Exploring linguistic structure in self-attentions of Neural Machine Translation

David Mareček

Institute of Formal and Applied Linguistics, Charles University, Prague

Copenhagen, June 18th, 2018

About me

- PhD in 2012: Unsupervised Dependency Parsing
- Postdoc grant (2014 – 2016): Sentence structure induction without annotated corpora
  - using supervised POS tags
  - without supervised POS tags
  - problems with evaluation - different annotation styles

Less linguistics in today's NLP

E.g. Machine Translation

- TectoMT (Zabokrtsky et al., 2006) – analysis-transfer-synthesis MT system, BLEU: 15
- Moses (Koehn et al., 2006) – phrase-based MT system, BLEU: 18
- NMT (Vaswani et al., 2017) – neural MT system, BLEU: 24

Linguistic structure representation in neural networks

- 3 year grant by National Scientific Foundation of Czech Republic
- 2018 – 2020
- Many end-to-end NLP applications do not use lingustic subtasks (tagging, parsing, ...)
- Project goal: Is there any linguistic structure inside the neural networks?
- How does it correspond to linguistic theories?

The Tasks

- Machine Translation (LSTM/GRU, Transformer)
- Image captioning (CNN + GRU)
- Sentiment analysis (LSTM)
- Text summarization

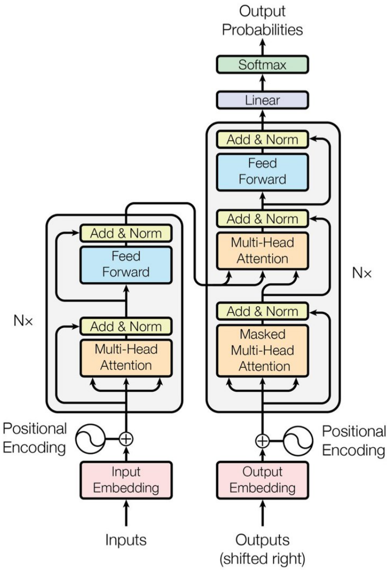
Goals

- Are there any linguistic features in the hidden states?
- How accurately can we predict e.g. POS tags, morpohogical fetaures, or semantic features form the hidden states?
- Does the attention mechanism somehow reflect the dependency or constituency structure?

This presentation

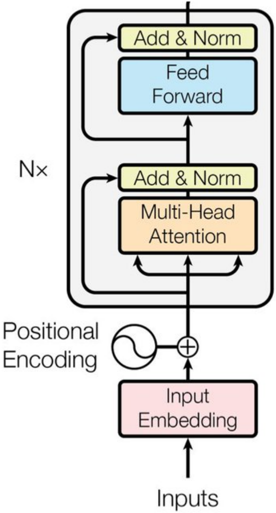
- Task: Machine translation using the Transformer architecture (Vaswani et al, 2017)
- Predicting constituency trees from the encoder's self-attentions

Transformer



Transformer - Encoder

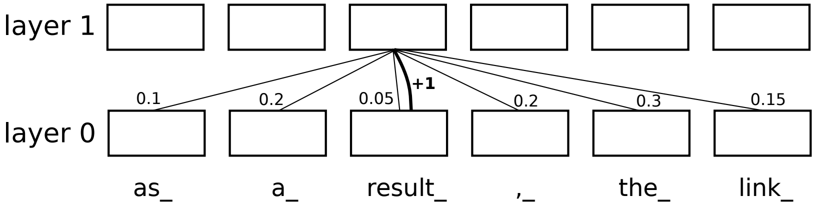
- For each position, the self-attention mechanism looks at all other positions in the previous layer
- Residual connections boost the information about particular position from the previous layer
- Attentions to the same positions are learned to be weaker because of these residual connections



