Natural language syntax: parsing and complexity

Timothée Bernard and Pascal Amsili

Université Paris Cité, Université Sorbonne Nouvelle
timothee.bernard@u-paris.fr, pascal.amsili@ens.fr

Ljubljana, Slovenia – August 7-11, 2023
ESSLLI foundational course in Language and Computation

Overview of the course

- Day 1: Formal languages and syntactic complexity.
- Day 2: The complexity of natural language.
- Day 3: Historic algorithms for parsing.
- Day 4: Modern approaches to parsing.
- Day 5: Neural networks and error propagation.

Day 5

Recap from Day 4

- An ML component (classifier/scorer) can replace a grammar.
- This is possible both in transition and chart parsing.
- Doing so requires data (usually treebanks).
- Dependency parsing is popular; either transition- or graph-based.
- For CCG and TAG: the supertagging+decoding paradigm combines a classifier and a grammar.
- Grammar-based parsing can be convenient for compositional semantics.

Today's contents

- The basics of neural networks.
- How neural networks in general are trained.
- How neural parsers in particular are trained.
- Why and how token embeddings are useful.
- Error propagation in transition parsers and how to fight it.

Machine learning is used to infer functions

- Today: heavy use of machine learning in parsing to power classifiers/scorers.
- **Machine learning** (ML): computational inference of functions from data.
- Goal: good enough approximation of usually fairly abstract and unknown functions.
  Ex: image classification, offensive language detection
- In practice: generalisable approximation of a finite [raw data $\mapsto$ annotation] function.
- Kinds of functions: classification functions, clustering functions, real-valued functions. . .

Neural networks work with vectors

- Neural network:
  - one of the most popular ML frameworks;
  - based on vector transformation.
- **Vector** (in ML): sequence of values of fixed length
  Ex: $[2.1, 0.33, -9.25, 0.0, 1.1] \in \mathbb{R}^5$
- One n-input neuron: $\mathbb{R}^n \to \mathbb{R}$
- m n-input neurons: $\mathbb{R}^n \to \mathbb{R}^m$
  $\to$ **layer** of width m
- A **neural network** (NN) may be seen as a graph (DAG) of layers.

In the beginning are word embeddings

- Input vectors?
- For most layers, output vectors of previous layers.
- Otherwise: vector representations of (all or part of) the input of the problem.
- In NLP: **word embeddings** represent tokens.
  - Word2Vec (Mikolov et al. 2013)
  - GloVe (Pennington, Socher & Manning 2014)
  - fastText (Bojanowski et al. 2017)
  - BERT (Devlin et al. 2019)

Linear layers are one of the basic building blocks of NNs

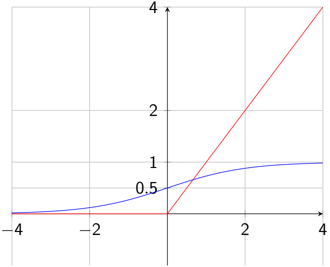
n-input "linear" neuron

- parameters: $\theta = ((a_i)_{1 \leq i \leq n}, b) \in \mathbb{R}^n \times \mathbb{R}$
- input: $u = (u_i)_{1 \leq i \leq n} \in \mathbb{R}^n$
- output: $(\sum_{i=1}^{n} a_i \times u_i) + b \in \mathbb{R}$

- "Linear" layer of width m: m linear neurons in parallel.
- In general, one wants to approximate a *non-linear* function.
  $\rightarrow$ NNs contain many linear layers followed by non-linearity functions.

Element-wise non-linearities: ReLU and σ

- Rectified linear unit (ReLU): $x \mapsto \max(x, 0)$
- Sigmoid (σ): $x \mapsto \frac{1}{1+\exp(-x)}$



- $\rightarrow$ applied element-wise to get non-linear layers
- Often found after every linear layer save the last one.

Softmax is used to generate probability distribution

- Neural classifiers produce probability distributions.
- Done by ending with a **softmax** layer.

