Transformer and Syntax in NMT

Ondřej Bojar

April 16, 2020

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research, Development and Education

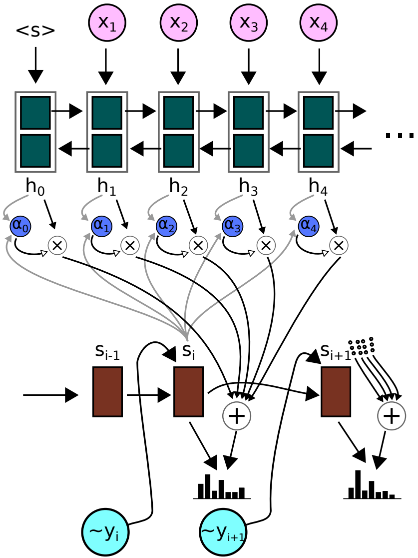
Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Overview

- Reminder: Seq2seq with Attention.
- Transformer Architecture.
    - Focus on Self-Attention.
- Explicit Syntax in NMT.
    - In Network Structure.
    - At Each Token.
    - In Attention.

Some images due to Jindřich Helcl and/or Jindřich Libovický.

Reminder: Seq2seq with Attention



Attention – Formal Notation

Inputs:

decoder state s_i

encoder states $h_j = \left[\overrightarrow{h_j}; \overleftarrow{h_j}\right] \quad \forall i = 1 \ldots T_x$

where $\overrightarrow{h_j} = \text{RNN}_{\text{enc}}(h_{j-1}, x_j) = \tanh(U_e \overrightarrow{h_{j-1}} + W_e E_e x_j + b_e)$

Attention energies: $e_{ij} = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j + b_a\right)$

Attention distribution: $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$

Context vector: $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$

Attention Mechanism in Equations (2)

Decoder state:

$$s_i = \text{tanh}(U_d s_{i-1} + W_d E_d \hat{y}_{i-1} + C c_i + b_d)$$

Output projection:

$$t_i = \text{tanh}\left(U_o s_i + W_o E_d \hat{y}_{i-1} + C_o c_i + b_o\right)$$

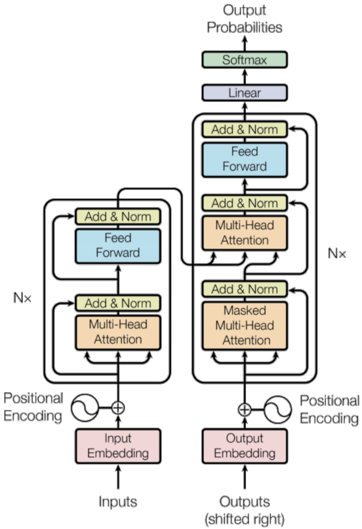
...context vector is mixed with the hidden state

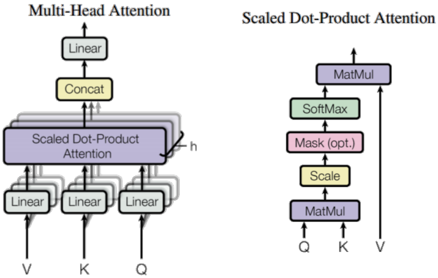
Output distribution:

$$p\left(y_i = k \mid s_i, y_{i-1}, c_i\right) \propto \exp\left(W_o t_i\right)_k + b_k$$

Transformer

Attention is All You Need (Vaswani et al., 2017)





Transformer Detailed Walkthroughs

Transformer Illustrated:

- http://jalammar.github.io/illustrated-transformer/
  Amazingly simple description! (I am reusing the pictures.)

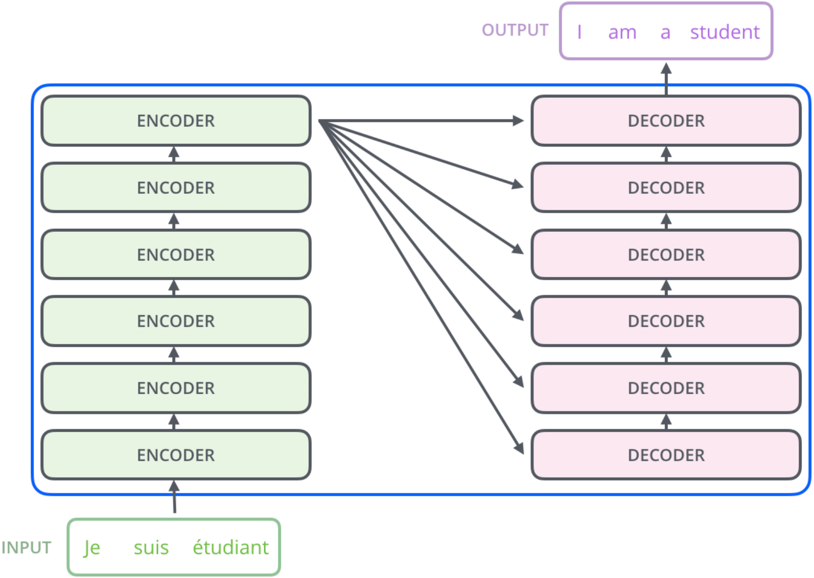
Transformer paper annotated with PyTorch code:

- http://nlp.seas.harvard.edu/2018/04/03/attention.html
- PyTorch by examples:
  https://github.com/jcjohnson/pytorch-examples

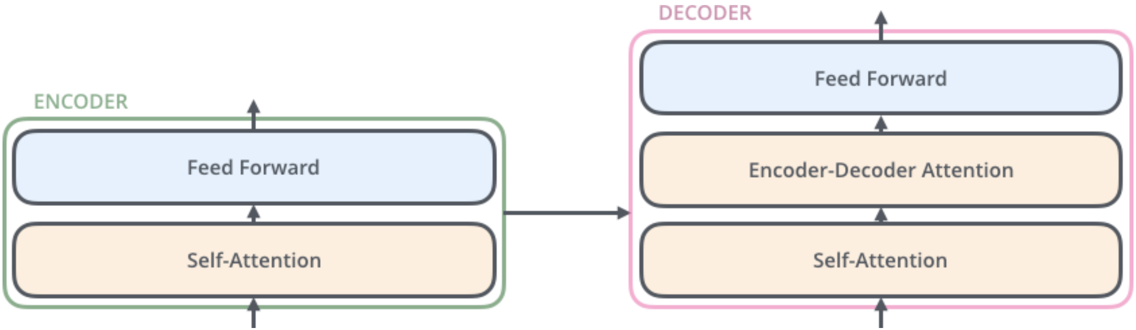
Summary at Medium:

- https://medium.com/@adityathiruvengadam/
  transformer-architecture-attention-is-all-you-need-aecc

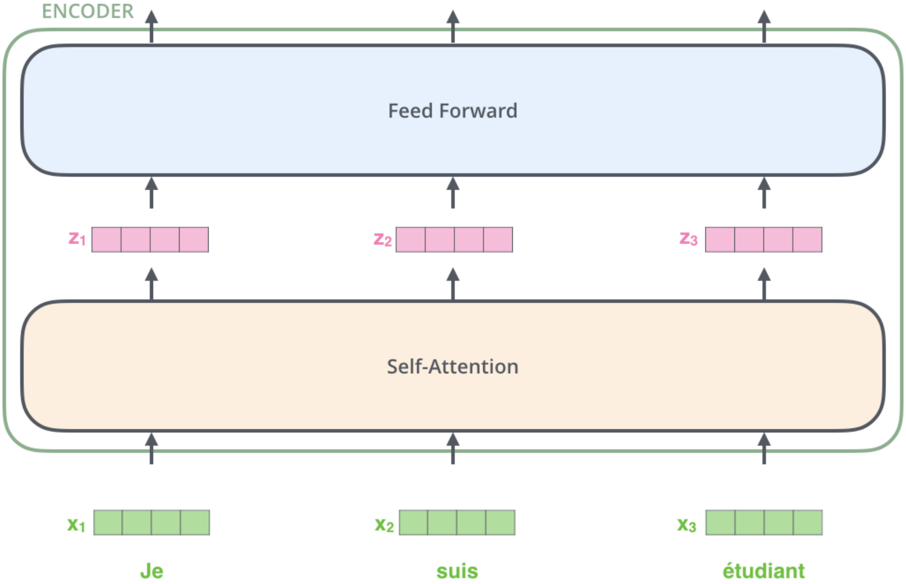
Transformer = 6 Layers Enc + 6 Dec



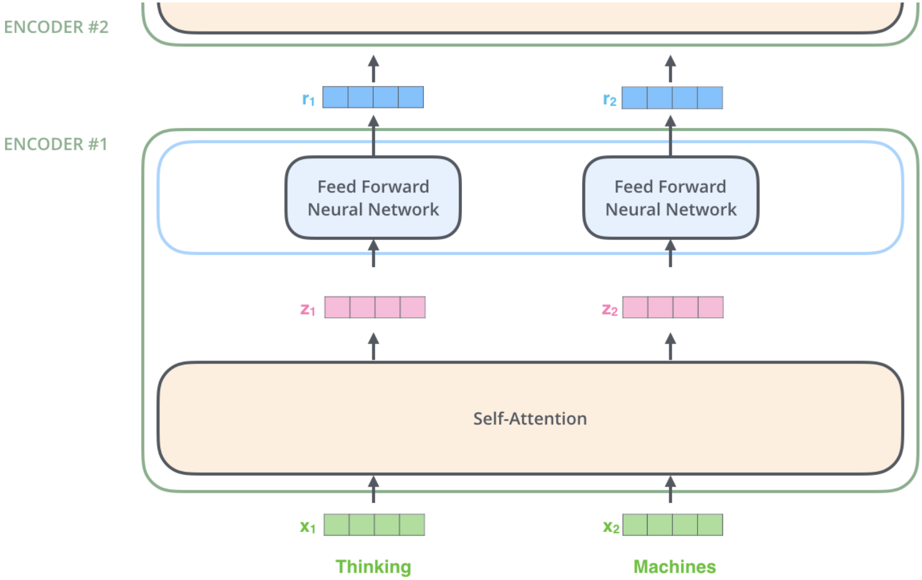
Composition of One Layer



Word Vectors in Encoder

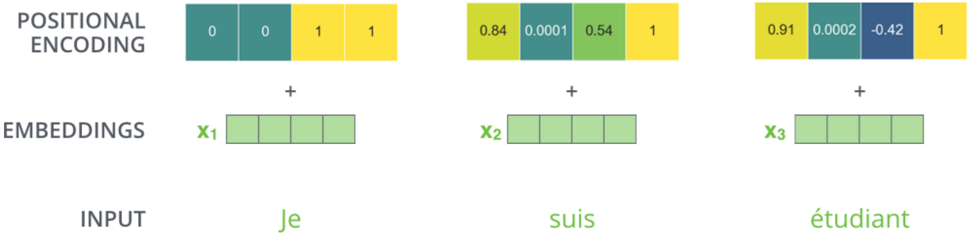


FF Is Actually Position-Independent

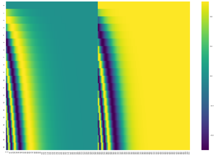


Positional Encoding

- Encode token position directly in the word vector:



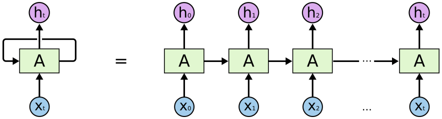
- Positional embedding can be random, or “frequency-like”:



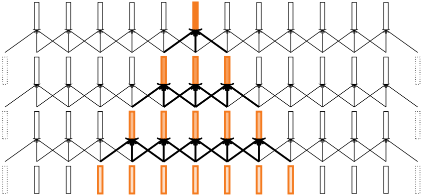
Self-Attention

Self-Attention Motivation (1/2)

- Sequences of arbitrary length n need to be processed.
- RNNs make the (time-unrolled) network as deep as n.



- CNNs allow to trade kernel size k and depth for a target "receptive field":



$x_0 = \vec{0}$ embeddings $\mathbf{x} = (x_1, \ldots, x_N)$ $x_n = \vec{0}$

Self-Attention Motivation (2/2)

- SANs (Self-Attentive Networks) can access any position in constant time.