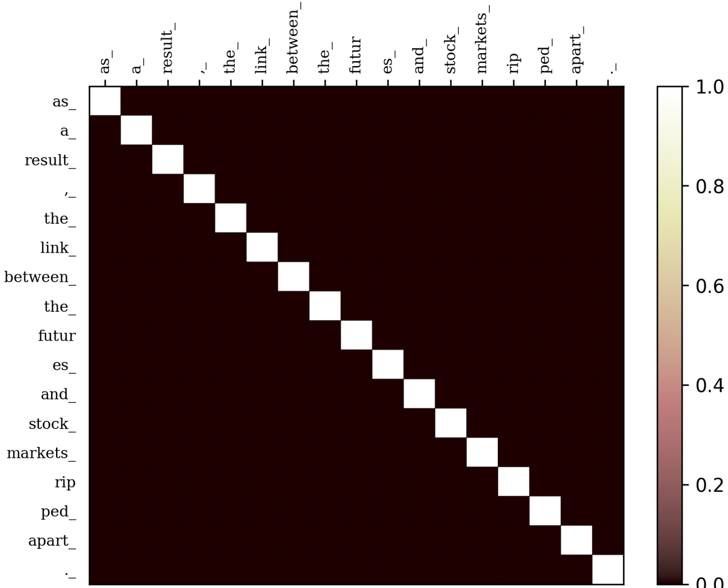
Self-attention Aggregation

We want to capture how much each token (or wordpiece) affects each particular position for each layer in the encoder.

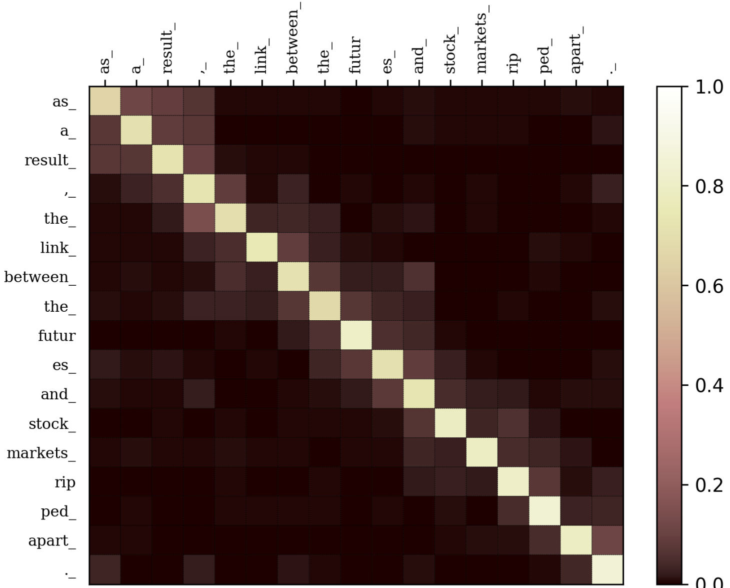
- Collect the attention distribution to the previous layer.
- Because of the residual connections, add +1 to boost the same position.
- Normalize.



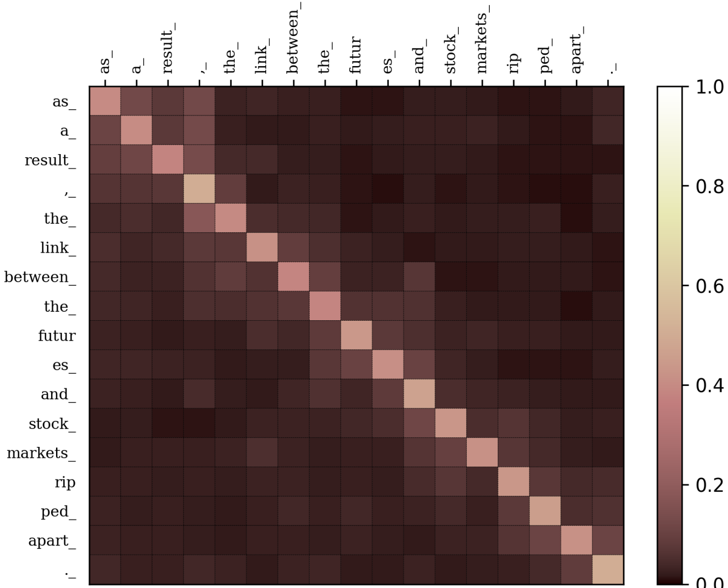
Aggregated attention visualisation - layer 0



