Two Learning Operators for Clause Selection Guidance: An Experimental Evaluation

Martin Suda*

Czech Technical University in Prague, Czech Republic

AITP, Aussios, September 2024

*Supported by the project RICAIP no. 857306 under the EU-H2020 programme and the Czech Science Foundation project no. 24-12759S.

Machine Learning Boosted Automated Theorem Proving

Machine Learning Boosted Automated Theorem Proving

ATP technology:

Machine Learning Boosted Automated Theorem Proving

ATP technology: saturation-based

Machine Learning Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Machine Learning Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost:

Machine Learning Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

Machine Learning Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

- the most important choice point
- "selecting the proof clauses" intuition

Machine Learning Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

- the most important choice point
- "selecting the proof clauses" intuition

Two main approaches to date:

Machine Learning Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

- the most important choice point
- "selecting the proof clauses" intuition

Two main approaches to date:

- ENIGMA-style
- RL-inspired

Machine Learning Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

- the most important choice point
- "selecting the proof clauses" intuition

Two main approaches to date:

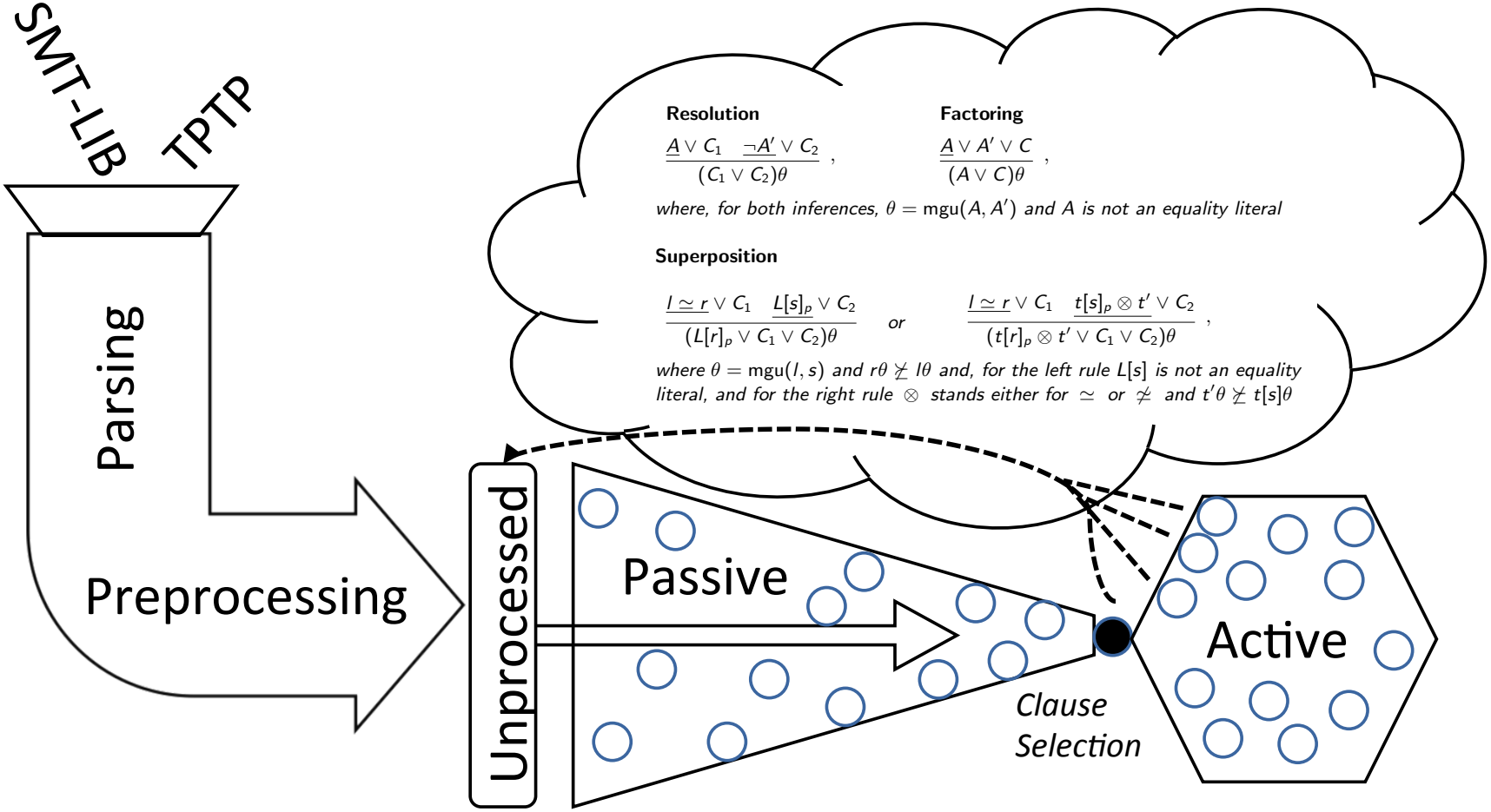
- ENIGMA-style
- RL-inspired

What are the differences? What is the same? Which one is better?

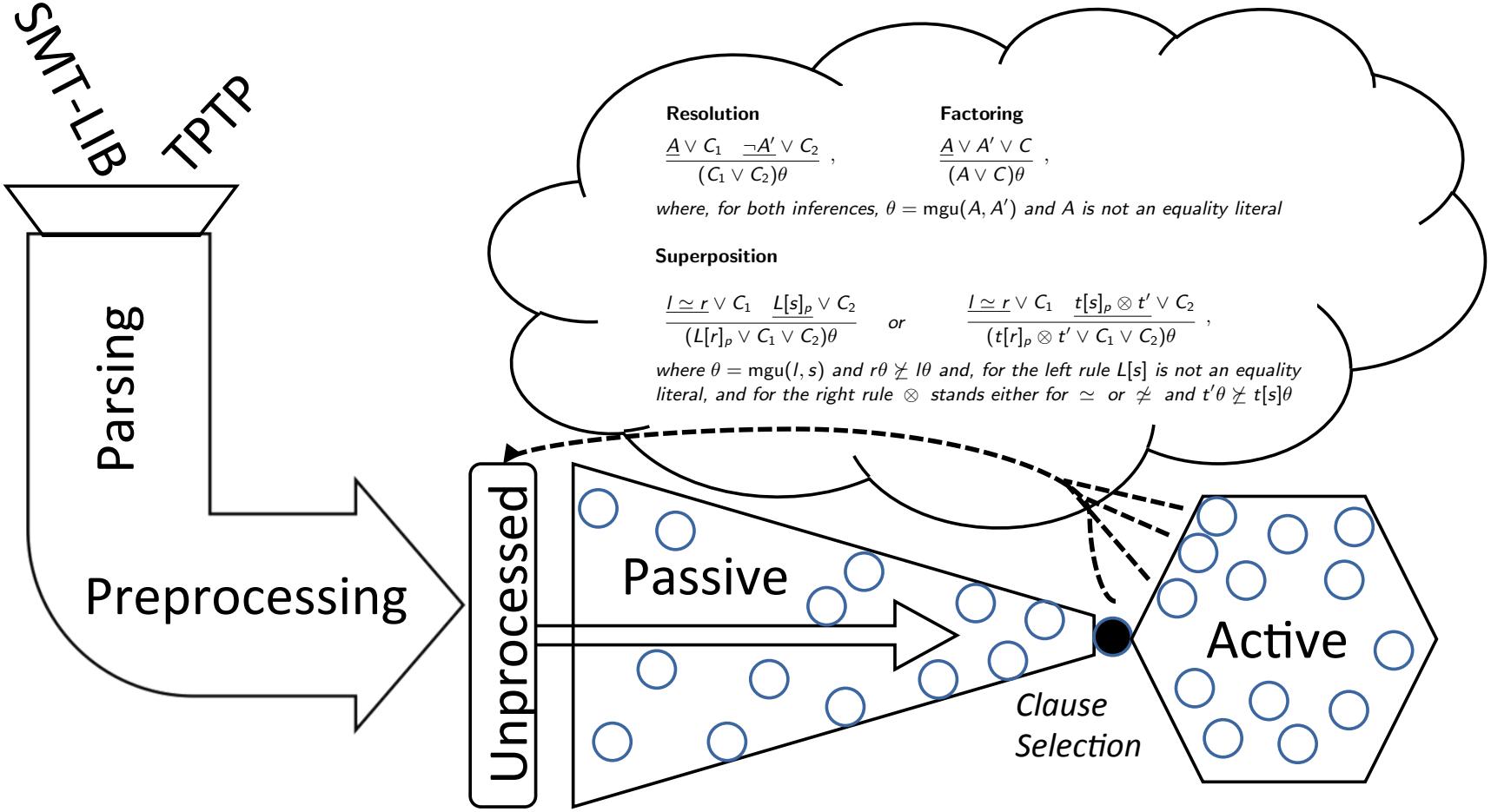
Outline

Outline

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:

- age: prefer clauses that were generated long time ago
- weight: 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?

