Types, Networks & Stuff

Kokos

LoLa 2019

7 January 2020

Overview

The paper starts with Lambek Calculus, some how uses dependancy labels in some of its semantic types, provides a parsing algorithm for it; there are neural networks and vectors used and some accuracy results provided, but I am still unsure about the contributions of the paper and their relevance

Overview

The paper starts with Lambek Calculus, some how uses dependancy labels in some of its semantic types, provides a parsing algorithm for it; there are neural networks and vectors used and some accuracy results provided, but I am still unsure about the contributions of the paper and their relevance

- Abstract Syntax with NLP
- Extracting & Learning Type Assignments
- Navigating proofs with neural nets
- Lexicalized syntax in language modeling

This Timeline

$$\begin{array}{ccccc} \text{(N)L} & & \text{LP} & & \text{IL} \\ \text{Syntactic Derivations} & \xrightarrow{h_{der}} & \text{Derivational Semantics} & \xrightarrow{h_{lex}} & \text{Lexical Semantics} \end{array}$$

Alternative Timeline

$$\begin{tikzcd}
\text{NLP} & \\
\text{Abstract Syntax} \arrow[ddr, "h_{lex}"] & \\
 & \text{IL} \\
 & \text{Lexical Semantics}
\end{tikzcd}$$

Alternative Timeline

$$\begin{tikzcd}
 & \text{NLP} & \\
 & \text{Abstract Syntax} \arrow[ddl, "?"'] \arrow[ddr, "h_{lex}"] & \\
\text{?} & & \text{IL} \\
\text{Syntactic Derivations} & & \text{Lexical Semantics}
\end{tikzcd}$$

NLP

Grammar

IILL + Structural Control Modalities

$$\mathcal{T} := A \mid \diamond^{d}\, T_1 \rightarrow T_2$$

$A \in \mathcal{A}$	::	Atoms denoting complete phrases (NP, S, ...)
$d \in \mathcal{D}$	::	Grammatical relations (subject, object, ...)
$\diamond^{d} T$	::	Type demarkated by *dependency domain* d
$T_1 \rightarrow T_2$	::	Linear functor from T_1 to T_2

-- I did not follow this part. First of all, why dependancy labels are used? Secondly, how are they used?

Why?

- Easier to extract from corpora
- Drastic reduction in lexical ambiguity
- More informative for semantics
- Built-in Interpretability
- Diamonds can regulate parsing (?)

-- I did not follow this part. First of all, why dependancy labels are used? Secondly, how are they used?

Why?

- Easier to extract from corpora
- Drastic reduction in lexical ambiguity
- More informative for semantics
- Built-in Interpretability
- Diamonds can regulate parsing (?)

How?

$$\frac{\Gamma \vdash M : A \to B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash (M\ N) : B} \to E$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \to I$$

$$\frac{\Gamma \vdash A}{\langle \Gamma \rangle^d \vdash \diamond^d A} \diamond^d I$$

$$\frac{\Delta \vdash \diamond^d A \quad \Gamma, \langle A \rangle^d \vdash B}{\Gamma, \Delta \vdash B} \diamond^d E$$

A dataset of types & proofs

Extracting (1/3)

Algorithm

Graph flooding on syntactic parse graphs
Init with maps:

- from POS/Phrasal-tags to atoms
- dependency labels to diamond operators

Two subroutines; each *selects* untyped nodes and *types* them

Extracting (1/3)

Algorithm

Graph flooding on syntactic parse graphs
Init with maps:

- from POS/Phrasal-tags to atoms
- dependency labels to diamond operators

Two subroutines; each *selects* untyped nodes and *types* them

function $\textsc{TypeDag}$: $\mathrm{DAG}\ D \to \mathrm{DAG}$
    $D \leftarrow$ DetachNonLocal(D)
    while D is not fully typed **do**
        $D \leftarrow$ TypeStandaloneNodes(D)
        $D \leftarrow$ TypeHeadsAndMods(D)
    end while
return AttachNonLocal(D)
end function

Extracting (2/3)

λ1 Stand-Alone Nodes

select untyped nodes with

1. no incoming head/mod edge
2. no untyped daughters (except heads/mods)

type translate pos/tag to atom

λ2 Heads and Mods

select untyped nodes that

1. incoming head/mod edge
2. have a typed parent
3. have all sisters (except head/mods) typed

type endofunctor of parent if mod, else functor from vector of arguments[1] to parent type

[1] minus hypotheses

Extracting (3/3)

