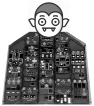
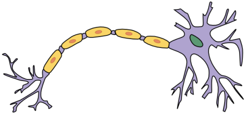
One Year With Deepire: Lessons Learned and Where to Go Next?

Martin Suda[1]

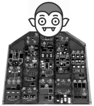
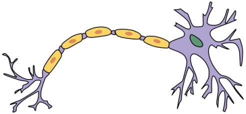
Czech Technical University in Prague, Czech Republic

AITP, September 2021

[1]Supported by Czech Science Foundation project 20-06390Y.

Deepire =  + 

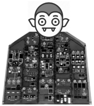
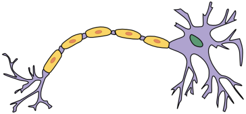
Deepire: Powering an ATP using Neural Networks

Deepire =  + 

Vampire

- Automatic Theorem Prover (ATP) for First-order Logic (FOL)
- state-of-the-art saturation-based prover

Deepire: Powering an ATP using Neural Networks

Deepire =  + 

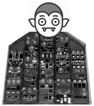
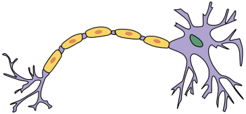
Vampire

- Automatic Theorem Prover (ATP) for First-order Logic (FOL)
- state-of-the-art <u>saturation-based</u> prover

Neural (ENIGMA-style) guidance

- targeting the <u>clause selection</u> decision point
- supervised learning from successful runs

Deepire: Powering an ATP using Neural Networks

Deepire =  + 

Vampire

- Automatic Theorem Prover (ATP) for First-order Logic (FOL)
- state-of-the-art saturation-based prover

Neural (ENIGMA-style) guidance

- targeting the clause selection decision point
- supervised learning from successful runs

➥ The special bit: uses a recursive neural network (RvNN) based solely on clause derivation history

Deepire Is One Year Old!

Deepire Is One Year Old!

The story so far:

- [AITP20] - introduced the first prototype
- [CADE21] - improved Vampire's theory reasoning on SMTLIB
- [FroCoS21] - surprisingly surpased ENIGMA on MIZAR40

The story so far:

- [AITP20] - introduced the first prototype
- [CADE21] - improved Vampire's theory reasoning on SMTLIB
- [FroCoS21] - surprisingly surpased ENIGMA on MIZAR40

In this talk:

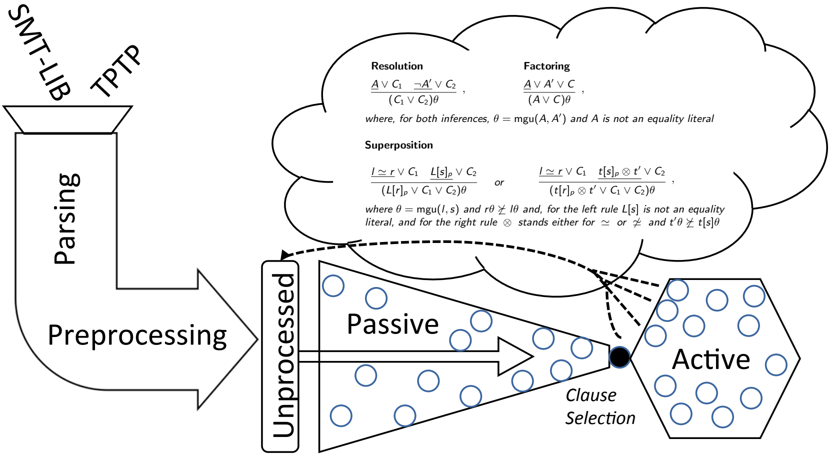
- overview of the main results, insights, and future outlooks

Outline

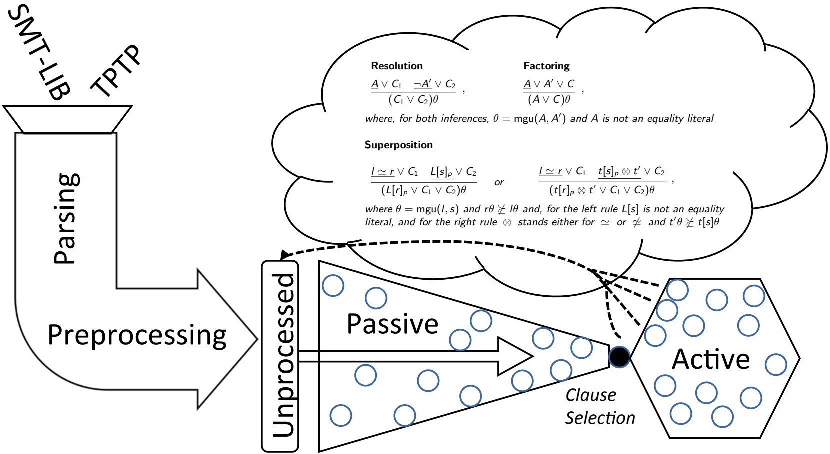
Outline

1. Saturation, Clause Selection, and Machine Learning
2. Recursive Neural Networks over Clause Derivations
3. Notes on Implementation and Training
4. Experiments on Mizar
5. Conclusion

Saturation-based Theorem Proving



Saturation-based Theorem Proving



At a typical successful end: $|Passive| \gg |Active| \gg |Proof|$

How is clause selection traditionally done?

Take simple clause evaluation criteria:

- <u>age</u>: prefer clauses that were generated long time ago
- <u>weight</u>: prefer clauses with fewer symbols

How is clause selection traditionally done?

Take simple clause evaluation criteria:

- age: prefer clauses that were generated long time ago
- weight: prefer clauses with fewer symbols

Combine them into a single scheme:

- have a priority queue ordering *Passive* for each criterion
- alternate between selecting from the queues using a fixed ratio

How is clause selection traditionally done?

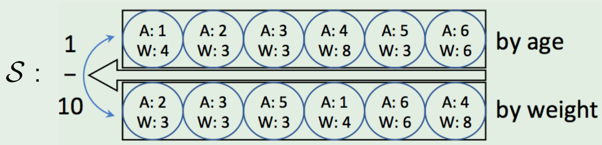
Take simple clause evaluation criteria:

- age: prefer clauses that were generated long time ago
- weight: prefer clauses with fewer symbols

Combine them into a single scheme:

- have a priority queue ordering *Passive* for each criterion
- alternate between selecting from the queues using a fixed ratio

Example (Organizing *Passive* via two priority queues)



Machine-learned Clause Selection Guidance

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], . . .

Machine-learned Clause Selection Guidance

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], ...

Technicalities:

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], ...

Technicalities:

- supervised learning setup
  - training examples $\sim$ the selected clauses

Machine-learned Clause Selection Guidance

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], ...

Technicalities:

- supervised learning setup
  - training examples $\sim$ the selected clauses
- clausal representations $\mathcal{R} : \mathcal{C} \rightarrow \mathbb{R}^n$
  - hand-crafted features, neural networks, ...

Machine-learned Clause Selection Guidance

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], ...

Technicalities:

- supervised learning setup
  - training examples $\sim$ the selected clauses
- clausal representations $\mathcal{R} : \mathcal{C} \to \mathbb{R}^n$
  - hand-crafted features, neural networks, ...
- learning algorithm
  - training yields a model $\mathcal{M} : \mathbb{R}^n \to \{0, 1\}$ (a binary classifier)

Machine-learned Clause Selection Guidance

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], ...

Technicalities:

- supervised learning setup
  - training examples $\sim$ the selected clauses
- clausal representations $\mathcal{R} : \mathcal{C} \to \mathbb{R}^n$
  - hand-crafted features, neural networks, ...
- learning algorithm
  - training yields a model $\mathcal{M} : \mathbb{R}^n \to \{0, 1\}$ (a binary classifier)
- integrating the learned advice back to the saturation loop

Adding the Learnt Advice $\mathcal{M}$ as Another Queue?

Priority:

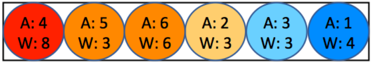
- sort by model's Y/N and tiebreak by age
- 

Adding the Learnt Advice $\mathcal{M}$ as Another Queue?

Priority:

- sort by model's Y/N and tiebreak by age
- 

Logits:

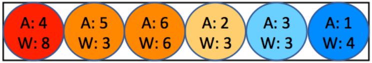
- even a binary classifier internally uses a real value
- 

Adding the Learnt Advice $\mathcal{M}$ as Another Queue?

Priority:

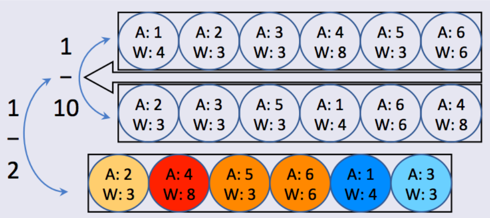
- sort by model's Y/N and tiebreak by age
- 

Logits:

- even a binary classifier internally uses a real value
- 

Combine with the original strategy

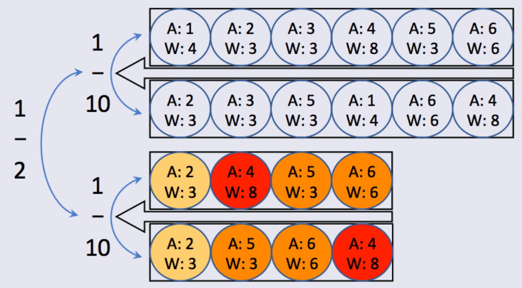
$\mathcal{S} \oplus \mathcal{M}^{1,0}$:



What Worked the Best?

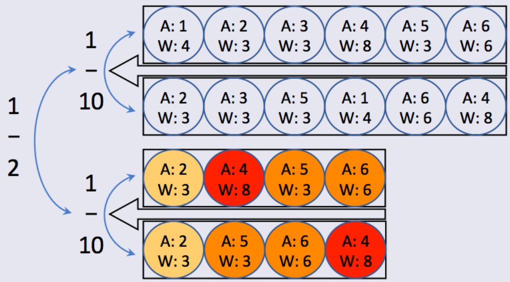
Layered Clause Selection [Tammet19,G&S20]:

$\mathcal{S} \oplus \mathcal{S}[\mathcal{M}^1]$:



Layered Clause Selection [Tammet19,G&S20]:

$\mathcal{S} \oplus \mathcal{S}[\mathcal{M}^1]$:



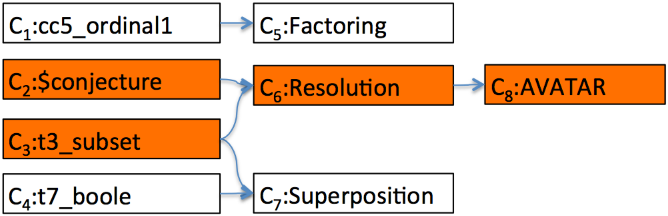
Advantages of LCS:

- keep using the well-tuned $\mathcal{S}$ also for the positively classified
- allows for the lazy evaluation trick [AITP20,CADE21]
- a smooth transition from the original to the ML-boosted

Outline

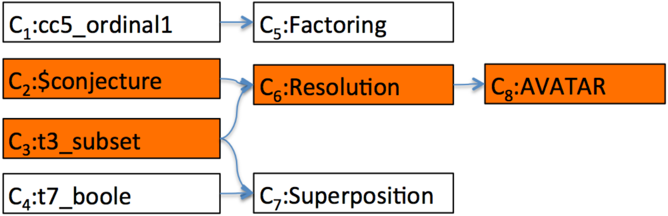
Represent Clauses by Their Derivation History

"Don't look at what the clause says, only where it's coming from."



Represent Clauses by Their Derivation History

"Don't look at what the clause says, only where it's coming from."



Focusing on the MIZAR dataset here:

- a large set of axioms $\mathcal{A}$ referenced by all the problems
- each problem P consists of a conjecture C_P and a $\mathcal{A}_P \subseteq \mathcal{A}$
- a small set of inference rules labeling the internal nodes

The idea of embeddings:

- represent each clause C by a real vector $v_C \in \mathbb{R}^n$

Recursive Neural Networks over Clause Derivations

The idea of embeddings:

- represent each clause C by a real vector $v_C \in \mathbb{R}^n$

Recursively compose the following neural building blocks:

- init function $I_A \in \mathbb{R}^n$, for every axiom type A
- deriv function $D_R : \mathbb{R}^n \times \cdots \times \mathbb{R}^n \to \mathbb{R}^n$, for every inference R
- eval function $E : \mathbb{R}^n \to \mathbb{R}$

Recursive Neural Networks over Clause Derivations

The idea of embeddings:

- represent each clause C by a real vector $v_C \in \mathbb{R}^n$

Recursively compose the following neural building blocks:

- init function $I_A \in \mathbb{R}^n$, for every axiom type A
- deriv function $D_R : \mathbb{R}^n \times \cdots \times \mathbb{R}^n \to \mathbb{R}^n$, for every inference R
- eval function $E : \mathbb{R}^n \to \mathbb{R}$

Example (Evaluating the derivation from the previous slide)

$$v_{C_2} := I_{\$conjecture}$$
$$v_{C_3} := I_{t3\_subset}$$
$$v_{C_6} := D_{Resolution}(v_{C_2}, v_{C_3})$$
$$v_{C_8} := D_{AVATAR}(v_{C_6})$$

C_8 is classified positive iff $E(v_{C_8}) \geq 0$

Recursive Neural Networks over Clause Derivations

The idea of embeddings:

- represent each clause C by a real vector $v_C \in \mathbb{R}^n$

Recursively compose the following neural building blocks:

- init function $I_A \in \mathbb{R}^n$, for every axiom type A
- deriv function $D_R : \mathbb{R}^n \times \cdots \times \mathbb{R}^n \to \mathbb{R}^n$, for every inference R
- eval function $E : \mathbb{R}^n \to \mathbb{R}$

Example (Evaluating the derivation from the previous slide)

$$
\begin{aligned}
v_{C_2} &:= I_{\$conjecture} \\
v_{C_3} &:= I_{t3\_subset} \\
v_{C_6} &:= D_{Resolution}(v_{C_2}, v_{C_3}) \\
v_{C_8} &:= D_{AVATAR}(v_{C_6})
\end{aligned}
$$

C_8 is classified positive iff $E(v_{C_8}) \geq 0$

➥ NB: Constant work per clause!

What Information Does the Network Mostly Pick Up On?

What Information Does the Network Mostly Pick Up On?

1) Axioms (and the init function)

- $|\mathcal{A}_{\mathrm{MIZAR}}| \approx 43$K, clipped to m=0.5/1/2K most frequent ones
- all others marked $l_{\$unknown}$
- the most important factor for good ATP performance

What Information Does the Network Mostly Pick Up On?

1) Axioms (and the init function)

- $|\mathcal{A}_{\mathrm{MIZAR}}| \approx$ 43K, clipped to m=0.5/1/2K most frequent ones
- all others marked $I_{\$unknown}$
- the most important factor for good ATP performance

2) Inference rules (and the deriv function)

- some form of deriv is obviously necessary for the recursion
- but the ability to distinguish rules $\sim$ extra 5% problems solved

What Information Does the Network Mostly Pick Up On?

1) Axioms (and the init function)

- $|\mathcal{A}_{\mathrm{MIZAR}}| \approx$ 43K, clipped to m=0.5/1/2K most frequent ones
- all others marked $l_{\$unknown}$
- the most important factor for good ATP performance

2) Inference rules (and the deriv function)

- some form of deriv is obviously necessary for the recursion
- but the ability to distinguish rules $\sim$ extra 5% problems solved

3) Conjecture relatedness

- in each problem, we mark conjecture clauses by $l_{\$conjecture}$
- the network learns to incorporate the right level of goal directedness

Outline

Experience with PyTorch

Experience with PyTorch

Training in Python $\longrightarrow$ inference from C++

- good experience with TorchScript model export
- almost any PyTorch code will get (VM-)interpreted in C++

Experience with PyTorch

Training in Python $\longrightarrow$ inference from C++

- good experience with TorchScript model export
- almost any PyTorch code will get (VM-)interpreted in C++

Dynamic computational graphs:

- elegant and flexible, but
- training needs to keep building them over and over!

Batching and Parallel Training Setup

Batching

- group derivations to create similarly-sized chunks for training
- merge equivalent nodes (within problem / across problems)

Batching and Parallel Training Setup

Batching

- group derivations to create similarly-sized chunks for training
- merge equivalent nodes (within problem / across problems)

➥ However, this is not SIMD $\rightarrow$ only trained on CPUs

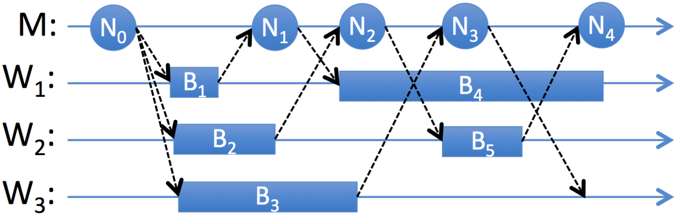
Batching and Parallel Training Setup

Batching

- group derivations to create similarly-sized chunks for training
- merge equivalent nodes (within problem / across problems)

➥ However, this is not SIMD $\rightarrow$ only trained on CPUs

Master-worker parallel training setup:



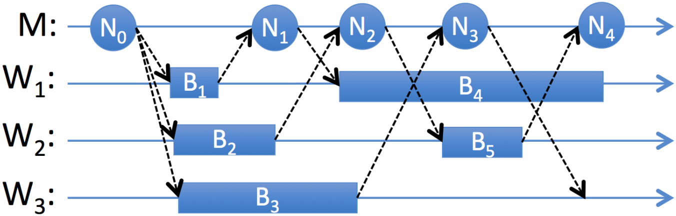
Batching and Parallel Training Setup

Batching

- group derivations to create similarly-sized chunks for training
- merge equivalent nodes (within problem / across problems)

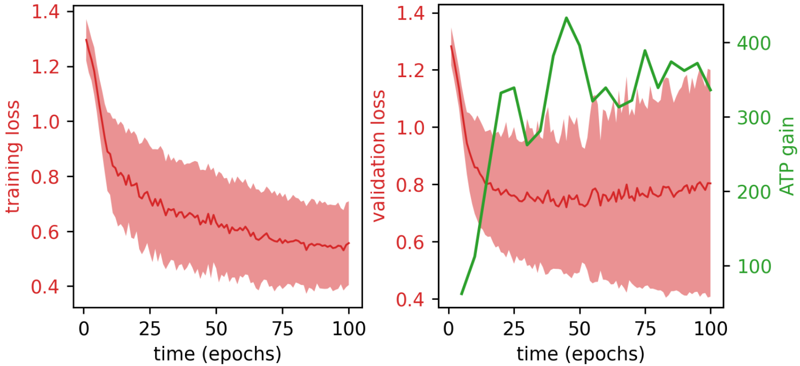
➥ However, this is not SIMD $\rightarrow$ only trained on CPUs

Master-worker parallel training setup:



➥ A funny "drift" effect that actually regularizes!

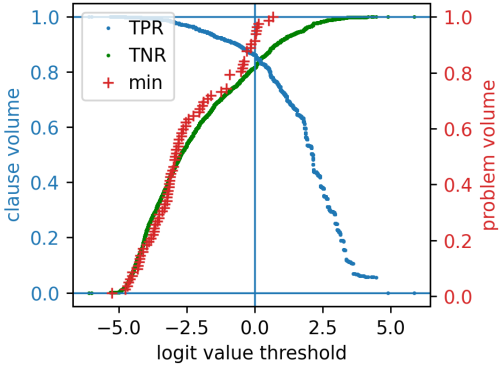
Validate and Compare to the ATP Performance



(from [CADE21]: Deepire for theory reasoning on SMTLIB)

Tweak Your Positive Bias

How often is $\mathcal{M}$ 100% correct?



[CADE21]: leaning a bit positively improves ATP performance

Outline

Experimental Setup

Experimental Setup

Mizar40 benchmark [Urban&Kaliszyk15]

- 57 880 problems in the TPTP format
- MPTP export from the Mizar Mathematical Library
- the small, *bushy* (i.e., re-proving), version

Experimental Setup

Mizar40 benchmark [Urban&Kaliszyk15]

- 57 880 problems in the TPTP format
- MPTP export from the Mizar Mathematical Library
- the small, *bushy* (i.e., re-proving), version

Fixed for the whole experiment:

- a base strategy $\mathcal{V}$:
  - previously shown to work well on Mizar40
- 10 s time limit

Training the First Models

Data preparation:

- $\mathcal{V}$ was able to solve 20 197 problems
- 800 MB of successful derivations (when zipped)
- 43 080 named Mizar axioms occurring in them
- largest derivation: 242 023 (merged) nodes

Data preparation:

- $\mathcal{V}$ was able to solve 20 197 problems
- 800 MB of successful derivations (when zipped)
- 43 080 named Mizar axioms occurring in them
- largest derivation: 242 023 (merged) nodes

How network size affects training:

model shorthand	$\mathcal{H}^{n128}$	$\mathcal{M}^{n64}$	$\mathcal{M}^{n128}$	$\mathcal{M}^{n256}$	$\mathcal{D}^{n128}$
revealed axioms m	0.5K	1K	1K	1K	2K
embedding size n	128	64	128	256	128
training time (min/epoch)	42	32	48	74	58
model size (MB)	4.6	1.6	5.0	17.9	5.8
best validation loss		0.455	0.455	0.452	

Training the First Models

Data preparation:

- $\mathcal{V}$ was able to solve 20 197 problems
- 800 MB of successful derivations (when zipped)
- 43 080 named Mizar axioms occurring in them
- largest derivation: 242 023 (merged) nodes

How network size affects training:

model shorthand	$\mathcal{H}^{n128}$	$\mathcal{M}^{n64}$	$\mathcal{M}^{n128}$	$\mathcal{M}^{n256}$	$\mathcal{D}^{n128}$
revealed axioms m	0.5K	1K	1K	1K	2K
embedding size n	128	64	128	256	128
training time (min/epoch)	42	32	48	74	58
model size (MB)	4.6	1.6	5.0	17.9	5.8
best validation loss		0.455	0.455	0.452	

Training the First Models

Data preparation:

- $\mathcal{V}$ was able to solve 20 197 problems
- 800 MB of successful derivations (when zipped)
- 43 080 named Mizar axioms occurring in them
- largest derivation: 242 023 (merged) nodes

How network size affects training:

model shorthand	$\mathcal{H}^{n128}$	$\mathcal{M}^{n64}$	$\mathcal{M}^{n128}$	$\mathcal{M}^{n256}$	$\mathcal{D}^{n128}$
revealed axioms m	0.5K	1K	1K	1K	2K
embedding size n	128	64	128	256	128
training time (min/epoch)	42	32	48	74	58
model size (MB)	4.6	1.6	5.0	17.9	5.8
best validation loss		0.455	0.455	0.452	

➥ Large capacity generalizes best!

Evaluation with the Prover

Num.probs solved by $\mathcal{V}$ and its RvNN boosted variants

strategy	$\mathcal{V}$	$\mathcal{H}^{n128}$	$\mathcal{M}^{n64}$	$\mathcal{M}^{n128}$	$\mathcal{M}^{n256}$	$\mathcal{D}^{n128}$
solved	20 197	24 581	25 484	25 805	25 287	26 014
$\mathcal{V}+$	+0	+5022	+5879	+6129	+5707	+6277
$\mathcal{V}-$	−0	−638	−592	−521	−617	−460
NN-eval.time	0 %	37.1 %	32.9 %	37.7 %	48.6 %	36.7 %

Points to note:

- $\mathcal{D}^{n128}$ solves almost 30 % more problems than $\mathcal{V}$

Evaluation with the Prover

Num.probs solved by $\mathcal{V}$ and its RvNN boosted variants

strategy	$\mathcal{V}$	$\mathcal{H}^{n128}$	$\mathcal{M}^{n64}$	$\mathcal{M}^{n128}$	$\mathcal{M}^{n256}$	$\mathcal{D}^{n128}$
solved	20 197	24 581	25 484	25 805	25 287	26 014
$\mathcal{V}+$	+0	+5022	+5879	+6129	+5707	+6277
$\mathcal{V}-$	−0	−638	−592	−521	−617	−460
NN-eval.time	0 %	37.1 %	32.9 %	37.7 %	48.6 %	36.7 %

Points to note:

- $\mathcal{D}^{n128}$ solves almost 30 % more problems than $\mathcal{V}$
- Could be even more greedy about the revealed axioms (m)

Evaluation with the Prover

Num.probs solved by $\mathcal{V}$ and its RvNN boosted variants

strategy	$\mathcal{V}$	$\mathcal{H}^{n128}$	$\mathcal{M}^{n64}$	$\mathcal{M}^{n128}$	$\mathcal{M}^{n256}$	$\mathcal{D}^{n128}$
solved	20 197	24 581	25 484	25 805	25 287	26 014
$\mathcal{V}+$	+0	+5022	+5879	+6129	+5707	+6277
$\mathcal{V}-$	−0	−638	−592	−521	−617	−460
NN-eval.time	0 %	37.1 %	32.9 %	37.7 %	48.6 %	36.7 %

Points to note:

- $\mathcal{D}^{n128}$ solves almost 30 % more problems than $\mathcal{V}$
- Could be even more greedy about the revealed axioms (m)
- Going over the embedding size ($n = 128$) makes it too slow

