..continuing

The agenda:

- λ Choosing the logic
- λ Making a dataset: proofs and lexical type assignments
- λ Learning the type assignment process
- λ Navigating the proof space
- λ Syntax-aware & type-correct text representations

Lexical type ambiguity

Type assignments are often *ambiguous* and context-dependent:

	very realistic lexicon	
running ::	$\overset{mod}{\Box}(np \multimap np)$	*running dog*
	inf	*I like running*
like ::	$\overset{obj}{\Diamond} np \multimap \overset{su}{\Diamond} np \multimap s_m$	*ducks like seeds*
	$\overset{obj}{\Diamond} np \multimap \overset{su}{\Diamond} pron \multimap s_m$	*I like ducks*
	$\overset{obj}{\Diamond} inf \multimap \overset{su}{\Diamond} np \multimap s_m$	*ducks like swimming*
	$\overset{obj}{\Diamond} inf \multimap \overset{su}{\Diamond} pron \multimap s_m$	*I like swimming*
	$\overset{obj}{\Diamond} np \multimap \overset{mod}{\Box}(s_m \multimap s_m)$	*I swim like a duck*
	$\ldots$	

Lexical type ambiguity

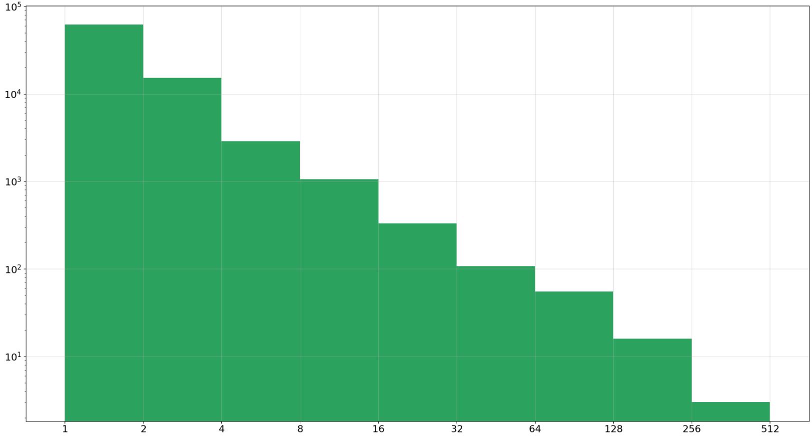
generally:

$$w :: t_1 \& t_2 \& t_3 \& \ldots \& t_n$$

more refined grammar $\Longrightarrow$

- ☹ harder lexical disambiguation (more n per word)
- ☺ easier parsing (*if* one starts from correct type)

Lexical type ambiguity



types (*log2-transformed*) per word

Supertagging

General idea

Given a sentence $w_1, w_2, \ldots, w_n$
find the type sequence $t_1, t_2, \ldots t_n$ that maximimizes

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

where θ the trainable parameter space

Discriminative approach

Markov assumption

$$p(t_1, t_2, \ldots t_n | w_1, w_2, \ldots w_n, \theta)$$
$$\approx \prod_{i=1}^{n} p(t_i | w_1, w_2, \ldots w_n, \theta)$$

- ☺ contextualized token classification
- ☹ single answer
- ☹ sample sparsity
- ☹ closed codomain assumption

Generative approach

~~Markov assumption~~

$$
\begin{aligned}
& p(t_1, t_2, \ldots t_n | w_1, w_2, \ldots w_n, \theta) \\
\approx & \prod_{i=1}^{n} p(t_i | t_1, t_2, \ldots, t_{i-1}, w_1, w_2, \ldots w_n, \theta)
\end{aligned}
$$

- ☺ seq2seq translation
- ☺ many answers
- ☹ sample sparsity
- ☹ closed codomain assumption

Generative approach: one more step

The type syntax:

$$cod(\mathcal{L}) := A \mid \overset{d}{\Diamond} T \multimap T' \mid \overset{d}{\Box}(T \multimap T')$$

corresponds to a simple cfg (in prefix notation) with meta-rules:

$$\begin{array}{ll} S \to A & \forall A \in \mathcal{A} \\ S \to \overset{d}{\Diamond} S\ S & \forall d \in \text{comps} \\ S \to \overset{d}{\Box} S\ S & \forall d \in \text{adjns} \end{array}$$

i.e. each type t_i can be written as a sequence of primitive symbols $\vec{\sigma}$

