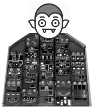
Deepire: First Experiments with Neural Guidance in Vampire

Martin Suda

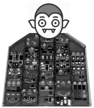
Czech Technical University in Prague, Czech Republic

AITP, September 2020



Vampire

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

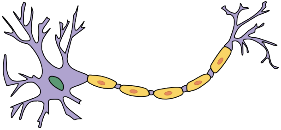


Vampire

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

Neural (internal) guidance

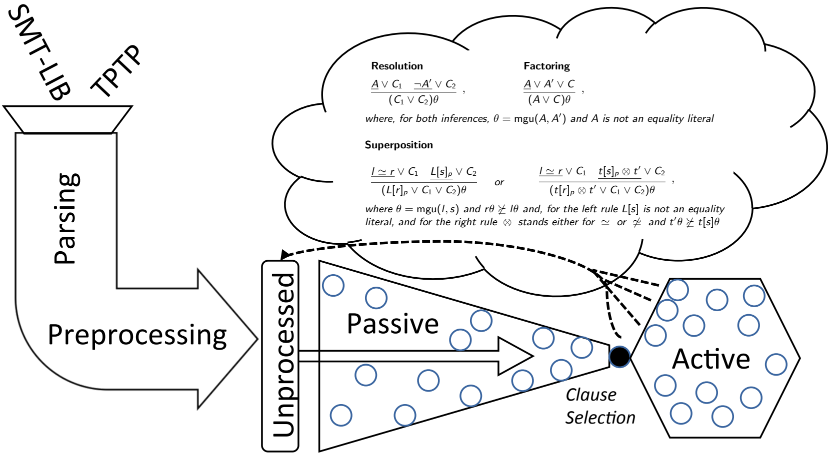
- targeting the clause selection decision point
- supervised learning from successful runs

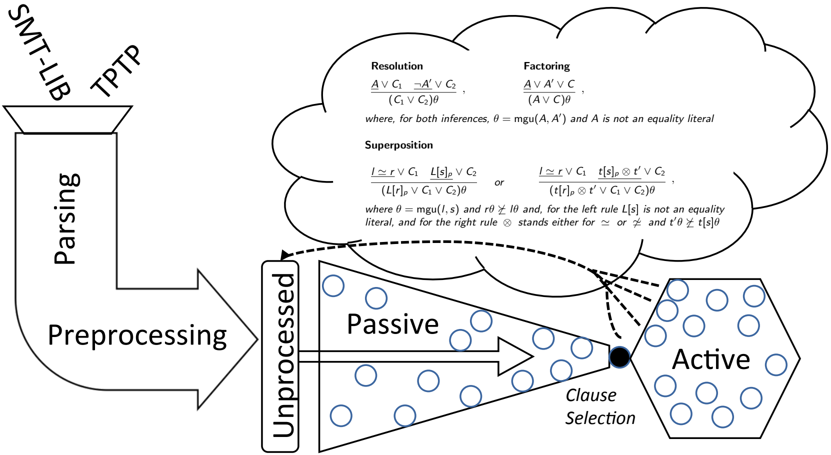


Outline

Outline

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:

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

How is clause selection ~~traditionally~~ done?

Take simple clause evaluation criteria:

- weight: prefer clauses with fewer symbols
- age: prefer clauses that were generated long time ago
- ...
- neural estimate of clause's usefulness

How is clause selection traditionally done?

Take simple clause evaluation criteria:

- weight: prefer clauses with fewer symbols
- age: prefer clauses that were generated long time ago
- ...
- neural estimate of clause's usefulness

Combine these into a single scheme:

- for each criterion ξ maintain a priority queue which orders Passive by ξ
- alternate between selecting from the queues using a fixed ratio; e.g. pick 5 times the smallest, 1 time the oldest, repeat

Outline

Stepping up on the Shoulders of the Giants

Mostly inspired by ENIGMA:

- ENIGMA: Efficient Learning-Based Inference Guiding Machine [Jakubův&Urban,2017]
- ENIGMA-NG: Efficient Neural and Gradient-Boosted Inference Guidance for E [Chvalovský et al.,2019]
- ENIGMA Anonymous: Symbol-Independent Inference Guiding Machine [Jakubův et al.,2020]

See also:

- Deep Network Guided Proof Search [Loos et al.,2017]
- Property Invariant Embedding for Automated Reasoning [Olšák et al.,2020]

Stepping up on the Shoulders of the Giants

Mostly inspired by ENIGMA:

- ENIGMA: Efficient Learning-Based Inference Guiding Machine [Jakubův&Urban,2017]
- ENIGMA-NG: Efficient Neural and Gradient-Boosted Inference Guidance for E [Chvalovský et al.,2019]
- ENIGMA Anonymous: Symbol-Independent Inference Guiding Machine [Jakubův et al.,2020]

See also:

- Deep Network Guided Proof Search [Loos et al.,2017]
- Property Invariant Embedding for Automated Reasoning [Olšák et al.,2020]

Things to consider:

- Evaluation speed
- Aligned signatures across problems?
- Can the choices depend on proof state?
- How exactly is the new advice integrated into the ATP?