Evaluation with the Prover

Num.probs solved by $\mathcal{V}$ and its RvNN boosted variants

strategy	$\mathcal{V}$	$\mathcal{H}^{n128}$	$\mathcal{M}^{n64}$	$\mathcal{M}^{n128}$	$\mathcal{M}^{n256}$	$\mathcal{D}^{n128}$
solved	20 197	24 581	25 484	25 805	25 287	26 014
$\mathcal{V}+$	+0	+5022	+5879	+6129	+5707	+6277
$\mathcal{V}-$	−0	−638	−592	−521	−617	−460
NN-eval.time	0 %	37.1 %	32.9 %	37.7 %	48.6 %	36.7 %

Points to note:

- $\mathcal{D}^{n128}$ solves almost 30 % more problems than $\mathcal{V}$
- Could be even more greedy about the revealed axioms (m)
- Going over the embedding size ($n = 128$) makes it too slow
- It's fine to spend 40% of time *just thinking what the next clause should be* if it results in a good enough advice!

Looping and a Comparison to ENIGMA

Looping [Jakubův&Urban19]

- iterate the learning and solving phases
- keep learning also from the newly discovered proofs

➥ There: boosted tree learner over hand-crafted features

Looping and a Comparison to ENIGMA

Looping [Jakubův&Urban19]

- iterate the learning and solving phases
- keep learning also from the newly discovered proofs

➥ There: boosted tree learner over hand-crafted features

Performance comparison

	ENIGMA [J&U19]		Deepire		
loop	solved	$+\mathcal{S}$ %	solved	$+\mathcal{V}$ %	note
0	14 933	0.0	20 197	0.0	
1	20 366	35.8	26 014	28.8	$m = 2000$
2	22 839	52.3	27 348	35.4	$m = 3000$
3	23 467	56.5	28 947	43.3	$m = 5000$
4	23 753	58.4			
4’	25 397	70.0			

Points to note:

- both show the effect of diminishing returns

Looping and a Comparison to ENIGMA

Looping [Jakubův&Urban19]

- iterate the learning and solving phases
- keep learning also from the newly discovered proofs

➥ There: boosted tree learner over hand-crafted features

Performance comparison

	ENIGMA [J&U19]		Deepire		
loop	solved	$+\mathcal{S}$ %	solved	$+\mathcal{V}$ %	note
0	14 933	0.0	20 197	0.0	
1	20 366	35.8	26 014	28.8	$m = 2000$
2	22 839	52.3	27 348	35.4	$m = 3000$
3	23 467	56.5	28 947	43.3	$m = 5000$
4	23 753	58.4			
4'	25 397	70.0			

Points to note:

- both show the effect of diminishing returns
- ENIGMA climbs higher (relatively) from lower numbers

Outline

Conclusion

Summary

- Deepire explores ENIGMA-style clause selection guidance deliberately focusing on just derivation history
- informedness / speed of evaluation balance
- convincing results on SMTLIB / MIZAR40

Summary

- Deepire explores ENIGMA-style clause selection guidance deliberately focusing on just derivation history
- informedness / speed of evaluation balance
- convincing results on SMTLIB / MIZAR40

➥ `https://github.com/quickbeam123/deepire3.1`

Conclusion

Summary

- Deepire explores ENIGMA-style clause selection guidance deliberately focusing on just derivation history
- informedness / speed of evaluation balance
- convincing results on SMTLIB / MIZAR40

➥ `https://github.com/quickbeam123/deepire3.1`

Open

- positive example selection is more tricky than it seems (AVATAR, LRS, but already in DISCOUNT)
- classification vs regression
- What did the model actually learn? (XAI)

Conclusion

Summary

- Deepire explores ENIGMA-style clause selection guidance deliberately focusing on just derivation history
- informedness / speed of evaluation balance
- convincing results on SMTLIB / MIZAR40

➥ `https://github.com/quickbeam123/deepire3.1`

Open

- positive example selection is more tricky than it seems (AVATAR, LRS, but already in DISCOUNT)
- classification vs regression
- What did the model actually learn? (XAI)

Outlook

- looping is the first step towards RL
- general ATP knowledge rather than benchmark-specific!

Conclusion

Summary

- Deepire explores ENIGMA-style clause selection guidance deliberately focusing on just derivation history
- informedness / speed of evaluation balance
- convincing results on SMTLIB / MIZAR40

➥ `https://github.com/quickbeam123/deepire3.1`

Open

- positive example selection is more tricky than it seems (AVATAR, LRS, but already in DISCOUNT)
- classification vs regression
- What did the model actually learn? (XAI)

Outlook

- looping is the first step towards RL
- general ATP knowledge rather than benchmark-specific!

Thank you for attention!