Generative approach: one more step

The type syntax:

$$cod(\mathcal{L}) := A \mid \overset{d}{\Diamond} T \multimap T' \mid \overset{d}{\Box}(T \multimap T')$$

corresponds to a simple cfg (in prefix notation) with meta-rules:

$$\begin{aligned} S &\to A && \forall A \in \mathcal{A} \\ S &\to \overset{d}{\Diamond} S\ S && \forall d \in \mathrm{comps} \\ S &\to \overset{d}{\Box} S\ S && \forall d \in \mathrm{adjns} \end{aligned}$$

i.e. each type t_i can be written as a sequence of primitive symbols $\vec{\sigma}$

$$\begin{aligned} & p(t_1, t_2, \ldots t_n | w_1, w_2, \ldots w_n, \theta) \\ \approx & \prod_{i=1}^{m} p(\sigma_i | \sigma_1, \sigma_2, \ldots, \sigma_{i-1}, w_1, w_2, \ldots w_n, \theta) \end{aligned}$$

- ☺ each type contains many subtypes (less sparsity)
- ☺ any valid types can be inductively constructed (open codomain)

Some implementation notes

RNNs

- single state compresses entire sentence (lossy)
- temporal dependence during training (slow)

Some implementation notes

RNNs /w attention

- one vector per sequence element (lossless)
- temporal dependence during training (slow)

Some implementation notes

just attention

- one vector per sequence element (lossless)
- fully parallel training (fast)

Some implementation notes

just attention

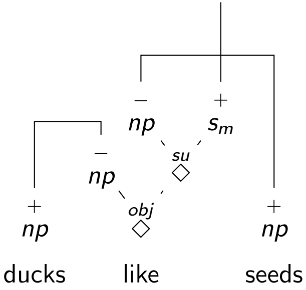
- one vector per sequence element (lossless)
- fully parallel training (fast)

open questions

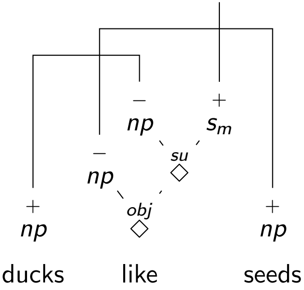
?	explicit tree structure	(tree-shaped decoders)
?	decoding order	(linear order vs. easy first)
?	ultra-long types	(two-step decoding, generic vs. concrete instances)
?	inference time delay	(linearization via kernel methods)
?	logical constraints	(reinforcement learning)

Proof Ambiguity

The type system (being non-directional) permits too many parses:



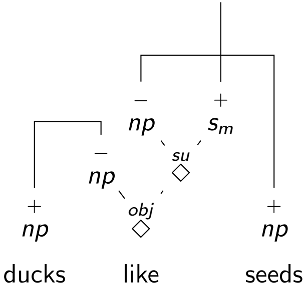
`like (ducks)`$^{\texttt{obj}}$ `(seeds)`$^{\texttt{su}}$

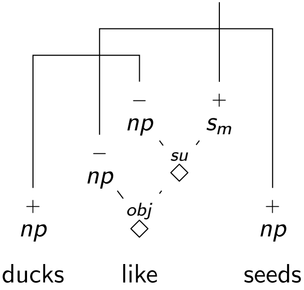


`like (seeds)`$^{\texttt{obj}}$ `(ducks)`$^{\texttt{su}}$

Proof Ambiguity

The type system (being non-directional) permits too many parses:





$\texttt{like}\ (\texttt{ducks})^{\texttt{obj}}\ (\texttt{seeds})^{\texttt{su}}$ $\qquad$ $\texttt{like}\ (\texttt{seeds})^{\texttt{obj}}\ (\texttt{ducks})^{\texttt{su}}$

How can we select the correct one?