My current “doctrine” for clause selection research

Keep it at simple as possible!

- start with small models
- feed them with abstractions only

My current "doctrine" for clause selection research

Keep it at simple as possible!

- start with small models
- feed them with abstractions only

Why?

- As a form of regularisation
  (Followed by "overfitting without shame")
- Explainability
  (Could we glean new "heuristics in the old-fashioned sense"?)

My current "doctrine" for clause selection research

Keep it at simple as possible!

- start with small models
- feed them with abstractions only

Why?

- As a form of regularisation
  (Followed by "overfitting without shame")
- Explainability
  (Could we glean new "heuristics in the old-fashioned sense"?)

Idea explored here:

- Learn from clause derivation history!

Outline

Basic architecture

Simple TreeNN over derivation trees of clauses

- leaf: user axiom, conjecture, theory axiom id:
  int_plus_commut, int_mult_assoc, ...
- node: inference rule id:
  superposition, demodulation, resolution, ...

Simple TreeNN over derivation trees of clauses

- leaf: user axiom, conjecture, theory axiom id:
  int_plus_commut, int_mult_assoc, ...
- node: inference rule id:
  superposition, demodulation, resolution, ...

➥ Finite enums: learnable embeddings + small MLPs

Basic architecture

Simple TreeNN over derivation trees of clauses

- leaf: user axiom, conjecture, theory axiom id:
  int_plus_commut, int_mult_assoc, ...
- node: inference rule id:
  superposition, demodulation, resolution, ...

➥ Finite enums: learnable embeddings + small MLPs

Properties:

- constant work per clause!
- signature agnostic
- intentionally no explicit proof state
- possible intuition: generalizes age

Obtaining the advice

What do we learn from?

- a complete list of selected clauses from a successful run
- mark as positive those that ended up in the found proof

➥ Common to all previous approaches.

Obtaining the advice

What do we learn from?

- a complete list of selected clauses from a successful run
- mark as positive those that ended up in the found proof

➥ Common to all previous approaches.

What do we learn?

- a binary classifier heavily biased to err on the negative side
- i.e. try to classify 100% of positive clause as positive and see how much can be thrown away on the negative side

➥ This is new stuff!

Integrating the advice

What has been tried:

- neural estimate (i.e., the "logits") orders clauses on a new separate clause queue
- ENIMGA: just classify (put all good before any bad) and break ties by age within the positive and negative groups

Integrating the advice

What has been tried:

- neural estimate (i.e., the "logits") orders clauses on a new separate clause queue
- ENIMGA: just classify (put all good before any bad) and break ties by age within the positive and negative groups

Here: layered clause selection [Tammet19,Gleiss&Suda20]

- layer one: age-weight selection as described earlier
- layer two: group clauses into good and bad
  1. have a layer-two ratio to always pick a group
  2. do layer-one selection in that group as before

What has been tried:

- neural estimate (i.e., the “logits”) orders clauses on a new separate clause queue
- ENIMGA: just classify (put all good before any bad) and break ties by age within the positive and negative groups

Here: layered clause selection [Tammet19,Gleiss&Suda20]

- layer one: age-weight selection as described earlier
- layer two: group clauses into good and bad
  1. have a layer-two ratio to always pick a group
  2. do layer-one selection in that group as before

➥ Delayed evaluation trick:
time spent evaluating dropped from around 90% to 30%

Outline

Experiments

Learning:

- Tanh for all non-linearities, various embedding sizes
- overfit to the dataset; ATP eval as the final judge
- positive examples weigh 10 time more than negative

Experiments

Learning:

- Tanh for all non-linearities, various embedding sizes
- overfit to the dataset; ATP eval as the final judge
- positive examples weigh 10 time more than negative

Evaluation:

- TPTP version 7.3 (CNF, FOF, TF0):
  18 294 problems
- a subset of SMTLIB (quantified; without BV, FP):
  20 795 problems

➥ Neither has aligned signatures (besides the theory part)

Experiments

Learning:

- Tanh for all non-linearities, various embedding sizes
- overfit to the dataset; ATP eval as the final judge
- positive examples weigh 10 time more than negative

Evaluation:

- TPTP version 7.3 (CNF, FOF, TF0):
  18 294 problems
- a subset of SMTLIB (quantified; without BV, FP):
  20 795 problems

➥ Neither has aligned signatures (besides the theory part)

base strategy = discount, awr = 1:5, av = off

Time limit 5 s per problem – also for running with the model!