Softmax

- input: $u = (u_i)_{1 \leq i \leq n} \in \mathbb{R}^n$
- output: $p = (\frac{\exp(u_i)}{\sum_{j=1}^{n} \exp(u_j)})_{1 \leq i \leq n} \in \mathbb{R}^n$

$\rightarrow$ p describes a probability distribution s.t. if $u_i \leq u_j$ then $p_i \leq p_j$

There is an NN design for every situation

- By stacking linear and non-linear layers on top of each other, one can build a very expressive $\mathbb{R}^n \to \mathbb{R}^m$ network.
- Other modules (averaging, LSTM, attention, etc.):
  - from a variable-length sequence of vectors to a single vector;
  - from a single vector to a variable-length sequence of vectors;
  - from a variable-length sequence of vectors to a sequence of same length;
  - ...
- All defined in terms of small set of fairly simple vector operations.
- From simplicity emerges complexity.

Basic NN design is dictated by the flow of information

- First stages of a typical NLP system:
  - conversion of the input text into a sequence of (static) word embeddings;
  - enrichment into *contextual* word embeddings.
- To represent a stack: use a variable-length sequence to single vector module.
- (idem for a buffer)
- To represent a parsing state: concatenation of relevant vectors.
- To represent a candidate dependency: concatenate the embeddings of the two tokens.
- ...

The importance of representations cannot be overstated

- For most NLP tasks (e.g. parsing), relatively little annotated data is available compared to the complexity of the task.
- Consequence: overall performance is *highly* dependant on the word representations used (i.a. Pennington, Socher & Manning 2014, Peters et al. 2018, Devlin et al. 2019).
- Past: lexicons with rich symbolic lexical features
- Now: token representations obtained from statistical/neural methods trained on massive data (see slide 8).

Lexical and sentential information is found in embeddings

- Word embeddings often contain morpho-syntactic and semantic information (i.a. Köhn 2015, Gupta et al. 2015, Gaddy, Stern & Klein 2018).
- Contextual embeddings in particular also contain information about syntax and sentence semantics (Tenney et al. 2019).
- Neural language models (e.g. BERT):
  - mainly used to produce context embeddings;
  - trained with a language modelling task (i.e. word prediction in context);
  - implicitly learn to perform many NLP tasks? (Tenney, Das & Pavlick 2019)

Designing NNs that train easily is an art

- NNs are often trained by **gradient descent**.
- Intuitively: repeatedly tweaking a little bit the parameters so that the output gets closer to the intended value (i.e. in the direction of the **gradient**).
- **Loss**: how far the output is from the intended value.
  $\rightarrow$ usually, more than one possibility
- (The parameters can be initialised randomly.)
- Many design choices only aim at making this process possible and effective.

Make the score/probability of the gold tree the highest

- Here are simple losses for some of the parsers mentioned during Day 4.
- Transition-based parser: opposite of the probability of a gold derivation (i.e. product of the probability of each action).
- Graph-based parser: difference between the score of the predicted tree (i.e. sum of the score of each dependency) and the score of the gold tree.
- Scorer-based CYK parser: similar.
- Many improvements are possible.

Overfitting is a common plague in ML

- Today's NNs have relatively many parameters compared to the size of the training data.
- Consequence: the loss can be very low (on the train. data), yet the performance can be unsatisfying (on the test. data). $\rightarrow$ **overfitting** (general problem in ML)
- General solutions: *regularisation* methods (e.g. L1/L2, dropout).
- Relatively little work on *data augmentation* and *domain adaptation* for parsing (but see e.g. Baucom, King & Kübler 2013).

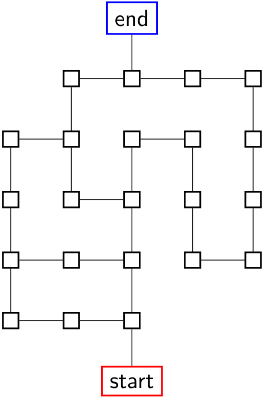
Another plague for sequential systems: error propagation

- **Error propagation**: when one mistake leads to more mistakes.
- Common in transition-based parsing.
- One reason: errors lead to unusual states (different from the ones seen at training) $\rightarrow$ classifiers/scorers become unreliable
- Can be fought at training and/or testing time.
- Example in the *structured perceptron* paradigm (Collins 2002): not only strengthen a gold derivation, but also weaken the predicted one.
  $\rightarrow$ deviation from teacher forcing
  (this idea alone does not define the paradigm)

Explore multiple trajectories with beam search

- **Beam search**:
  - introduced by Lowerre (1976) for speech recognition;
  - heavily used in transition parsing;
  - consists in exploring multiple derivations in parallel.
- Beam of size k:
  - stores up to k distinct partial derivations (AKA **hypotheses**);
  - at each step,
    - look at all possible continuations of all current hypotheses,
    - select the k best ones overall as the new content of the beam;
  - at the end, return the best derivation in the beam.