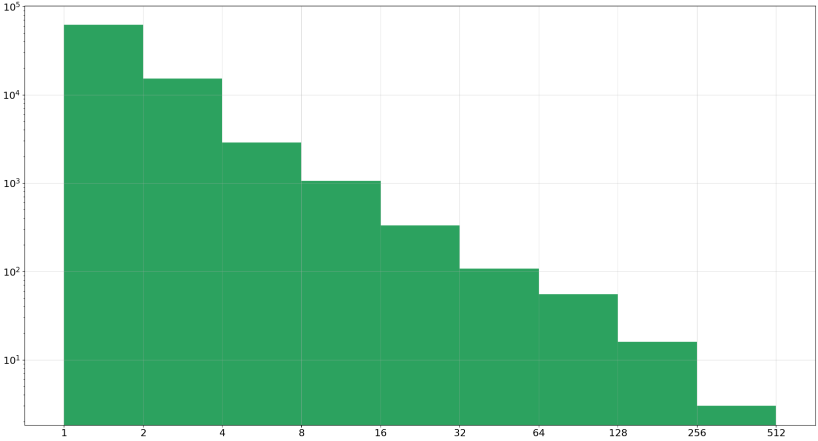
Non-Local Dependencies

function $\textsc{DetachNonLocal}$: DAG $D \to$ $\textsc{Tree}$
  for each $e \in \textsc{Filter}(\textsc{Reentrant}, D.edges)$ **do**
    $c \leftarrow \text{copy}(e.target)$
    $D.nodes \leftarrow D.nodes \cup \{c\}$
    $D.edges \leftarrow D.edges - \{e\}$
    $D.edges \leftarrow D.edges \cup \{(e.source, e.label, c)\}$
  end for
return D
end function

Mommy, where do types come from?

Lexical Type Ambiguity

Word/Type histogram



Supertagging (1/3)

General idea

$$p(t_1, t_2, \ldots t_n | w_1, w_2, \ldots w_n, \theta)$$

Markov Assumption

$$\approx \prod_{i=1}^{n} p(t_i | w_1, w_2, \ldots w_n, \theta)$$

- ☺ Seq2Seq Classification
- ☹ Single Answer
- ☹ Sample Sparsity
- ☹ Closed Domain Assumption

Supertagging (1/3)

General idea

$$p(t_1, t_2, \ldots t_n | w_1, w_2, \ldots w_n, \theta)$$

Markov Assumption

$$\approx \prod_{i=1}^{n} p(t_i | w_1, w_2, \ldots w_n, \theta)$$

- ☺ Seq2Seq Classification
- ☹ Single Answer
- ☹ Sample Sparsity
- ☹ Closed Domain Assumption

Supertagging (2/3)

General idea

$$p(t_1, t_2, \ldots t_n | w_1, w_2, \ldots w_n, \theta)$$

~~Markov Assumption~~

$$\approx \prod_{i=1}^{n} p(t_i | t_1, \ldots t_{i-1}, w1, \ldots, w_n, \theta)$$

- ☺ Seq2Seq Transduction
- ☺ Many Answers
- ☹ Sample Sparsity
- ☹ Closed Domain Assumption

Supertagging (3/3)

A step further

The syntax of the type system forms a simple CFG

$$
\begin{aligned}
S &\implies A & \forall\, A &\in \mathcal{A} \\
S &\implies d\; S\; S & \forall\, d &\in \mathcal{D}
\end{aligned}
$$

Supertagging (3/3)

A step further

The syntax of the type system forms a simple CFG

$$\begin{aligned} S &\implies A & \forall\, A \in \mathcal{A} \\ S &\implies d\; S\; S & \forall\, d \in \mathcal{D} \end{aligned}$$

- Supertagging as conditional CFG generation
- CFG terminals as decoding targets

$$\approx \prod_{i=1}^{m} p(\sigma_i | \sigma_1, \ldots \sigma_{i-1}, w1, \ldots, w_n, \theta)$$

$$\sigma \in \mathcal{A} \cup \mathcal{D} \cup \{\text{<SEP>}\}$$

Supertagging (3/3)

A step further

The syntax of the type system forms a simple CFG

$$
\begin{array}{lllcccc}
S & \implies & A & \quad & \forall\, A & \in & \mathcal{A} \\
S & \implies & d\ S\ S & \quad & \forall\, d & \in & \mathcal{D}
\end{array}
$$

- Supertagging as conditional CFG generation
- CFG terminals as decoding targets

$$\approx \prod_{i=1}^{m} p(\sigma_i | \sigma_1, \ldots \sigma_{i-1}, w1, \ldots, w_n, \theta)$$

$$\sigma \in \mathcal{A} \cup \mathcal{D} \cup \{\text{<SEP>}\}$$

☺ ~~Sample Sparsity~~ Many (sub)type examples

☺ ~~Closed Domain Assumption~~ Inductive construction of any type

Structural Ambiguity

Navigating Proofs

Parse State

- A logical judgement (premises & conclusion)
- Word associations for (some) premise formulas
- A single element stack

Navigating Proofs

Parse State

- A logical judgement (premises & conclusion)
- Word associations for (some) premise formulas
- A single element stack

Framework

Given a parse state:

1. Decide between introduction $\oplus$ elimination
2. Perform either
3. Update state(s)
4. Repeat

Proof Ambiguity

The Problem

- Introduction steps are deterministic
- Eliminations are not

Proof Ambiguity

The Problem

- Introduction steps are deterministic
- Eliminations are not

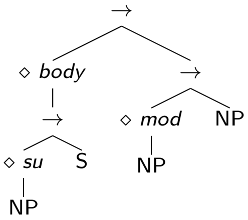
Key Insight

Elimination branching $\sim$ binary sequence chunking
Semantic content can help disambiguate structure

Vectorizing Types

Types are trees

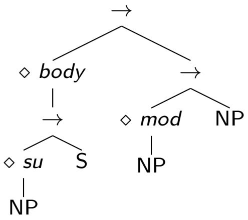
$$\diamond^{body}(\diamond^{su}NP \rightarrow S) \rightarrow \diamond^{mod}NP \rightarrow NP$$



Vectorizing Types

Types are trees

$$\diamond^{body}(\diamond^{su}NP \to S) \to \diamond^{mod}NP \to NP$$



- Atoms as vectors in $\mathbb{R}^n$
- Dependencies as functions $\mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}^n$

Proof Traversal

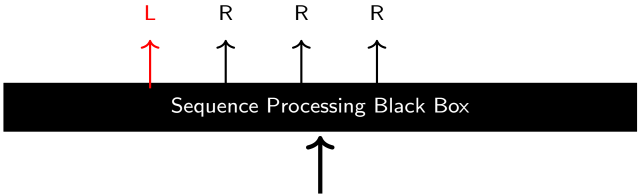
$$
\dfrac{\dfrac{}{\text{ducks} \vdash NP}\,Ax. \qquad \dfrac{\dfrac{\dfrac{}{\text{predators} \vdash NP}\,Ax. \quad \dfrac{}{\text{many} \vdash NP \to NP}\,Ax.}{\text{many, predators} \vdash NP}\,\to E \qquad \dfrac{}{\text{have} \vdash NP \to NP \to S}\,Ax.}{\text{have, many, predators} \vdash NP \to S}\,\to E}{\text{ducks, have, many, predators} \vdash S}
$$

Sequence Processing Black Box

ducks, have, many, predators $\vdash$ S

Proof Traversal

$$\dfrac{\dfrac{}{\text{ducks} \vdash \text{NP}}\,Ax. \qquad \dfrac{\dfrac{\dfrac{}{\text{predators} \vdash \text{NP}}\,Ax. \quad \dfrac{}{\text{many} \vdash \text{NP} \to \text{NP}}\,Ax.}{\text{many, predators} \vdash \text{NP}}\,\to E \qquad \dfrac{}{\text{have} \vdash \text{NP} \to \text{NP} \to \text{S}}\,Ax.}{\text{have, many, predators} \vdash \text{NP} \to \text{S}}\,\to E}{\text{ducks, have, many, predators} \vdash \text{S}}$$



ducks, have, many, predators $\vdash$ S

Proof Traversal

$$\dfrac{\dfrac{}{\text{ducks} \vdash \mathsf{NP}}\,Ax. \qquad \dfrac{\dfrac{\dfrac{}{\text{predators} \vdash \mathsf{NP}}\,Ax. \quad \dfrac{}{\text{many} \vdash \mathsf{NP} \to \mathsf{NP}}\,Ax.}{\text{many, predators} \vdash \mathsf{NP}}\,\to E \quad \dfrac{}{\text{have} \vdash \mathsf{NP} \to \mathsf{NP} \to \mathsf{S}}\,Ax.}{\text{have, many, predators} \vdash \mathsf{NP} \to \mathsf{S}}\,\to E}{\text{ducks, have, many, predators} \vdash \mathsf{S}}$$

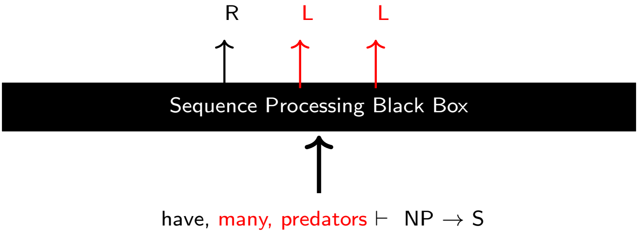
Sequence Processing Black Box

$\uparrow$

have, many, predators $\vdash$ NP $\to$ S

Proof Traversal

$$\dfrac{\dfrac{}{\text{ducks} \vdash \mathsf{NP}}\,Ax. \qquad \dfrac{\dfrac{\dfrac{}{\text{predators} \vdash \mathsf{NP}}\,Ax. \quad \dfrac{}{\text{many} \vdash \mathsf{NP} \to \mathsf{NP}}\,Ax.}{\color{red}{\text{many}}, \color{red}{\text{predators}} \vdash \mathsf{NP}}\,\to E \qquad \dfrac{}{\text{have} \vdash \mathsf{NP} \to \mathsf{NP} \to \mathsf{S}}\,Ax.}{\text{have, many, predators} \vdash \mathsf{NP} \to \mathsf{S}}\,\to E}{\text{ducks, have, many, predators} \vdash \mathsf{S}}$$



Proof Traversal

$$\dfrac{\dfrac{}{\text{ducks} \vdash \text{NP}}\,Ax. \qquad \dfrac{\dfrac{\dfrac{}{\text{predators} \vdash \text{NP}}\,Ax. \quad \dfrac{}{\text{many} \vdash \text{NP} \to \text{NP}}\,Ax.}{\text{many, predators} \vdash \text{NP}}\,\to E \qquad \dfrac{}{\text{have} \vdash \text{NP} \to \text{NP} \to \text{S}}\,Ax.}{\text{have, many, predators} \vdash \text{NP} \to \text{S}}\,\to E}{\text{ducks, have, many, predators} \vdash \text{S}}$$

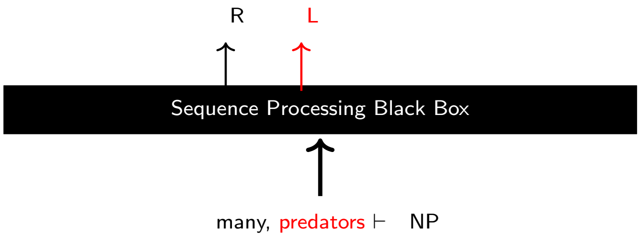
Sequence Processing Black Box

$\uparrow$

many, predators $\vdash$ NP

Proof Traversal

$$\dfrac{\dfrac{}{\text{ducks} \vdash \text{NP}}\,Ax. \qquad \dfrac{\dfrac{\dfrac{}{\color{red}{\text{predators}} \vdash \text{NP}}\,Ax. \quad \dfrac{}{\text{many} \vdash \text{NP} \to \text{NP}}\,Ax.}{\text{many, predators} \vdash \text{NP}}\,\to E \qquad \dfrac{}{\text{have} \vdash \text{NP} \to \text{NP} \to \text{S}}\,Ax.}{\text{have, many, predators} \vdash \text{NP} \to \text{S}}\,\to E}{\text{ducks, have, many, predators} \vdash \text{S}}$$



Proof Traversal

$$
\dfrac{
  \dfrac{}{\text{ducks} \vdash \text{NP}}\,Ax.
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{}{\text{predators} \vdash \text{NP}}\,Ax.
      \quad
      \dfrac{}{\text{many} \vdash \text{NP} \to \text{NP}}\,Ax.
    }{\text{many}, \text{predators} \vdash \text{NP}}\,\to E
    \quad
    \dfrac{}{\text{have} \vdash \text{NP} \to \text{NP} \to \text{S}}\,Ax.
  }{\text{have}, \text{many}, \text{predators} \vdash \text{NP} \to \text{S}}\,\to E
}{\text{ducks}, \text{have}, \text{many}, \text{predators} \vdash \text{S}}
$$

Sequence Processing Black Box

Proof Traversal

$$
\dfrac{\dfrac{}{\mathsf{ducks} \vdash \mathsf{NP}}\,Ax. \qquad \dfrac{\dfrac{\dfrac{}{\mathsf{predators} \vdash \mathsf{NP}}\,Ax. \quad \dfrac{}{\mathsf{many} \vdash \mathsf{NP} \to \mathsf{NP}}\,Ax.}{\mathsf{many}, \mathsf{predators} \vdash \mathsf{NP}}\,\to E \quad \dfrac{}{\mathsf{have} \vdash \mathsf{NP} \to \mathsf{NP} \to \mathsf{S}}\,Ax.}{\mathsf{have}, \mathsf{many}, \mathsf{predators} \vdash \mathsf{NP} \to \mathsf{S}}\,\to E}{\mathsf{ducks}, \mathsf{have}, \mathsf{many}, \mathsf{predators} \vdash \mathsf{S}}
$$

Sequence Processing Black Box

$$
\left.\begin{array}{lcl}
\text{Training sample} & : & \text{Elimination branching} \\
\text{Sentence} & : & \text{N } \textit{independent} \text{ samples}
\end{array}\right\} \text{Training Parallelism}
$$

Proof Traversal

$$\dfrac{\dfrac{}{\text{ducks} \vdash \text{NP}}\,Ax. \qquad \dfrac{\dfrac{\dfrac{}{\text{predators} \vdash \text{NP}}\,Ax. \quad \dfrac{}{\text{many} \vdash \text{NP} \to \text{NP}}\,Ax.}{\text{many, predators} \vdash \text{NP}}\,\to E \quad \dfrac{}{\text{have} \vdash \text{NP} \to \text{NP} \to \text{S}}\,Ax.}{\text{have, many, predators} \vdash \text{NP} \to \text{S}}\,\to E}{\text{ducks, have, many, predators} \vdash \text{S}}$$

Sequence Processing Black Box

Parsing in $\mathcal{O}(1)$[2]

[2] Sensationalist

Parsing as Latent Permutation

$$np, np \to np \to s, np \to np, np \vdash s$$

Parsing as Latent Permutation

$$np_1, np_2 \to np_3 \to s_1, np_4 \to np_5, np_6 \vdash s_2$$

Parsing as Latent Permutation

$$np_1^+, np_2^- \to np_3^- \to s_1^+, np_4^- \to np_5^+, np_6^+ \vdash s_2^-$$

Parsing as Latent Permutation

$$np_1^+, np_2^- \rightarrow np_3^- \rightarrow s_1^+, np_4^- \rightarrow np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

Parsing as Latent Permutation

$$np_1^+, np_2^- \to np_3^- \to s_1^+, np_4^- \to np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

	np_2	np_3	np_4	s_2
np_1		X		
s_1				X
np_5	X			
np_6			X	

Parsing as Latent Permutation

$$np_1^+, np_2^- \to np_3^- \to s_1^+, np_4^- \to np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

$$\mathcal{R} \in \mathbb{B}^{4\times 4} = \begin{array}{c|cccc} & np_2 & np_3 & np_4 & s_2 \\ \hline np_1 & 0 & 1 & 0 & 0 \\ s_1 & 0 & 0 & 0 & 1 \\ np_5 & 1 & 0 & 0 & 0 \\ np_6 & 0 & 0 & 1 & 0 \end{array}$$

$$\text{match}(N) = P\mathcal{R}^T$$

Parsing as Latent Permutation

$$np_1^+, np_2^- \to np_3^- \to s_1^+, np_4^- \to np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

$$\mathcal{R} \in \mathbb{B}^{4\times 4} = \begin{array}{c|cccc} & np_2 & np_3 & np_4 & s_2 \\ \hline np_1 & 0 & 1 & 0 & 0 \\ s_1 & 0 & 0 & 0 & 1 \\ np_5 & 1 & 0 & 0 & 0 \\ np_6 & 0 & 0 & 1 & 0 \end{array}$$

$$\text{match}(N) = P\mathcal{R}^T$$

- $\mathcal{R}$ is *discrete*

Parsing as Latent Permutation

$$np_1^+, np_2^- \to np_3^- \to s_1^+, np_4^- \to np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

$$\mathcal{R} \in \mathbb{B}^{4\times 4} = \begin{array}{c|cccc} & np_2 & np_3 & np_4 & s_2 \\ \hline np_1 & 0 & 1 & 0 & 0 \\ s_1 & 0 & 0 & 0 & 1 \\ np_5 & 1 & 0 & 0 & 0 \\ np_6 & 0 & 0 & 1 & 0 \end{array}$$

$$\text{match}(N) = P\mathcal{R}^T$$

- $\mathcal{R}$ is *discrete*
- .. but its continuous relaxations can be approximated (Sinkhorn-Knopp)

References

Grammar & Extraction

- ÆTHEL: Automatically Extracted Type-Logical Derivations for Dutch link

Supertagging & Parsing

- Constructive Type-Logical Supertagging with Self-Attention Networks link
- Deductive Parsing with an Unbounded Type Lexicon link

Permutations

- Sinkhorn-Knopp Algorithm link
- Learning Latent Permutations with Gumbel-Sinkhorn Networks link