Neural ENIGMA

Karel Chvalovský Jan Jakubův
Martin Suda Josef Urban

Czech Technical University in Prague, Czech Republic

AITP'19, Obergurgl, April 2019

Motivation

ENIGMA:

- guiding clause selection in a first-order saturation-based ATP (E-prover)

Why to use neural networks?

Motivation

ENIGMA:

- guiding clause selection in a first-order saturation-based ATP (E-prover)

Why to use neural networks?

- *It's cool and we don't want to be left behind!*

Motivation

ENIGMA:

- guiding clause selection in a first-order saturation-based ATP (E-prover)

Why to use neural networks?

- *It's cool and we don't want to be left behind!*
- implicit automatic feature extraction

Motivation

ENIGMA:

- guiding clause selection in a first-order saturation-based ATP (E-prover)

Why to use neural networks?

- *It's cool and we don't want to be left behind!*
- implicit automatic feature extraction

Why maybe not to use them?

- Training tends to be more expensive
- Evaluation is slow-ish for the task [Loos et al., 2017]

Outline

Outline

Idea of embeddings:

- map logical objects (terms, literals, clauses) into $\mathbf{R}^n$
- hope they capture semantics rather than just syntax!

Recursive Neural Networks and Embeddings

Idea of embeddings:

- map logical objects (terms, literals, clauses) into $\mathbf{R}^n$
- hope they capture semantics rather than just syntax!

Recursive Neural Networks [Goller and Kuchler, 1996]

- recursively follow the inductive definition of logical objects
- share sub-network blocks among occurrences of the same entity

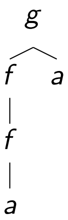
Idea of embeddings:

- map logical objects (terms, literals, clauses) into $\mathbf{R}^n$
- hope they capture semantics rather than just syntax!

Recursive Neural Networks [Goller and Kuchler, 1996]

- recursively follow the inductive definition of logical objects
- share sub-network blocks among occurrences of the same entity

$$
\begin{aligned}
a &: \mathbf{R}^n \\
f &: \mathbf{R}^n \to \mathbf{R}^n \\
g &: \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}^n
\end{aligned}
$$



Building Blocks of our Network

All under the aligned-signature assumption!

Building Blocks of our Network

All under the aligned-signature assumption!

- abstracting all first-order variables by a single embedding
- single block for every skolem symbol of a specific arity

Building Blocks of our Network

All under the aligned-signature assumption!

- abstracting all first-order variables by a single embedding
- single block for every skolem symbol of a specific arity
- separate block for every function and predicate
- block for negation and equality

Building Blocks of our Network

All under the aligned-signature assumption!

- abstracting all first-order variables by a single embedding
- single block for every skolem symbol of a specific arity
- separate block for every function and predicate
- block for negation and equality
- "or"-ing LSTM to embed a clause
- "and"-ing LSTM to embed the negated conjecture

Building Blocks of our Network

All under the aligned-signature assumption!

- abstracting all first-order variables by a single embedding
- single block for every skolem symbol of a specific arity
- separate block for every function and predicate
- block for negation and equality
- "or"-ing LSTM to embed a clause
- "and"-ing LSTM to embed the negated conjecture
- final FF block taking the clause embedding $v_C \in \mathbf{R}^n$ and the negated conjecture embedding $v_{Thm} \in \mathbf{R}^m$ and producing a probability estimate of usefulness:

$$p(C \text{ useful for proving } \mathrm{Thm}) = \sigma(\mathit{final}(v_C, v_{Thm}))$$

where σ is the sigmoid function, "squashing" $\mathbf{R}$ nicely into $[0, 1]$

Architecture Parameters and Training

Current neural model parameters:

- $n = 64$
- function and predicate symbols are represented by a linear layer and ReLU6: $(\min(\max(0, x), 6))$
- conjecture embedding has size $m = 16$
- the final layer is a sequence of linear, ReLU, linear, ReLU, and linear layers $(\mathbf{R}^{n+m} \to \mathbf{R}^{\frac{n}{2}} \to \mathbf{R}^2)$
- rare symbols are grouped together — we can loosely speaking obtain a general constant, binary function, . . .