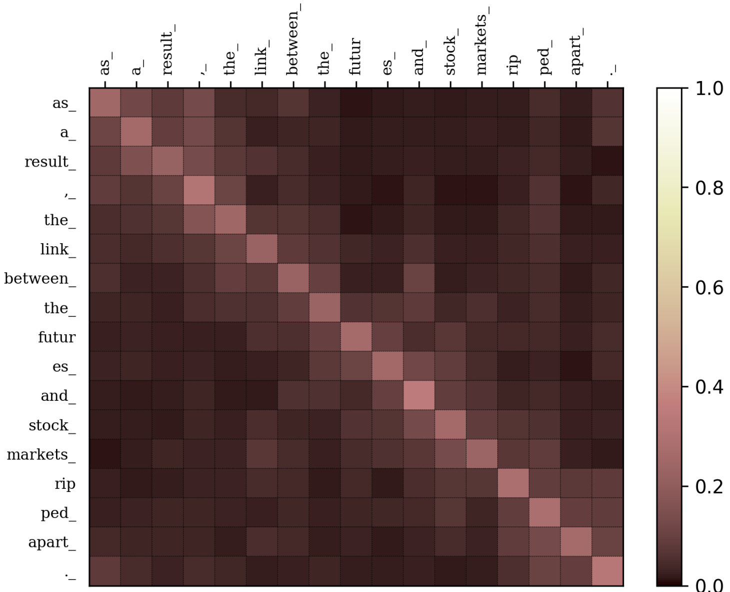
Aggregated attention visualisation - layer 1



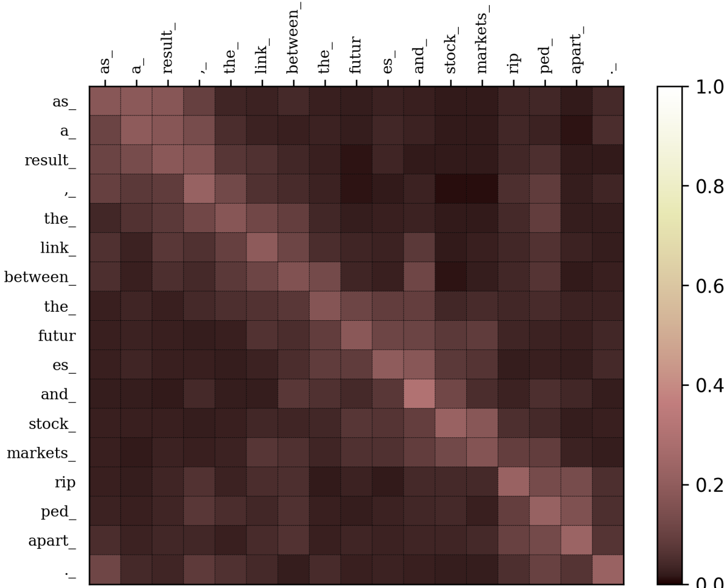
Aggregated attention visualisation - layer 2



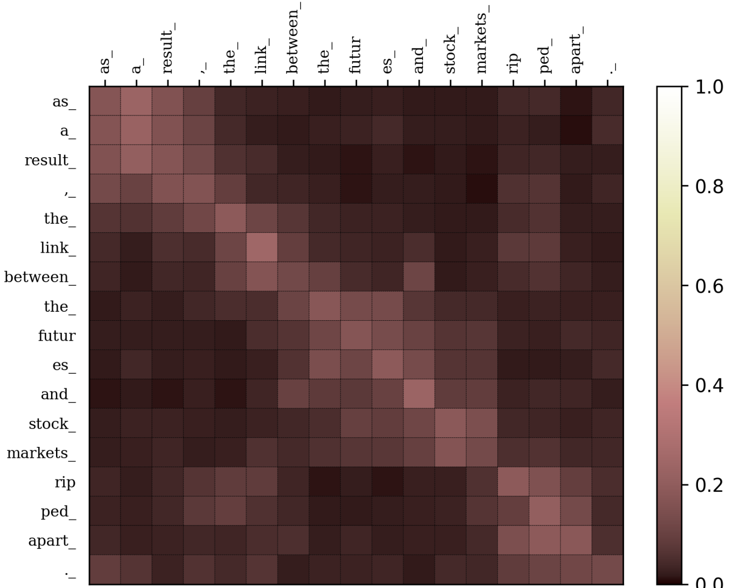
Aggregated attention visualisation - layer 3



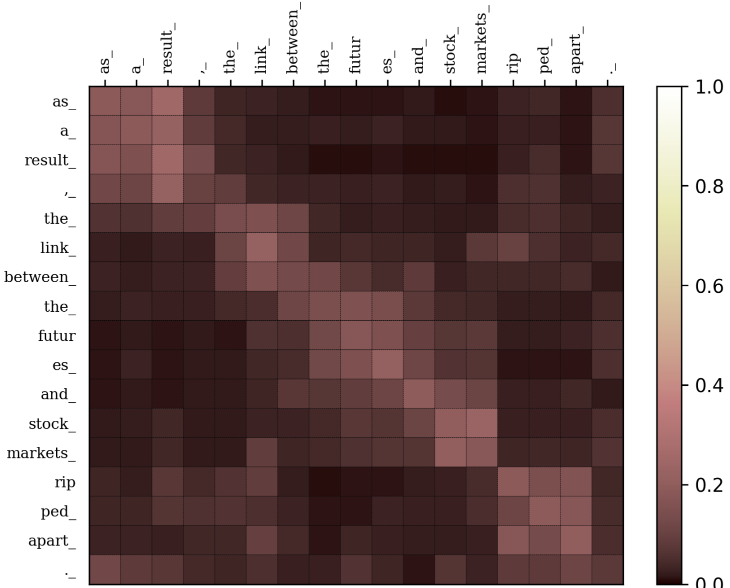
Aggregated attention visualisation - layer 4



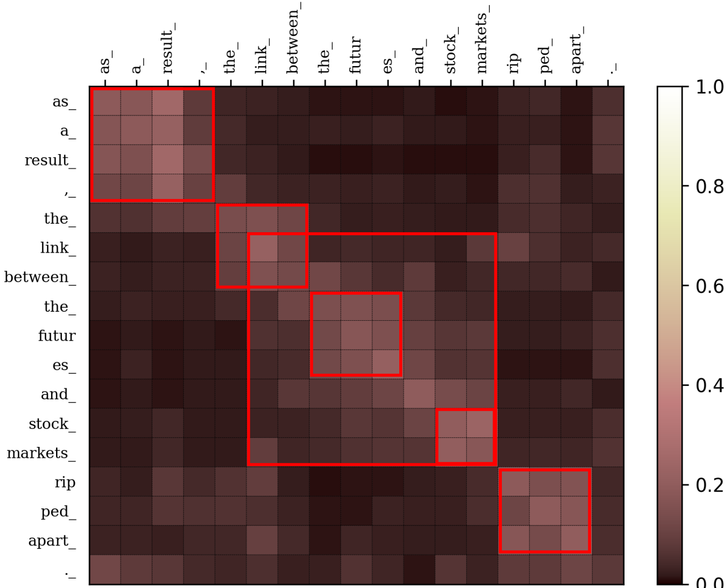
Aggregated attention visualisation - layer 5



Aggregated attention visualisation - layer 6



Something like phrases can be found there...