	Operations	Sequential Steps	Memory
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n \cdot d)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(n \cdot d)$
Self-attentive	$O(n^2 \cdot d)$	$O(1)$	$O(n^2 \cdot d)$

- Sequence length n, state dimensionality d, kernel size k.
- Assuming infinitely many GPU cores (or rather ALU), operations can be run in parallel, but may depend on each other, needing some Sequential Steps.

Self-Attention

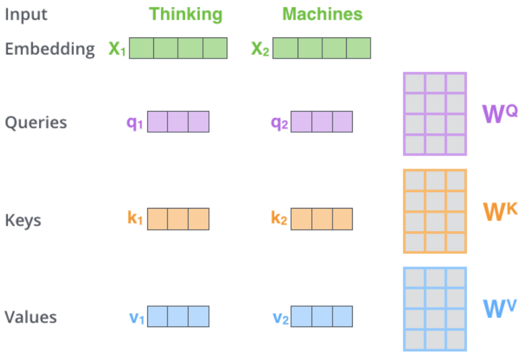
- Goal: Aggregate arbitary-length input to fixed-size vector.
  Allow data-driven, trainable aggregation.

Self-Attention

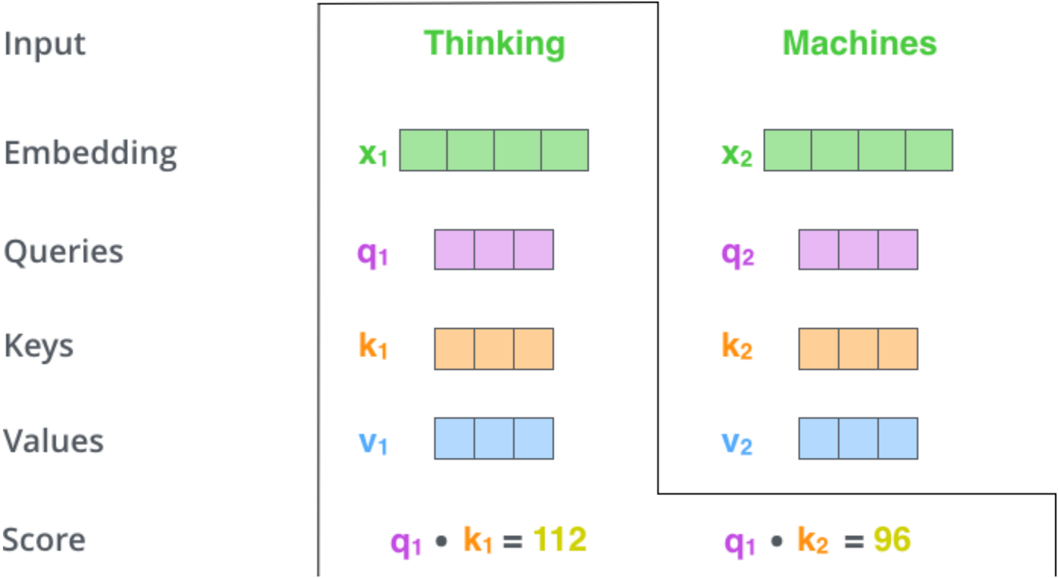
- Goal: Aggregate arbitary-length input to fixed-size vector.
  Allow data-driven, trainable aggregation.

Given the sequence of inputs $x_1, \ldots, x_n$:

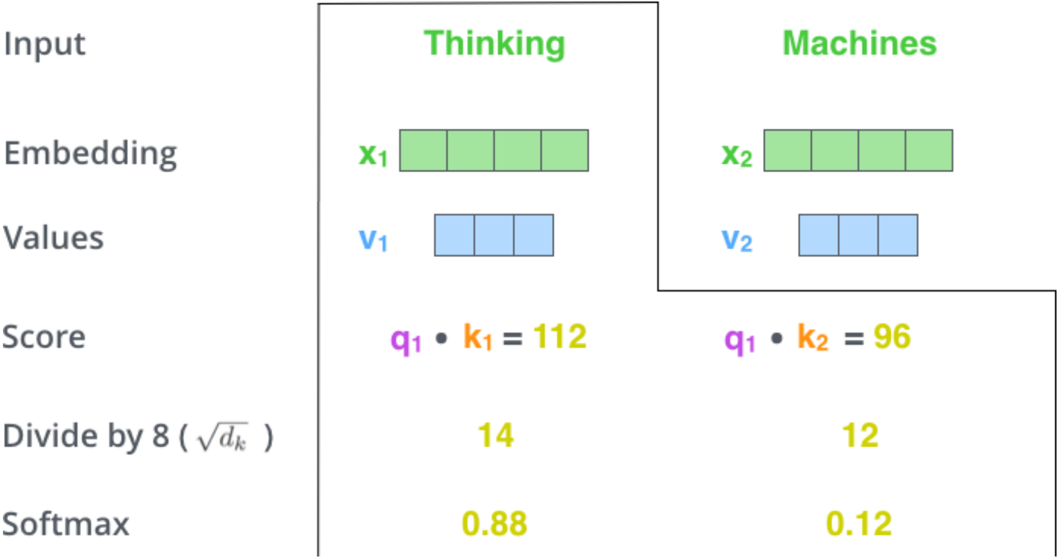
- Create three "views" of them: queries, keys, values.
- Using trained matrices W^Q, W^K, W^V.