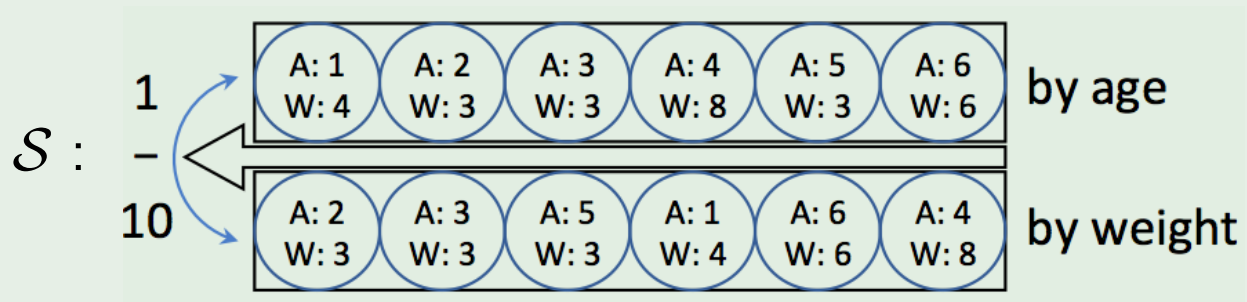
Take simple clause evaluation criteria:

- 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)



Outline

ENIGMA-style

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], [Loos et al.'17], ...

ENIGMA-style

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], [Loos et al.'17], ...

The "pos/neg"s of E:

E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

ENIGMA-style

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], [Loos et al.'17], ...

The "pos/neg"s of E:

E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

Next comes the ML:

- represent those clauses somehow (features, NNs, ...)
- train a binary classifier on the task
- integrate back with the prover:

ENIGMA-style

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], [Loos et al.'17], ...

The "pos/neg"s of E:

E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

Next comes the ML:

- represent those clauses somehow (features, NNs, ...)
- train a binary classifier on the task
- integrate back with the prover: "try to do more of the pos"

Possible Ways of Integrating the Learnt Advice

Priority:

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

Possible Ways of Integrating the Learnt Advice

Priority:

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

Logits:

- even a binary classifier internally uses a real value
- 

Possible Ways of Integrating the Learnt Advice

Priority:

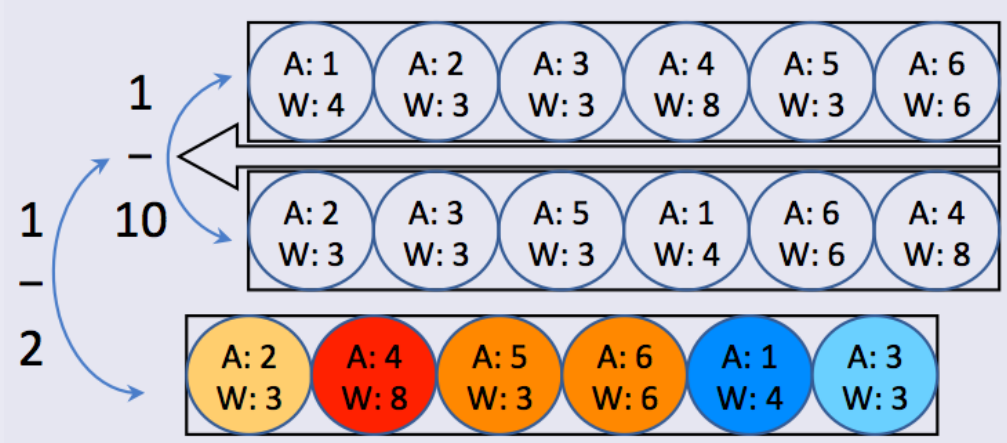
- sort by model's Y/N and tiebreak by age
- 

Logits:

- even a binary classifier internally uses a real value
- 

Combine with the original strategy

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



Outline

Saturation as an Reinforcement-Learning Environment

What ATP heuristics would the aliens come up with?