Neural Proof Nets

the berries that blackbirds steal

1. pass the sentence through the supertagger

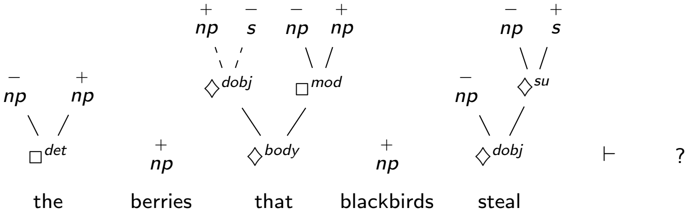
Neural Proof Nets

$$\Box^{det}, np, np, \#, np, \#, \Diamond^{body}, \Diamond^{dobj}, np, s, \Box^{mod}, np, np, \#, \Diamond^{dobj}, np, \Diamond^{su}, np, s$$

the berries that blackbirds steal

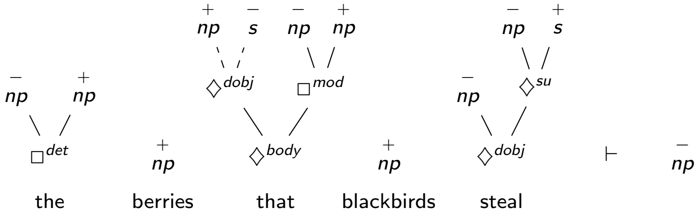
2. parse types to trees and assign polarity information

Neural Proof Nets



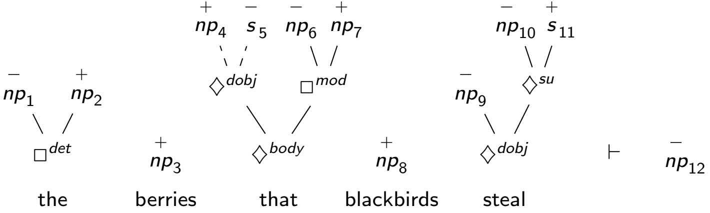
3. find conclusion as the singleton $\mathcal{A}^{+} - \mathcal{A}^{-}$

Neural Proof Nets



4. index pos/neg occurrences and arrange in a table

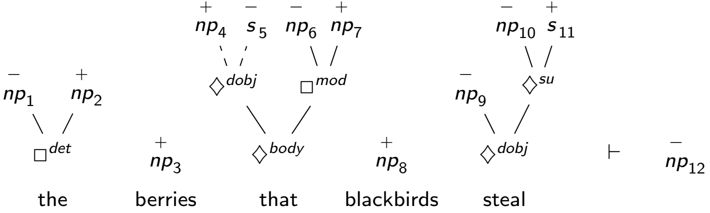
Neural Proof Nets



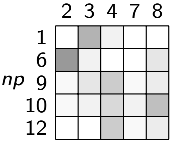
5. fill table with pair-wise agreement scores

		2	3	4	7	8
	1					
	6					
np	9					
	10					
	12					

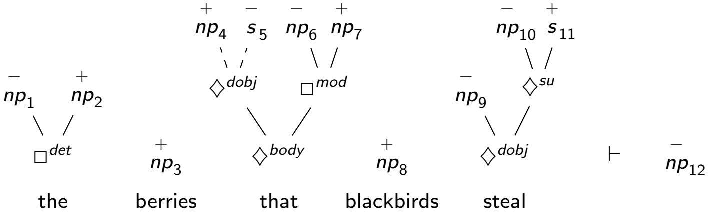
Neural Proof Nets



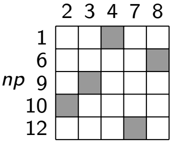
6. discretize result with Sinkhorn



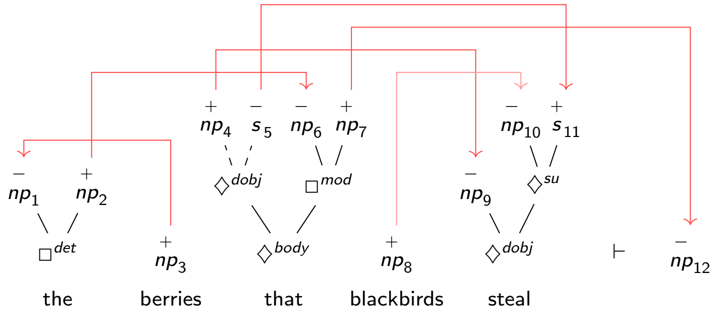
Neural Proof Nets



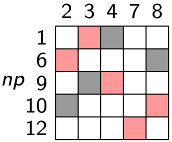
7. train against ground-truth axiom links



Neural Proof Nets



7. train against ground-truth axiom links



Some implementation notes #2

open questions

?	structural ambiguity	(sampling with noise)
?	polymorphic linking	(coherent chunks as single atom)