Results on TPTP – let's not look at them (yet)

```
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-10.1.pkl 7166 -1052
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-5.1.pkl 7332 -886
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-2.1.pkl 7628 -590
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-1.1.pkl 7798 -420
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-1.2.pkl 7877 -341
problemsFOL_deepire3_5s_d4861_model-77Tanh_p98n19_nesqr-10.1.pkl 7884 -334
problemsFOL_deepire3_5s_d4861_model-10Tanh_p99n19_nesqr-100.1.pkl 7895 -323
problemsFOL_deepire3_5s_d4861_model-77Tanh_p98n19_nesqr-5.1.pkl 7897 -321
problemsFOL_deepire3_5s_d4861_model-10Tanh_p99n19_nesqr-10.1.pkl 7913 -305
problemsFOL_deepire3_5s_d4861_model-10Tanh_p99n19_nesqr-1.1.pkl 7942 -276
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-1.5.pkl 7958 -260
problemsFOL_deepire3_5s_d4861_model-77Tanh_p98n19_nesqr-1.1.pkl 7974 -244
problemsFOL_deepire3_5s_d4861_model-77Tanh_p98n19_nesqr-1.5.pkl 8002 -216
problemsFOL_deepire3_5s_d4858_fastBase0.pkl 8218 0

Greedy cover:
problemsFOL_deepire3_5s_d4858_fastBase0.pkl contributes 8218 total 8218 uniques 163
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-2.1.pkl contributes 322 total 7628 uniques 12
problemsFOL_deepire3_5s_d4861_model-77Tanh_p98n19_nesqr-10.1.pkl contributes 72 total 7884 uniques 7
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-1.5.pkl contributes 58 total 7958 uniques 24
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-10.1.pkl contributes 47 total 7166 uniques 30
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-1.2.pkl contributes 16 total 7877 uniques 7
problemsFOL_deepire3_5s_d4861_model-10Tanh_p99n19_nesqr-10.1.pkl contributes 13 total 7913 uniques 5
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-5.1.pkl contributes 12 total 7332 uniques 11
problemsFOL_deepire3_5s_d4861_model-77Tanh_p98n19_nesqr-1.1.pkl contributes 10 total 7974 uniques 7
problemsFOL_deepire3_5s_d4861_model-55Tanh_p77n67_nesqr-1.1.pkl contributes 9 total 7798 uniques 9
problemsFOL_deepire3_5s_d4861_model-10Tanh_p99n19_nesqr-100.1.pkl contributes 4 total 7895 uniques 4
problemsFOL_deepire3_5s_d4861_model-10Tanh_p99n19_nesqr-1.1.pkl contributes 2 total 7942 uniques 1
problemsFOL_deepire3_5s_d4861_model-77Tanh_p98n19_nesqr-5.1.pkl contributes 2 total 7897 uniques 2
problemsFOL_deepire3_5s_d4861_model-77Tanh_p98n19_nesqr-1.5.pkl contributes 1 total 8002 uniques 1
Total 8786
```

Results on SMTLIB – two levels of "looping"

model	ratio	solved	delta
base	—	447	0
m14	10:1	526	79
m14	5:1	528	81
m14	1:1	553	106
m41	1:5	555	108
m41	10:1	578	131
m14	1:5	580	133
m41	5:1	581	134
m41	1:1	592	145
m99-p99n56	1:5	650	203
m99-p99n56	5:1	699	252
m99-p99n56	10:1	708	261
m99-p99n56	20:1	713	266
m99-p99n56	1:1	735	288

Results on SMTLIB – greedy cover

model	ratio	contributes	(total)	uniques
m99-p99n56	1.1	735	735	39
m99-p99n56	20.1	56	713	13
base	—	40	**447**	15
m41	10.1	14	578	5
m14	1:5	8	580	0
m41	5.1	4	581	2
...		...		
Union		**868**		

The last (official) slide

How to get even better numbers?

- Add more features: SInE levels, AVATAR, length, ...
- Do more looping
- "Time hook" idea

The last (official) slide

How to get even better numbers?

- Add more features: SInE levels, AVATAR, length, ...
- Do more looping
- "Time hook" idea

What's wrong with TPTP?

- only a small subset contains theories
- too "non-uniform"?
- some crazy deep proofs ("computational" rather than search)

The last (official) slide

How to get even better numbers?

- Add more features: SInE levels, AVATAR, length, ...
- Do more looping
- "Time hook" idea

What's wrong with TPTP?

- only a small subset contains theories
- too "non-uniform"?
- some crazy deep proofs ("computational" rather than search)

As a next step

- a careful analysis of how to influence (ATP) generalization

The last (official) slide

How to get even better numbers?

- Add more features: SInE levels, AVATAR, length, ...
- Do more looping
- "Time hook" idea

What's wrong with TPTP?

- only a small subset contains theories
- too "non-uniform"?
- some crazy deep proofs ("computational" rather than search)

As a next step

- a careful analysis of how to influence (ATP) generalization

Thank you for attention!

Technicalities

- PyTorch 1.6 / export model via TorchScript
- (Sigmoid + binary cross-entropy loss)
- Tanh for now; try gradient clipping and ReLU next
- (a dropout-like trick; no ablation yet, though)
- training on per-problem basis $\sim$ mini-batch
  - one little forest
  - (could merge multiple-ones)
- building the forst: 1s, backward: 0.7s, optimiser.step: 0.01s
- How to parallelize?
  - Master/slaves architecture: one master model one optimiser; send out copies and collect gradient updates "asynchronously"