Saturation as an Reinforcement-Learning Environment

What ATP heuristics would the aliens come up with?

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

Saturation as an Reinforcement-Learning Environment

What ATP heuristics would the aliens come up with?

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

State

- static - the conjecture we are trying to prove
- evolving - the internal state of the prover at particular moment

Saturation as an Reinforcement-Learning Environment

What ATP heuristics would the aliens come up with?

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

State

- static - the conjecture we are trying to prove
- evolving - the internal state of the prover at particular moment

Reward

- Score 1 point for solving a problem (within the time limit)

Saturation as an Reinforcement-Learning Environment

What ATP heuristics would the aliens come up with?

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

State

- static - the conjecture we are trying to prove
- evolving - the internal state of the prover at particular moment

Reward

- Score 1 point for solving a problem (within the time limit) ???

Saturation as an Reinforcement-Learning Environment

What ATP heuristics would the aliens come up with?

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

State

- static - the conjecture we are trying to prove
- evolving - the internal state of the prover at particular moment

Reward

- Score 1 point for solving a problem (within the time limit) ???

➥ TRAIL [Crouse et al.'21], [McKeown'23], [Shminke'23], ...

Policy Gradient and REINFORCE [Williams'92]

The (evolving) state s of an ATP is a large amorphous blob:

- value-based methods (Q-learning, DQN, ...) seem hopeless

Policy Gradient and REINFORCE [Williams'92]

The (evolving) state s of an ATP is a large amorphous blob:

- value-based methods (Q-learning, DQN, ...) seem hopeless

Instead, with policy gradient methods, we train a network to directly predict the policy $\pi(a|\not{s};\theta)$

Policy Gradient and REINFORCE [Williams'92]

The (evolving) state s of an ATP is a large amorphous blob:

- value-based methods (Q-learning, DQN, ...) seem hopeless

Instead, with policy gradient methods, we train a network to directly predict the policy $\pi(a|\not{s};\theta)$

- can sample actions according to the distribution π

Policy Gradient and REINFORCE [Williams'92]

The (evolving) state s of an ATP is a large amorphous blob:

- value-based methods (Q-learning, DQN, ...) seem hopeless

Instead, with policy gradient methods, we train a network to directly predict the policy $\pi(a|\not{s};\theta)$

- can sample actions according to the distribution π
- imperfect information $\Rightarrow$ the optimal policy may be stochastic!

Policy Gradient and REINFORCE [Williams'92]

The (evolving) state s of an ATP is a large amorphous blob:

- value-based methods (Q-learning, DQN, ...) seem hopeless

Instead, with policy gradient methods, we train a network to directly predict the policy $\pi(a|\not{s};\theta)$

- can sample actions according to the distribution π
- imperfect information $\Rightarrow$ the optimal policy may be stochastic!

Policy Gradient Theorem

$$\nabla_\theta v_\pi(s_{initial}) \propto \mathbb{E}_{s\sim\mu}\mathbb{E}_{a\sim\pi} q_\pi(s,a)\nabla_\theta \ln \pi(a|s;\theta)$$

Policy Gradient and REINFORCE [Williams'92]

The (evolving) state s of an ATP is a large amorphous blob:

- value-based methods (Q-learning, DQN, ...) seem hopeless

Instead, with policy gradient methods, we train a network to directly predict the policy $\pi(a|\not{s};\theta)$

- can sample actions according to the distribution π
- imperfect information $\Rightarrow$ the optimal policy may be stochastic!

Policy Gradient Theorem

$$\nabla_\theta v_\pi(s_{initial}) \propto \mathbb{E}_{s\sim\mu}\mathbb{E}_{a\sim\pi} q_\pi(s,a)\nabla_\theta \ln \pi(a|s;\theta)$$

The devil in the details:

- with $\pi(C|s_i;\theta) = \mathrm{softmax}\left([\mathrm{NN}_\theta(features_C)]_{C\in Passive_i}\right)$, the "$\nabla_\theta \ln \pi$"-bit boils down to the usual NLL loss

Policy Gradient and REINFORCE [Williams'92]

The (evolving) state s of an ATP is a large amorphous blob:

- value-based methods (Q-learning, DQN, ...) seem hopeless

Instead, with policy gradient methods, we train a network to directly predict the policy $\pi(a|\not{s};\theta)$

- can sample actions according to the distribution π
- imperfect information $\Rightarrow$ the optimal policy may be stochastic!