Illustration of beam search with a maze (introduction)



- Edges of equal length.
- Goal: find a short path in the maze (discovered in the process).
- Rule: the same vertex cannot be visited more than once.
  $\rightarrow$ no backtracking
- Comparison of greedy search vs beam search ($k = 2$).

Illustration of beam search with a maze (greedy search)

Illustration of beam search with a maze (greedy search)

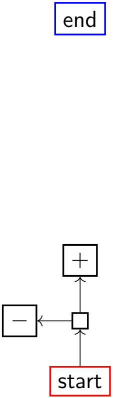


Illustration of beam search with a maze (greedy search)

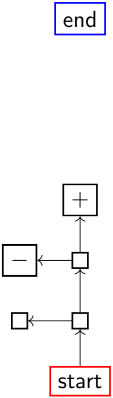


Illustration of beam search with a maze (greedy search)

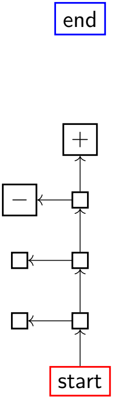


Illustration of beam search with a maze (greedy search)

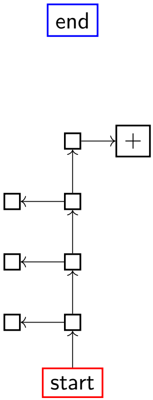


Illustration of beam search with a maze (greedy search)

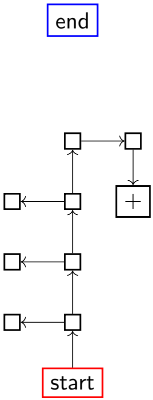


Illustration of beam search with a maze (greedy search)

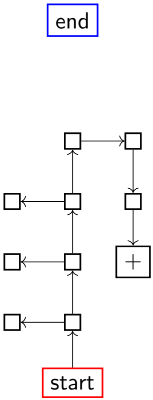


Illustration of beam search with a maze (greedy search)

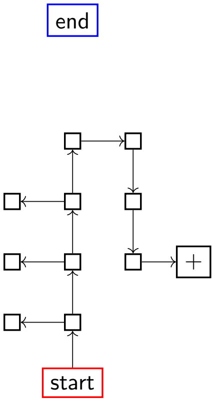


Illustration of beam search with a maze (greedy search)

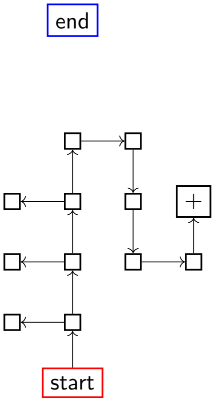


Illustration of beam search with a maze (greedy search)

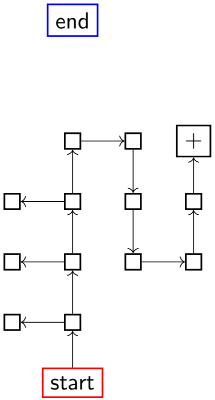


Illustration of beam search with a maze (greedy search)

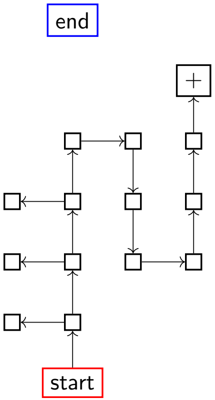


Illustration of beam search with a maze (greedy search)

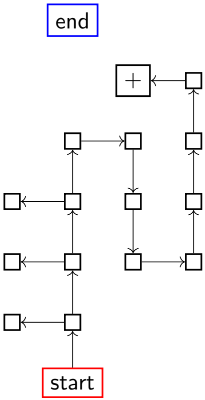


Illustration of beam search with a maze (greedy search)