Getting phrase trees from aggregated attention

- each potential phrase (constituent) gets its score
- score of a constituent with span from position i to position j:

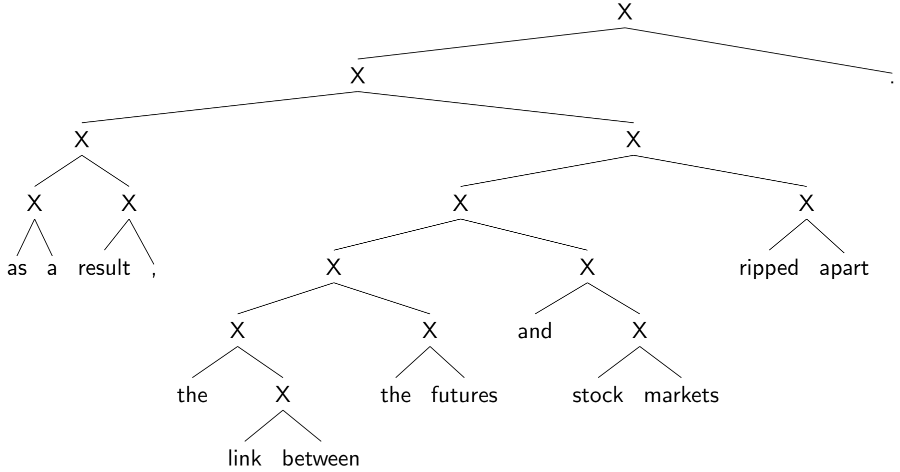
$$score(i,j) = \frac{\sum_{x \in [i,\ldots,j]} \sum_{y \in [i,\ldots,j]} w[x,y]}{j - i + 1}$$

- we build the binary constituency tree by recurrent splitting of a sentence
- each split is made to maximize the scores of both the subtrees
- when splitting the phrase with span (i,j), we are looking for k which maximizes $score(i,k) * score(k+1,j)$

Constraints

- We do not want to split words.
- Each constituent must start and end on the word level, not between two wordpieces in the middle of a word.
- (Trees on the wordpieces would be interesting too, however, we want to compare the trees to annotated treebanks, where everything is done on the word (token) level.)

Getting phrase trees from aggregated attention



Experimental set-up

- We use English sentences from Penn Treebank
- Tokenization conversion needed (brackets, hyphens, ...)
- We use English→German NMT translation using transformer architecture (Vaswani et al., 2017)
- NeuralMonkey toolkit (Helcl and Libovicky, 2017), `https://github.com/ufal/neuralmonkey`
- Dictionary size: 40k, Embedding size: 512, Hidden size: 4096, Number of heads: 16
- Translate the PennTreebank sentences, extract encoder self-attentions for each sentence
- Infer the phrase trees with respect to the original tokens

Evaluation and comparison to unsupervised methods

Metric:

- precision, recall, and F-score on constituents (brackets)

Baselines:

- left-branching
- right-branching
- random baseline

Unuspervised constituency parsing:

- Constituent-context model (Klein and Manning, 2005)
- All subtrees approach (Bod 2007)