Policy Gradient Theorem

$$\nabla_\theta v_\pi(s_{initial}) \propto \mathbb{E}_{s\sim\mu}\mathbb{E}_{a\sim\pi} q_\pi(s,a)\nabla_\theta \ln \pi(a|s;\theta)$$

The devil in the details:

- with $\pi(C|s_i;\theta) = \mathrm{softmax}\left([\mathrm{NN}_\theta(features_C)]_{C\in Passive_i}\right)$, the "$\nabla_\theta \ln \pi$"-bit boils down to the usual NLL loss
- for $q_\pi(s,C)$ we simply pick $\mathbb{I}_{\text{Did } C \text{ show up in the found proof?}}$

Outline

Basic Comparison I

Starts with:

- ENIGMA-style: a working clause selection heuristic
- RL-inspired: “tabula rasa”

Basic Comparison I

Starts with:

- ENIGMA-style: a working clause selection heuristic
- RL-inspired: “tabula rasa”

Training data:

- ENIGMA-style: pos/neg; over selected only (static)
- RL-inspired: traces; over all the generated (changes in time)

Basic Comparison I

Starts with:

- ENIGMA-style: a working clause selection heuristic
- RL-inspired: "tabula rasa"

Training data:

- ENIGMA-style: pos/neg; over selected only (static)
- RL-inspired: traces; over all the generated (changes in time)

Attractor:

- Both: clauses from found proofs

Basic Comparison I

Starts with:

- ENIGMA-style: a working clause selection heuristic
- RL-inspired: "tabula rasa"

Training data:

- ENIGMA-style: pos/neg; over selected only (static)
- RL-inspired: traces; over all the generated (changes in time)

Attractor:

- Both: clauses from found proofs

Integrating the learned advice:

- ENIGMA-style: combine with your original heuristic
- RL-inspired: one queue sorted by the predicted scores

Basic Comparison II

Model:

- ENIGMA-style: a binary classifier
- RL-inspired: regression (logits) $\Rightarrow$ action probabilities

Basic Comparison II

Model:

- ENIGMA-style: a binary classifier
- RL-inspired: regression (logits) $\Rightarrow$ action probabilities

Loss function (for the neural incarnations):

- ENIGMA-style: binary cross entropy (NLL)
- RL-inspired: weighted NLL (weights $\sim$ returns)

Basic Comparison II

Model:

- ENIGMA-style: a binary classifier
- RL-inspired: regression (logits) $\Rightarrow$ action probabilities

Loss function (for the neural incarnations):

- ENIGMA-style: binary cross entropy (NLL)
- RL-inspired: weighted NLL (weights $\sim$ returns)

Iterative improvement:

- Both: yes (ENIGMA calls it "looping")

How High Can It Fly?

Architecture

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- a neural part: $\mathrm{MLP}(\mathit{features}_C) \rightarrow \mathit{logit}$

How High Can It Fly?

Architecture

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- a neural part: $\mathrm{MLP}(\mathit{features}_C) \rightarrow \mathit{logit}$

Experimental setup

- extend Vampire theorem prover
- 3000 randomly select TPTP problems (FOF/CNF)
- time limit: $\sim$10s per problem

How High Can It Fly?

Architecture

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- a neural part: $\mathrm{MLP}(\mathit{features}_C) \rightarrow \mathit{logit}$

Experimental setup

- extend Vampire theorem prover
- 3000 randomly select TPTP problems (FOF/CNF)
- time limit: $\sim$10s per problem

ENIGMA-style

- unfortunately, did not manage to beat the baseline

How High Can It Fly?