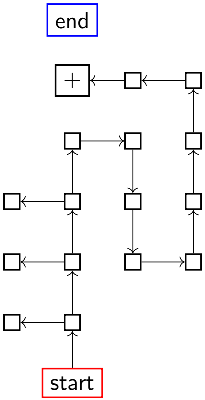


Illustration of beam search with a maze (greedy search)

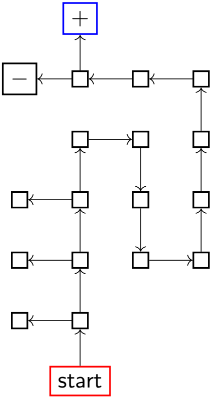


Illustration of beam search with a maze (beam search)

Illustration of beam search with a maze (beam search)

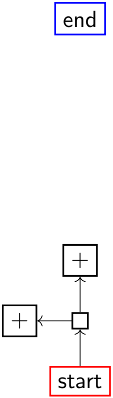


Illustration of beam search with a maze (beam search)

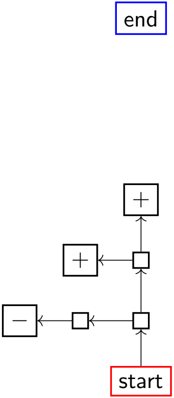


Illustration of beam search with a maze (beam search)

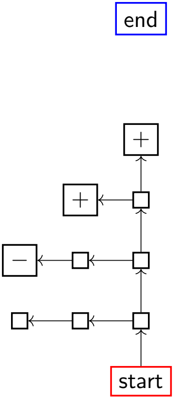


Illustration of beam search with a maze (beam search)

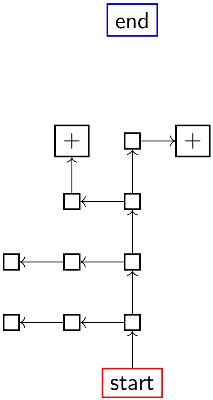


Illustration of beam search with a maze (beam search)

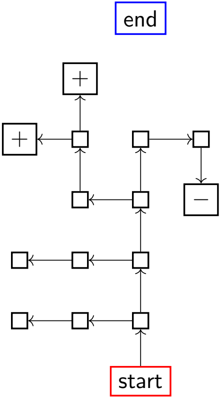


Illustration of beam search with a maze (beam search)

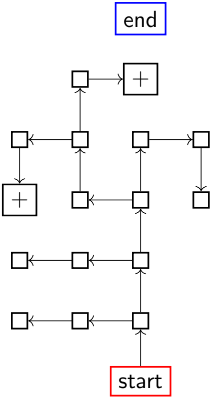
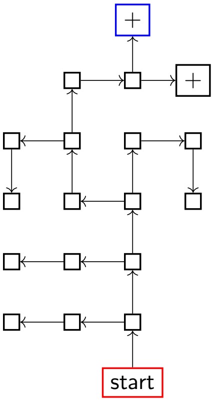


Illustration of beam search with a maze (beam search)



Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
b. The

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
    b. The horse

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
    b. The horse raced

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
    b. The horse raced past

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
b. The horse raced past the

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
    b. The horse raced past the barn

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
    b. The horse raced past the barn fell

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
    b. The horse raced past the barn fell.

Test and train with a beam for maximum efficiency

- Beam search tends to make parsers more robust.
- Think about garden path sentences:

(1) a. After the student moved the chair broke.
    b. The horse raced past the barn fell.

- There are several strategies to improve *training* when using a beam (Collins & Roark 2004, Huang, Fayong & Guo 2012).

Forget about gold derivations

- **Dynamic oracle** (DO; Goldberg & Nivre 2012):
  - given any parsing state, what are the best actions?
  - known for some (i.e. Arc-Eager) but not all (i.e. Arc-Standard) transition systems;
  - used to train a parser on its own *trajectories*.
- **Reinforcement learning** (RL; Sutton & Barto 2018, Lê & Fokkens 2017):
  - define a reward system (i.e. action taken $\mapsto$ scalar reward);
  - run the system on training data;
  - a loss is defined in terms of the rewards;
  - $\rightarrow$ by minimising the loss, the expected sum of rewards is maximised.