Architecture Parameters and Training

Current neural model parameters:

- $n = 64$
- function and predicate symbols are represented by a linear layer and ReLU6: $(\min(\max(0, x), 6))$
- conjecture embedding has size $m = 16$
- the final layer is a sequence of linear, ReLU, linear, ReLU, and linear layers ($\mathbf{R}^{n+m} \to \mathbf{R}^{\frac{n}{2}} \to \mathbf{R}^2$)
- rare symbols are grouped together — we can loosely speaking obtain a general constant, binary function, ...

Training:

- we use minibatches, where we group together examples that share the same conjecture and we cache all the representations obtained in one batch

Outline

Perfect Term Sharing and Caching

Terms in E are perfectly shared:

- at most one instance of every possible term in memory
- equality test in constant time

Caching of embeddings:

- thanks to the chosen architecture (i.e. the recursive nets), each logical term has a unique embedding
- hash table using term pointer as key gives us an efficient cache

➥ Each term embedded only once!

Outline

Connecting the network with E

Clause selection in E – a recap:

- a variety of heuristics for ordering clauses called *clause weight functions*
- each to govern its own queue
- multiple queues combined in a round-robin fashion under some frequencies: e.g. $3 * \mathit{fifo} + 4 * \mathit{symbols}$

Clause selection in E – a recap:

- a variety of heuristics for ordering clauses called *clause weight functions*
- each to govern its own queue
- multiple queues combined in a round-robin fashion under some frequencies: e.g. $3 * \mathit{fifo} + 4 * \mathit{symbols}$

New clause weight function based on the NN:

- could use the predicted probability values (`order by, desc`)
- however, just yes / no works better!

  ➥ Insider knowledge: *fifo* then breaks the ties!

Connecting the network with E

Clause selection in E – a recap:

- a variety of heuristics for ordering clauses called *clause weight functions*
- each to govern its own queue
- multiple queues combined in a round-robin fashion under some frequencies: e.g. $3 * \mathit{fifo} + 4 * \mathit{symbols}$

New clause weight function based on the NN:

- could use the predicted probability values (`order by, desc`)
- however, just yes / no works better!
  ➥ Insider knowledge: *fifo* then breaks the ties!
- also, mix NN with the original heuristic for the best results (we mixed 50-50 in experiments)

Outline

Experimental Setup

Selected benchmark:

- MPTP 2078: FOL translation of selected articles from Mizar Mathematical Library (MML)

Furthermore:

- Fix a good E strategy $\mathcal{S}$ from the past
- 10 second time limit
- first run $\mathcal{S}$ to collect training data from found proofs
  - solved 1086 out of 2078
  - which yielded approx 21000 positives and 201000 negatives

Experimental Setup

Selected benchmark:

- MPTP 2078: FOL translation of selected articles from Mizar Mathematical Library (MML)

Furthermore:

- Fix a good E strategy $\mathcal{S}$ from the past
- 10 second time limit
- first run $\mathcal{S}$ to collect training data from found proofs
  - solved 1086 out of 2078
  - which yielded approx 21000 positives and 201000 negatives

- force Pytorch to use just single core!

TPR/TNR: True Positive/Negative Rates

- Training Accuracy:

	$\mathcal{M}_{\mathrm{lin}}$	$\mathcal{M}_{\mathrm{tree}}$	$\mathcal{M}_{\mathrm{nn}}$
TPR	90.54 %	99.36 %	97.82 %
TNR	83.52 %	93.32 %	94.69 %

- Testing Accuracy:

	$\mathcal{M}_{\mathrm{lin}}$	$\mathcal{M}_{\mathrm{tree}}$	$\mathcal{M}_{\mathrm{nn}}$
TPR	80.54 %	83.35 %	82.00 %
TNR	62.28 %	72.60 %	76.88 %