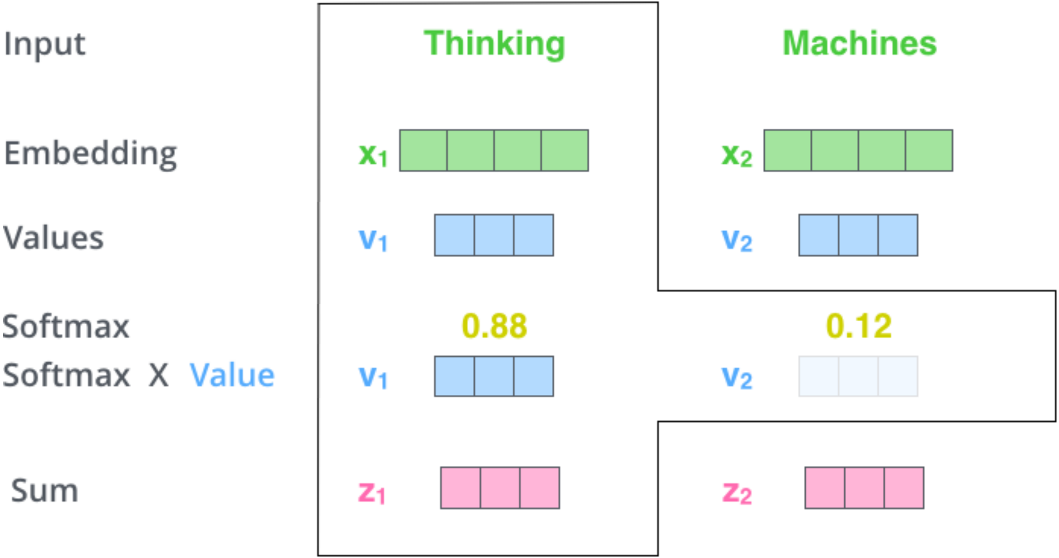
Match All Queries with All Keys



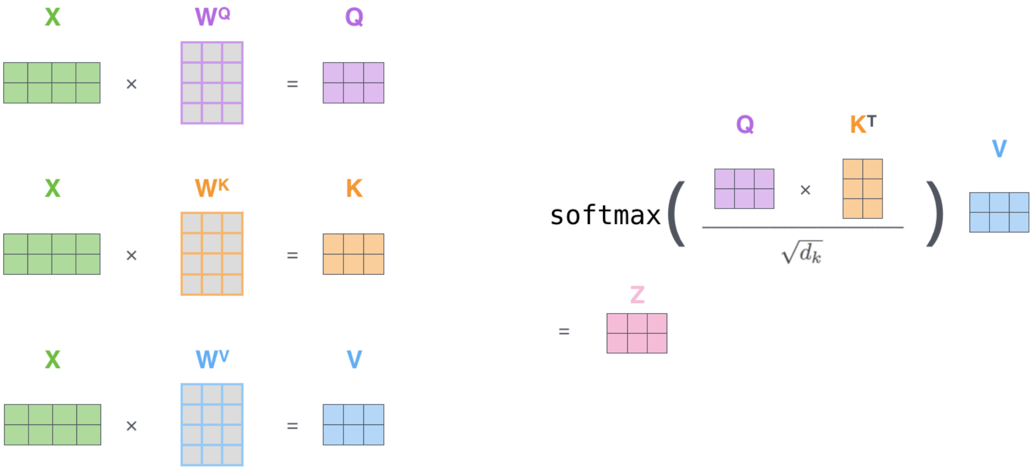
Normalize Scores



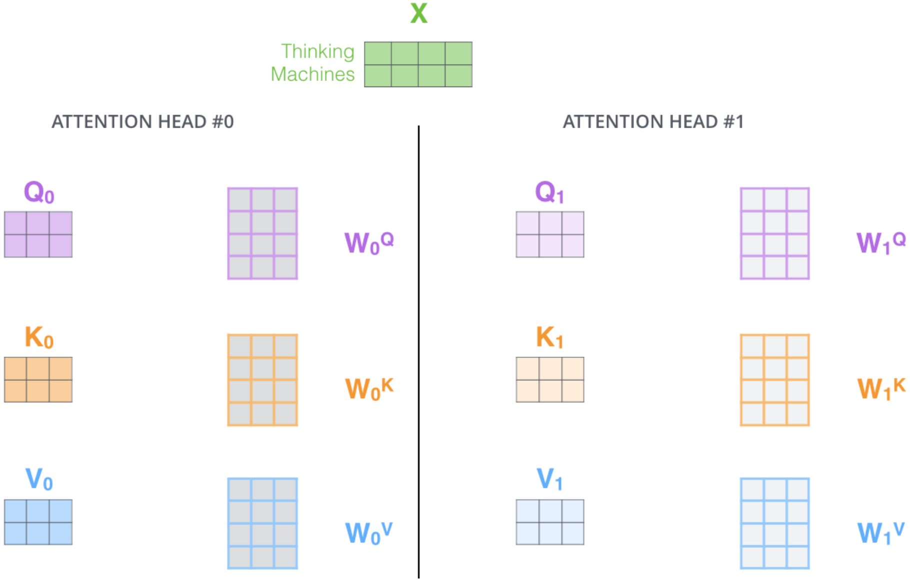
Aggregate Values Accordingly



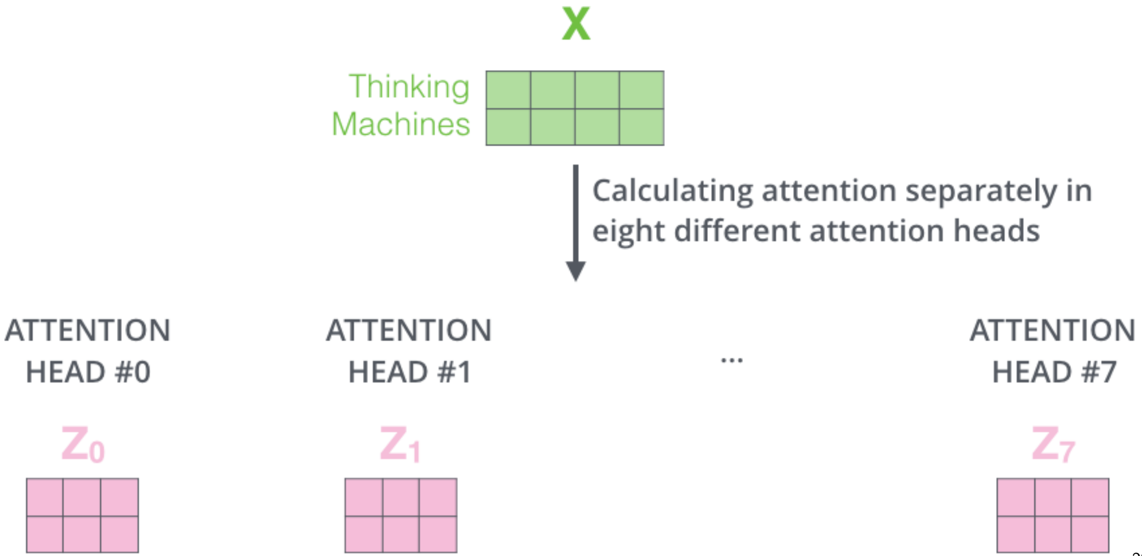
Self-Attention as Matrix Calculation



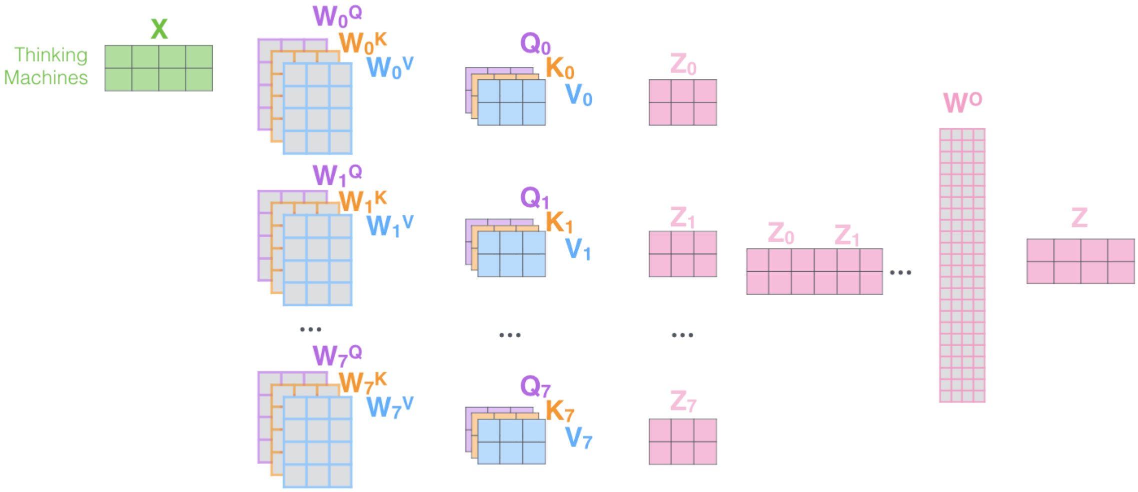
Multi-Head Attention



Multi-Head Attention



Self-Attention Summary



Self-Attention in Transformer

Three uses of multi-head attention in Transformer

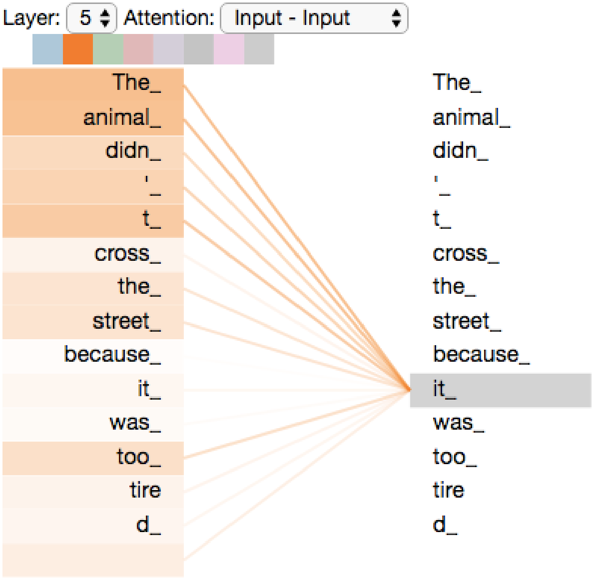
- Encoder-Decoder Attention:
  - Q: previous decoder layers; K $=$ V: outputs of encoder

  $\Rightarrow$ Decoder positions attend to all positions of the input.
- Encoder Self-Attention:
  - Q $=$ K $=$ V: outputs of the previous layer of the encoder

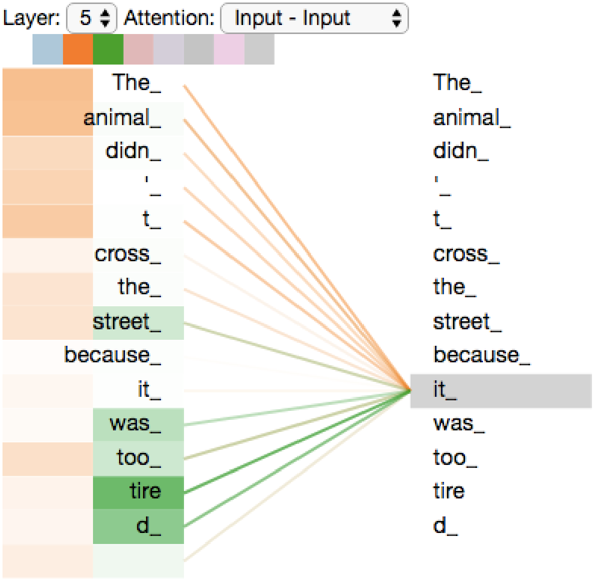
  $\Rightarrow$ Encoder positions attend to all positions of previous layer.
- Decoder Self-Attention:
  - Q $=$ K $=$ V: outputs of the previous decoder layer.
  - Masking used to prevent depending on future outputs.

  $\Rightarrow$ Decoder attends to all its previous outputs.