DO and RL can lead to more data-efficient training

- training on a system's own trajectories
  $\rightarrow$ makes the training and testing distributions (of states) more similar
  $\rightarrow$ makes ML components more reliable/less prone to error propagation
- Rmk: For the same annotated data, there are many more possible trajectories than gold trajectories.

Day 5: Summary

- Today, NNs are heavily used in parsing as classifiers/scorers.
- NNs work by combining (many) simple linear and non-linear vector transformations.
- Parsing performance depends on the “quality” of token representations.
- Error propagation is a plague for many transition parsers.
- It can be fought by decoding with a beam and/or by deviating from teacher forcing.

- Baucom, Eric, Levi King & Sandra Kübler. 2013. Domain Adaptation for Parsing. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, 56–64. Hissar, Bulgaria: INCOMA Ltd. Shoumen, BULGARIA. https://aclanthology.org/R13-1008.
- Bojanowski, Piotr, Edouard Grave, Armand Joulin & Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5. 135–146. https://doi.org/10.1162/tacl_a_00051.
- Collins, Michael. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, 1–8. Association for Computational Linguistics. https://doi.org/10.3115/1118693.1118694.
- Collins, Michael & Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Annual*

Meeting of the Association for Computational Linguistics (ACL-04), 111–118. Barcelona, Spain. https://doi.org/10.3115/1218955.1218970.

- Devlin, Jacob, Ming-Wei Chang, Kenton Lee & Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423.
- Gaddy, David, Mitchell Stern & Dan Klein. 2018. What's Going On in Neural Constituency Parsers? An Analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 999–1010. New

Orleans, Louisiana: Association for Computational Linguistics. https://doi.org/10.18653/v1/N18-1091.

- Goldberg, Yoav & Joakim Nivre. 2012. A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of COLING 2012*, 959–976. Mumbai, India: The COLING 2012 Organizing Committee. https://www.aclweb.org/anthology/C12-1059.
- Gupta, Abhijeet, Gemma Boleda, Marco Baroni & Sebastian Padó. 2015. Distributional vectors encode referential attributes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 12–21. Lisbon, Portugal: Association for Computational Linguistics. https://doi.org/10.18653/v1/D15-1002.
- Huang, Liang, Suphan Fayong & Yang Guo. 2012. Structured Perceptron with Inexact Search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*,

142–151. Montréal, Canada: Association for Computational Linguistics. https://aclanthology.org/N12-1015.

- Köhn, Arne. 2015. What's in an Embedding? Analyzing Word Embeddings through Multilingual Evaluation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2067–2073. Lisbon, Portugal: Association for Computational Linguistics. https://doi.org/10.18653/v1/D15-1246.
- Lê, Minh & Antske Fokkens. 2017. Tackling Error Propagation through Reinforcement Learning: A Case of Greedy Dependency Parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, 677–687. Valencia, Spain: Association for Computational Linguistics. https://www.aclweb.org/anthology/E17-1064/.
- Lowerre, Bruce T. 1976. *The HARPY Speech Recognition System*. Pittsburgh, PA, USA: Carnegie Mellon University Ph.D. Thesis. https://apps.dtic.mil/docs/citations/ADA035146.

- Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado & Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26*, 3111–3119. Curran Associates, Inc. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.
- Pennington, Jeffrey, Richard Socher & Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. http://www.aclweb.org/anthology/D14-1162/.
- Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee & Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the*

Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2227–2237. Association for Computational Linguistics. https://doi.org/10.18653/v1/N18-1202.

- Sutton, Richard S. & Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. Second Edition (Adaptive Computation and Machine Learning series). MIT Press. https://mitpress.mit.edu/books/reinforcement-learning-second-edition.
- Tenney, Ian, Dipanjan Das & Ellie Pavlick. 2019. BERT Rediscovers the Classical NLP Pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4593–4601. Florence, Italy: Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1452.
- Tenney, Ian, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das & Ellie Pavlick. 2019. What

do you learn from context? Probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJzSgnRcKX.