = \sum_{(i,j,l) \in T} s(i, j, l)$$

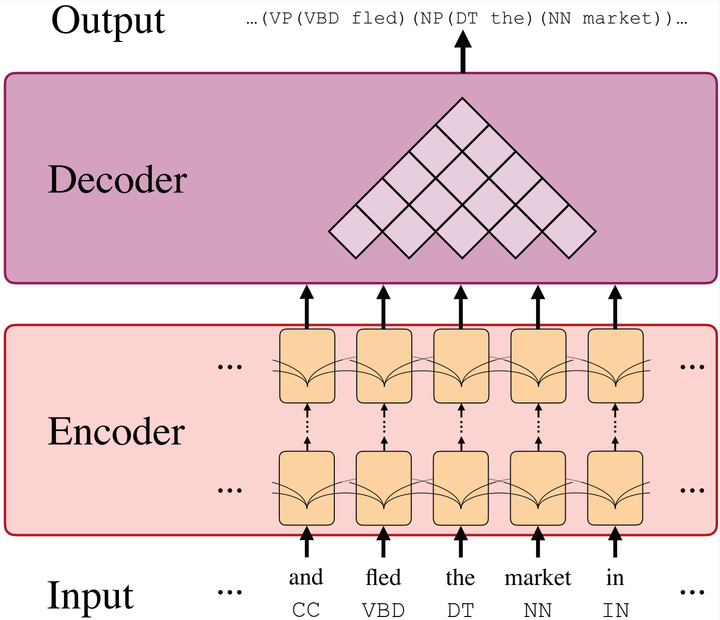
where $s(i, j, l)$ is a score for the constituent located between position i and position j with label l.

At test time, we compute:

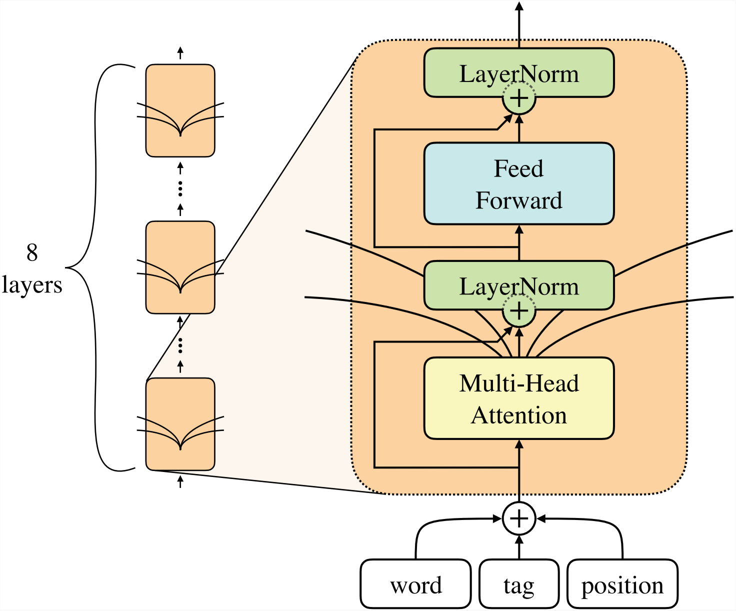
$$\hat{T} = \operatorname*{argmax}_{T} s(T)$$

This can be found efficiently using CYK (remember ANLP?). **This gives us the optimal output sequence, unlike beam search!**

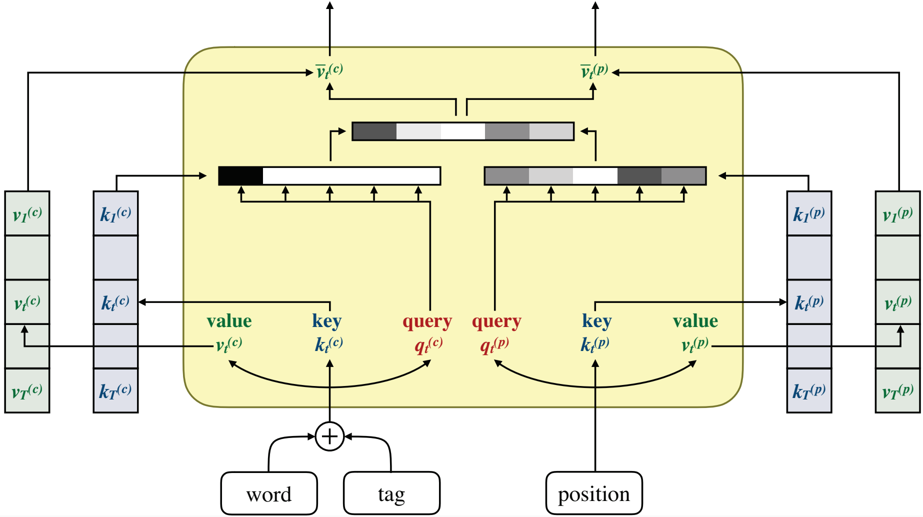
Overall Architecture



Transformer Block



Factored Attention Head



Results

Encoder Architecture	F1 (dev)	Δ
LSTM (Gaddy et al., 2018)	92.24	-0.43
Self-attentive (Section 2)	92.67	0.00
+ Factored (Section 3)	93.15	0.48
+ CharLSTM (Section 5.1)	93.61	0.94
+ ELMo (Section 5.2)	95.21	2.54

	LR	LP	F1
Single model, WSJ only			
Vinyals et al. (2015)	–	–	88.3
Cross and Huang (2016)	90.5	92.1	91.3
Gaddy et al. (2018)	91.76	92.41	92.08
Stern et al. (2017b)	92.57	92.56	92.56
Ours (CharLSTM)	**93.20**	**93.90**	**93.55**

ELMo is a pre-trained embedding model similar to BERT.

Summary

- We can use encoder-decoder architectures for parsing.
- For this, the output sequence has to be a linearized parse tree.
- Even a simple LSTM encoder-decoder works well, if given enough training data.
- It learns to generate well-formed trees (balanced brackets).
- We can improve this further by using a transformer-based encoder instead of an LSTM.
- For the decoder, we can use CYK over span scores to compute the optimal output sequence.

References

Kitaev, Nikita and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. Melbourne, pages 2676–2686.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*. Curran Associates, Red Hook, NY, pages 5998–6008.

Vinyals, Oriol, Terry Koo, Lukasz Kaiser, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*. Curran Associates, Red Hook, NY.