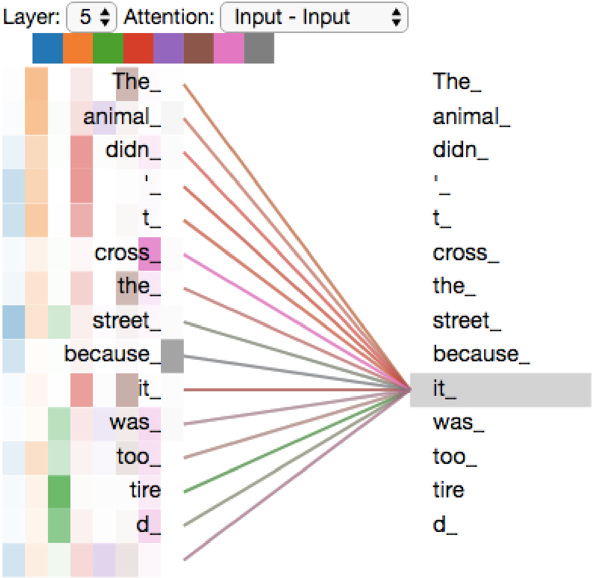
Self-Attention at Enc Layer #5: 1 Head



Self-Attention at Enc Layer #5: 2 Heads



Self-Attention at Enc Layer #5: 8 Heads



Explicit Linguistic Information in NMT

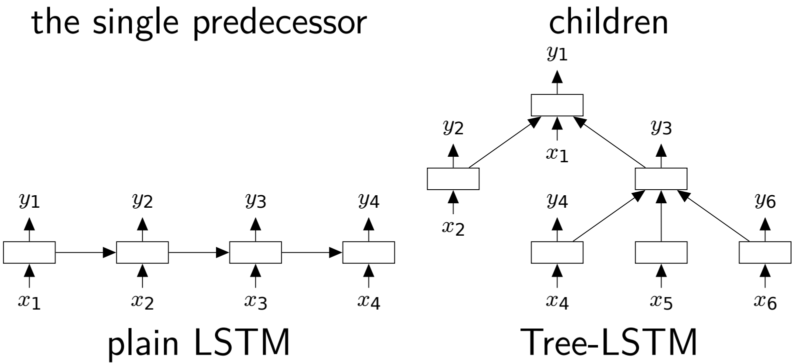
Ways of Adding Linguistic Annotation

- Construct network structure along linguistic structure.
  - $\sim$ What we discussed in Syntax in SMT.
  - Tree-LSTMs.
  - Graph-Convolutional Networks.
  - ... Source information only.
- Enrich information at each token.
  - $\sim$ What we discussed in Morphology in SMT.
  - Factors on the source side.
  - Multi-Task on the target side.
- Improve attention using linguistic annotation.
  - Attention calculation respecting syntax.
  - Attention forced to reflect syntax in multi-task.

Linguistics in NN Structure

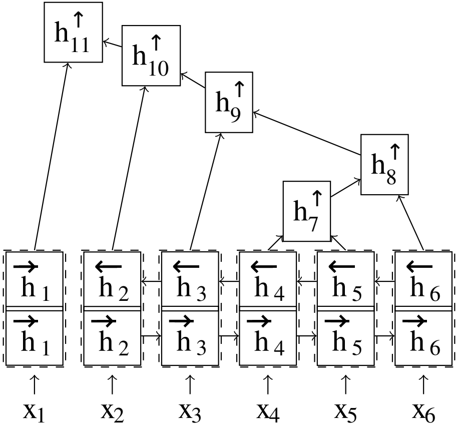
Tree-LSTMs (Tai et al., 2015)

Memory comes from:



- Two flavors:
  - Dependency trees: Sum over all children.
  - Constituency trees: Up to N children, respecting order.

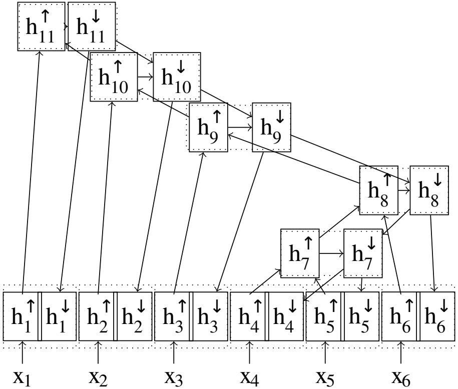
Chen et al. (2017a) (1/2)



Tree-GRU Encoder:

- Constituency syntax of the tree provides additional states.

Chen et al. (2017a) (2/2)



Bidirectional tree encoder.

- Can be seen as many RNNs running from each word up to the root and back to the word.

Graph-Convolutional Networks (Bastings et al., 2017)

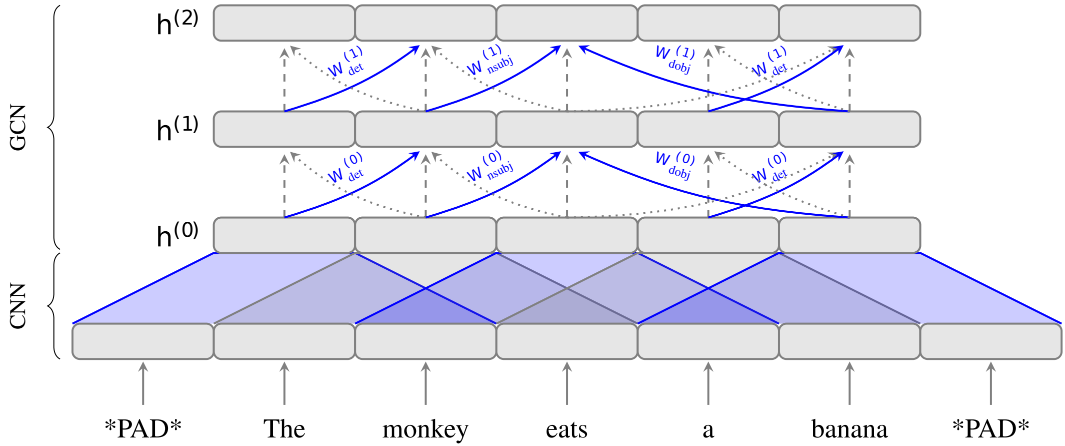


Figure 2: A 2-layer syntactic GCN on top of a convolutional encoder. Loop connections are depicted with dashed edges, syntactic ones with solid (dependents to heads) and dotted (heads to dependents) edges. Gates and some labels are omitted for clarity.

Linguistics at Each Token

CCGs to Encode Syntax at Each Token

Syntax reflects long-distance dependencies.