Models ATP Performance

- $\mathcal{S}$ with model $\mathcal{M}$ alone ($\odot$) or combined 50-50 ($\oplus$) in 10s

	$\mathcal{S}$	$\mathcal{S} \odot \mathcal{M}_{\mathrm{lin}}$	$\mathcal{S} \odot \mathcal{M}_{\mathrm{tree}}$	$\mathcal{S} \odot \mathcal{M}_{\mathrm{nn}}$
solved	1086	1115	1231	1167
unique	0	3	10	3
$\mathcal{S}+$	0	+119	+155	+114
$\mathcal{S}-$	0	-90	-10	-33
	$\mathcal{S}$	$\mathcal{S} \oplus \mathcal{M}_{\mathrm{lin}}$	$\mathcal{S} \oplus \mathcal{M}_{\mathrm{tree}}$	$\mathcal{S} \oplus \mathcal{M}_{\mathrm{nn}}$
solved	1086	1210	**1256**	1197
unique	0	7	**15**	2
$\mathcal{S}+$	0	+138	**+173**	+119
$\mathcal{S}-$	0	-14	**-3**	-8

Smartness and Speed

All Solved Relative Processed Average:

	$\mathcal{M}_{\text{lin}}$	$\mathcal{M}_{\text{tree}}$	$\mathcal{M}_{\text{nn}}$
$\mathcal{S}\odot$	2.18 ± 20.35	0.60 ± 0.98	0.59 ± 0.75
$\mathcal{S}\oplus$	0.91 ± 0.58	0.59 ± 0.36	0.69 ± 0.94

Smartness and Speed

All Solved Relative Processed Average:

	$\mathcal{M}_{\text{lin}}$	$\mathcal{M}_{\text{tree}}$	$\mathcal{M}_{\text{nn}}$
$\mathcal{S}\odot$	2.18 ± 20.35	0.60 ± 0.98	0.59 ± 0.75
$\mathcal{S}\oplus$	0.91 ± 0.58	0.59 ± 0.36	0.69 ± 0.94

None Solved Relative Generated Average:

	$\mathcal{M}_{\text{lin}}$	$\mathcal{M}_{\text{tree}}$	$\mathcal{M}_{\text{nn}}$
$\mathcal{S}\odot$	0.61 ± 0.52	0.42 ± 0.38	0.06 ± 0.08
$\mathcal{S}\oplus$	0.56 ± 0.35	0.43 ± 0.35	0.07 ± 0.09

Smartness and Speed

All Solved Relative Processed Average:

	$\mathcal{M}_{\text{lin}}$	$\mathcal{M}_{\text{tree}}$	$\mathcal{M}_{\text{nn}}$
$\mathcal{S}\odot$	2.18 ± 20.35	0.60 ± 0.98	0.59 ± 0.75
$\mathcal{S}\oplus$	0.91 ± 0.58	0.59 ± 0.36	0.69 ± 0.94

None Solved Relative Generated Average:

	$\mathcal{M}_{\text{lin}}$	$\mathcal{M}_{\text{tree}}$	$\mathcal{M}_{\text{nn}}$
$\mathcal{S}\odot$	0.61 ± 0.52	0.42 ± 0.38	0.06 ± 0.08
$\mathcal{S}\oplus$	0.56 ± 0.35	0.43 ± 0.35	0.07 ± 0.09

➥ without caching, NSRGA of $\mathcal{S} \oplus \mathcal{M}_{\text{nn}}$ drops from 7.1 to 3.6 percent of the speed of $\mathcal{S}$

Conclusion

Summary:

- recursive neural networks catching up on gradient boosted trees for clause selection in E
- evaluation speed improved via caching

Still open:

- What when symbols are not aligned?
- What is the best way of integrating the guidance and why?
- Proof state charaterizations for better context.

Conclusion

Summary:

- recursive neural networks catching up on gradient boosted trees for clause selection in E
- evaluation speed improved via caching

Still open:

- What when symbols are not aligned?
- What is the best way of integrating the guidance and why?
- Proof state charaterizations for better context.

Thank you for attention!