Architecture

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- a neural part: $\mathrm{MLP}(\mathit{features}_C) \rightarrow \mathit{logit}$

Experimental setup

- extend Vampire theorem prover
- 3000 randomly select TPTP problems (FOF/CNF)
- time limit: $\sim$10s per problem

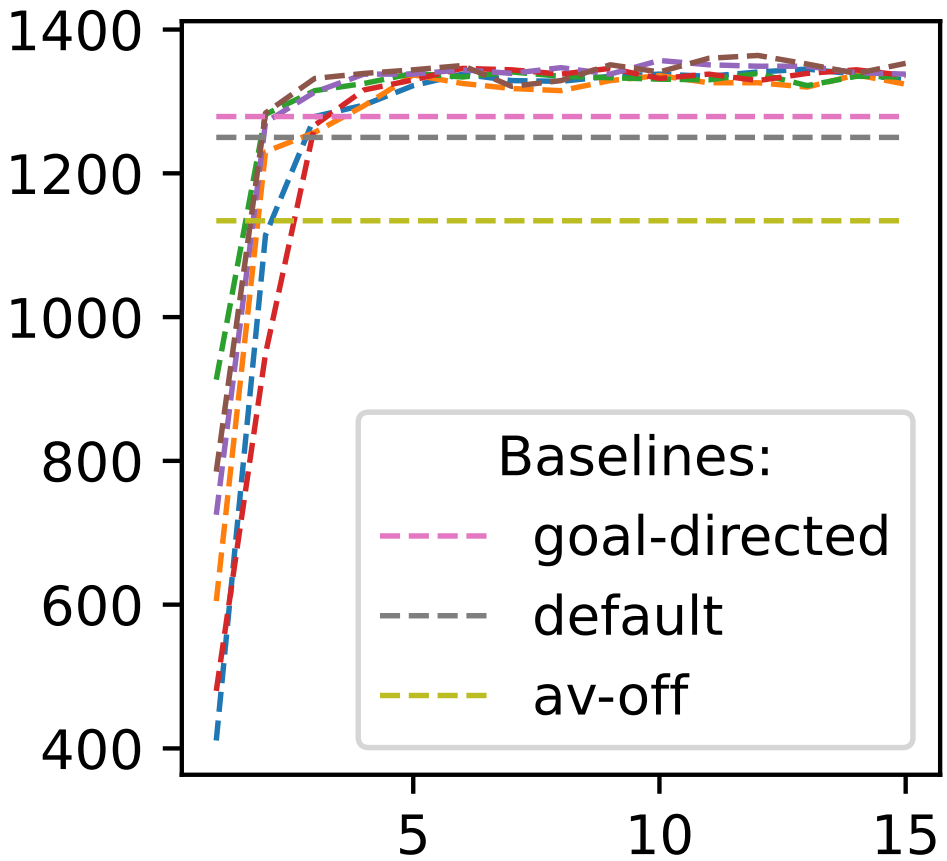
ENIGMA-style

- unfortunately, did not manage to beat the baseline

RL-inspired

- can beat the default strategy by 6%,
  (a good goal-directed strategy by 3.5%) on the test set