- What city is the Taj Mahal in?
- Where is the Taj Mahal $\emptyset$?

The need to produce in depends on the *What*/*Where*.

- CCG tags for is differ $\Rightarrow$ dependency highlighted.
- Following CCG tags, the decoder can know if *in* is needed.
- CCG tags are denser than words $\Rightarrow$ better generalization.

CCGs to Encode Syntax at Each Token

Syntax reflects long-distance dependencies.

- What city is the Taj Mahal in?
- Where is the Taj Mahal $\emptyset$?

The need to produce in depends on the *What*/*Where*.

- CCG tags for is differ $\Rightarrow$ dependency highlighted.
- Following CCG tags, the decoder can know if *in* is needed.
- CCG tags are denser than words $\Rightarrow$ better generalization.

- What$_{\text{(S[wq]/(S[q]/NP))/N}}$ city is$_{\text{(S[q]/P P)/NP}}$ the Taj Mahal in?
- Where$_{\text{S[wq]/(S[q]/NP)}}$ is$_{\text{(S[q]/NP)/NP}}$ the Taj Mahal?

Factors in NMT

- Source word factors easy to incorporate:
  - Concatenate embeddings of the various factors.
  - POS tags, morph. features, source dependency labels help en↔de and en→ro (Sennrich and Haddow, 2016).
- Target word factors:
  - Interleave for morphology: (Tamchyna et al., 2017)

Src	there are a million different kinds of pizza .
Baseline (BPE)	existují miliony druhů piz@@ zy .
Interleave	VB3P existovat NNIP1 milion NNIP2 druh NNFS2 pizza Z: .

  - Interleave for syntax: (Nadejde et al., 2017)

Src BPE	Obama receives Net+ an+ yahu in the capital of USA
Tgt	NP Obama ((S[dcl]\NP)/PP)/NP receives NP Net+ an+ yahu PP/NP in NP/N the N cap

  - Multiple decoders, each predicting own sequence.

Predicting Target Syntax

My students Dan Kondratyuk and Ronald Cardenas retried Nadejde et al. (2017) with:

- sequence-to-sequence model,
- Transformer model.

Predicting target syntax using:

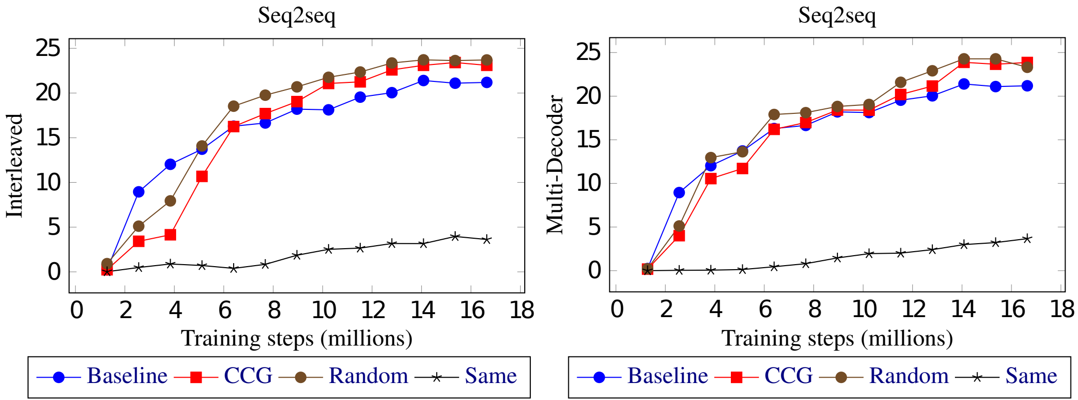
- a secondary decoder
- interleaving.

As tags, they used:

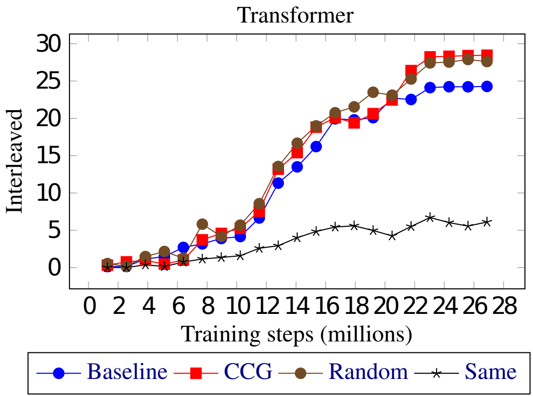
- correct CCG tags, • random tags, • a single dummy tag.

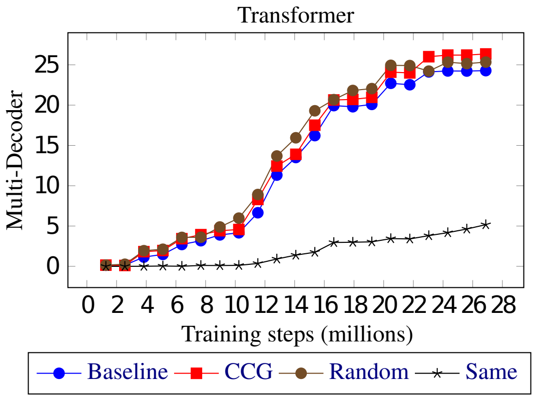
(Kondratyuk et al. Replacing Linguists with Dummies. PBML 2019.)