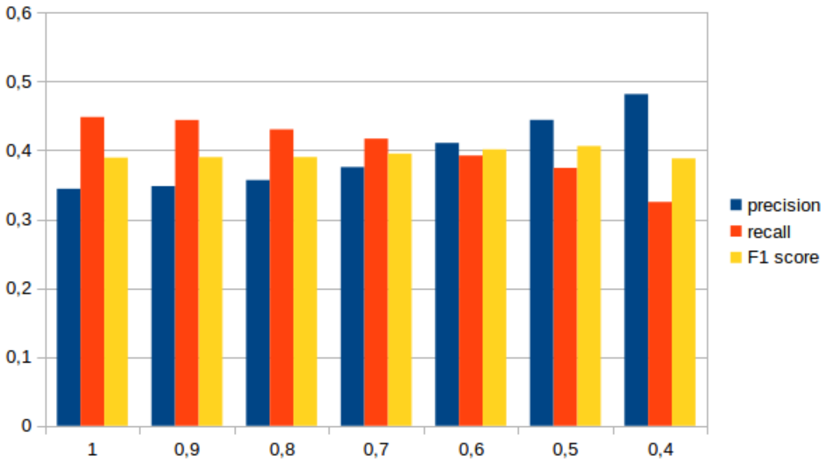
Results

	precision	recall	F1-score	F1-score @10
random baseline	–	–	–	0.35
left baseline	0.10	0.14	0.12	0.29
right baseline	0.31	0.46	0.37	0.56
(Klein and Manning, 2005)	–	–	–	0.78
(Bod, 2007)	–	–	0.66	0.83
Constituents from attentions	0.44	0.37	0.41	0.60

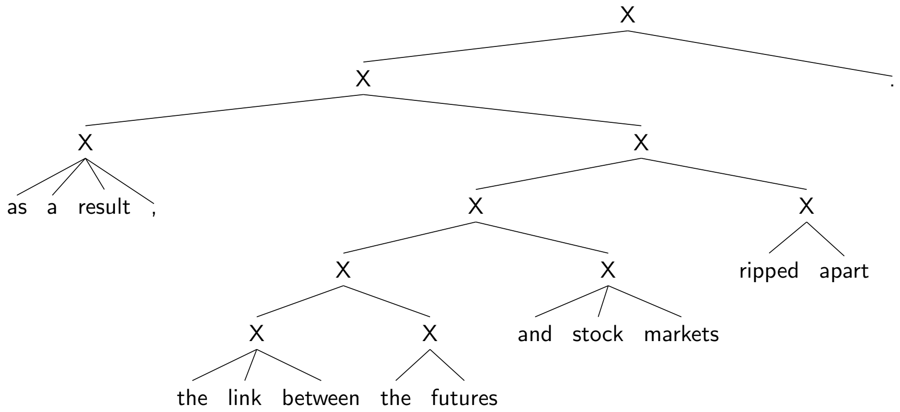
Allowing more than two children

- PennTreebank trees are much more flat that our binary branching trees
- We can change algorithm to be able to split a constituent to more than two parts.
- If $(score(i,k) + score(k+1,j)) * \alpha < score(i,j)$, continue splitting on the same level.

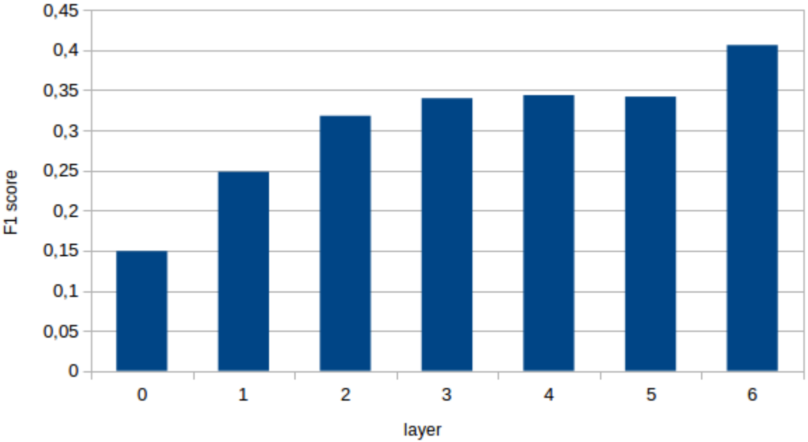
Allowing more than two children – changing α



Allowing more than two children than two – example



Results on lower layers



Future work

- Attention has typically 16 heads on each layer
  - currently we make the average
  - some heads could be better for parsing than the others
  - supervised / unsupervised selection of heads
- Dependencies instead of constituents
- More language pairs