Predicting Target Syntax (S2S)

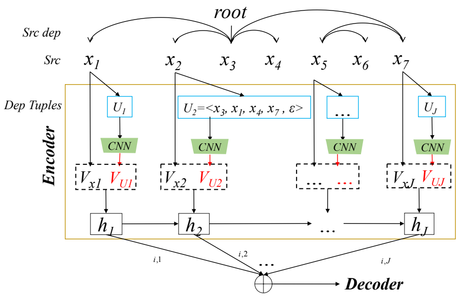


Predicting Target Syntax (Transformer)





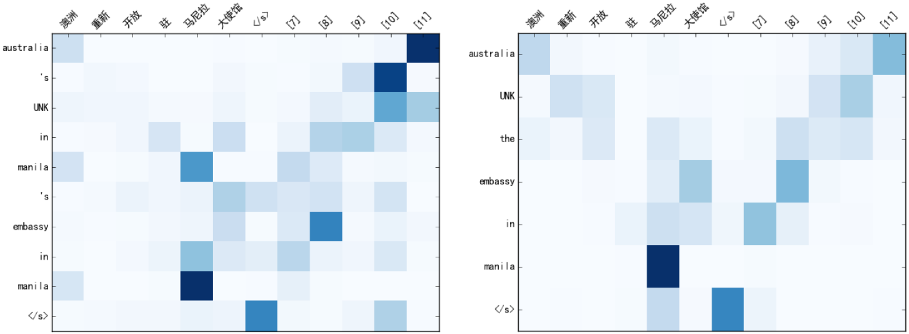
Syntax in Word Embeddings Chen et al. (2017b)



- CNN-derived embeddings of nodes' syntactic neighbourhood included: (parent, siblings).
- Two mechanisms:
    - Concatenated to standard embeddings.
    - Separate attention over these word-level annotations

Linguistics in Attention

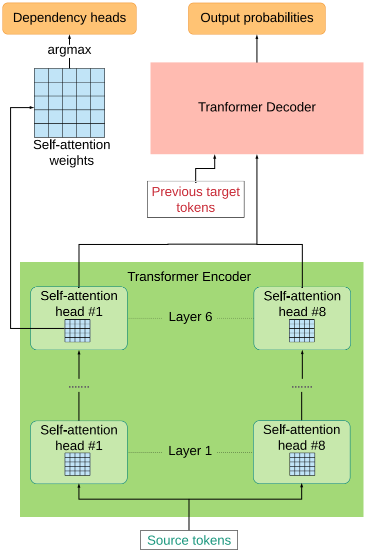
Tree Coverage in Attention Chen et al. (2017a)



Tree coverage model:

- Attention coverage depends on source syntax.
- Without it (left), output is repeated.

Multi-Task to Request Dependency Tree (1/2)



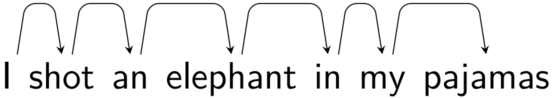
- Pham et al. (2019) noticed that attention head could be interpreted as dependency parse.
- Add secondary objective to require head #1 to match source dependency tree.

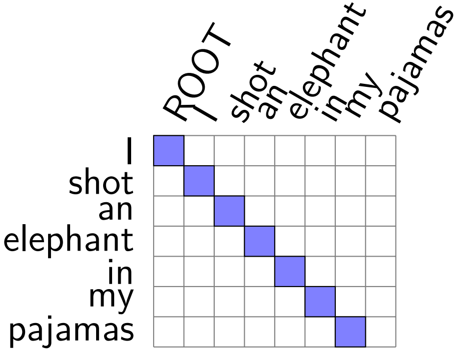
Multi-Task to Request Dependency Tree (2/2)

- Czech-to-English translation (BLEU).
- Czech dependency parse from head #1 (UAS).

	BLEU		**UAS**	
	Dev	**Test**	**Dev**	**Test**
Transformer Baseline	37.28	36.66	–	–
Parse from layer 0	36.95	36.60	81.39	82.85
Parse from layer 1	**38.51**	**38.01**	90.17	90.78
Parse from layer 2	38.50	37.87	91.31	91.18
Parse from layer 3	38.37	37.67	91.43	91.43
Parse from layer 4	37.86	37.60	**91.65**	**91.56**
Parse from layer 5	37.63	37.67	91.44	91.46

Dummy Dependency Tree





Multi-Task to Request Dummy Tree

	BLEU		Precision	
	Dev	Test	Dev	Test
Transformer Baseline	37.28	36.66	–	–
Dummy Parse from layer 0	38.68	**38.14**	99.97	99.96
Dummy Parse from layer 1	**39.11**	38.06	**99.99**	**99.99**
Dummy Parse from layer 2	37.85	37.85	99.98	99.98
Dummy Parse from layer 3	37.93	37.70	99.97	99.98
Dummy Parse from layer 4	37.68	37.47	99.98	99.96
Dummy Parse from layer 5	37.53	37.54	99.96	99.95
True Parse from layer 1	**38.51**	**38.01**	90.17	90.78

Summary

- Transformer is a great replacement for RNN.
  - Constant-time processing.
  - (CNNs *can* be comparable, but Gehring et al. (2016) was kind of missed.)
- Explicit syntax can be useful.
  - Many options how to include it.
  - Some gains hard to reproduce.
  - Dummy information can be equally useful.
  - Transformer seems to learn syntax for free.

References

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Simaan. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 1947–1957. Association for Computational Linguistics.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017a. Improved Neural Machine Translation with a Syntax-Aware Encoder and Decoder. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1936–1945, Vancouver, Canada, July. Association for Computational Linguistics.

Kehai Chen, Rui Wang, Masao Utiyama, Lemao Liu, Akihiro Tamura, Eiichiro Sumita, and Tiejun Zhao. 2017b. Neural Machine Translation with Source Dependency Representation. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 2846–2852, Copenhagen, Denmark, September. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. 2016. A convolutional encoder model for neural machine translation. CoRR, abs/1611.02344.

Maria Nadejde, Siva Reddy, Rico Sennrich, Tomasz Dwojak, Marcin Junczys-Dowmunt, Philipp Koehn, and Alexandra Birch. 2017. Predicting target language ccg supertags improves neural machine translation. In Proceedings of the Second Conference on Machine Translation, Volume 1: Research Paper, pages 68–79, Copenhagen, Denmark, September. Association for Computational Linguistics.

Thuong-Hai Pham, Dominik Macháček, and Ondřej Bojar. 2019. Promoting the knowledge of source syntax in transformer nmt is not needed. Computación y Sistemas, 23(3):923–934.

Rico Sennrich and Barry Haddow. 2016. Linguistic input features improve neural machine translation. In Proceedings of the First Conference on Machine Translation, pages 83–91, Berlin, Germany, August. Association for Computational Linguistics.