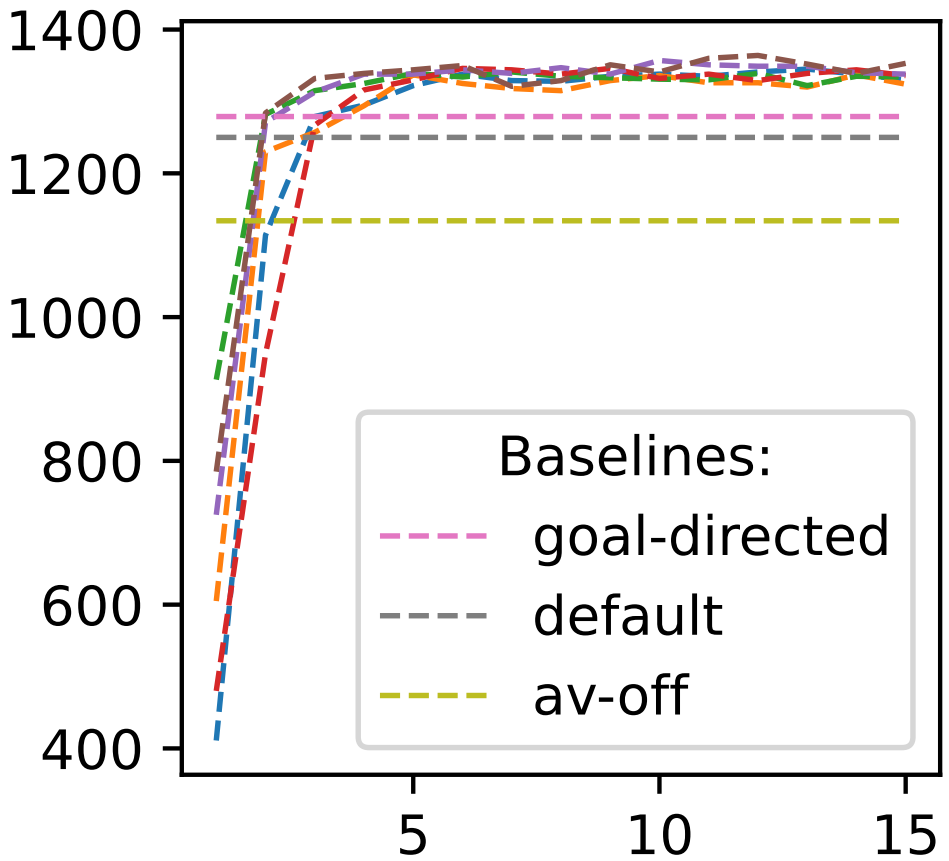
Training with the RL-inspired Operator



Observations:

- random initialization $\Rightarrow$ different start performance

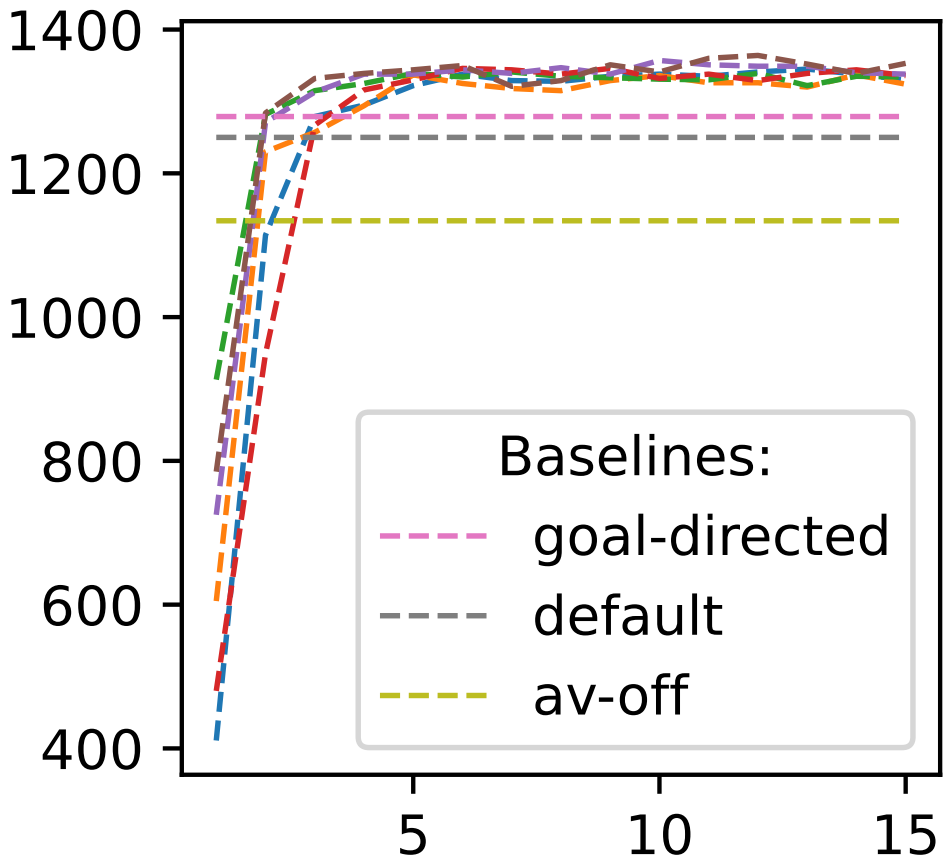
Training with the RL-inspired Operator



Observations:

- random initialization $\Rightarrow$ different start performance
- hidden layer size 16, 32, 64, 128, 256 all similar performance

Training with the RL-inspired Operator

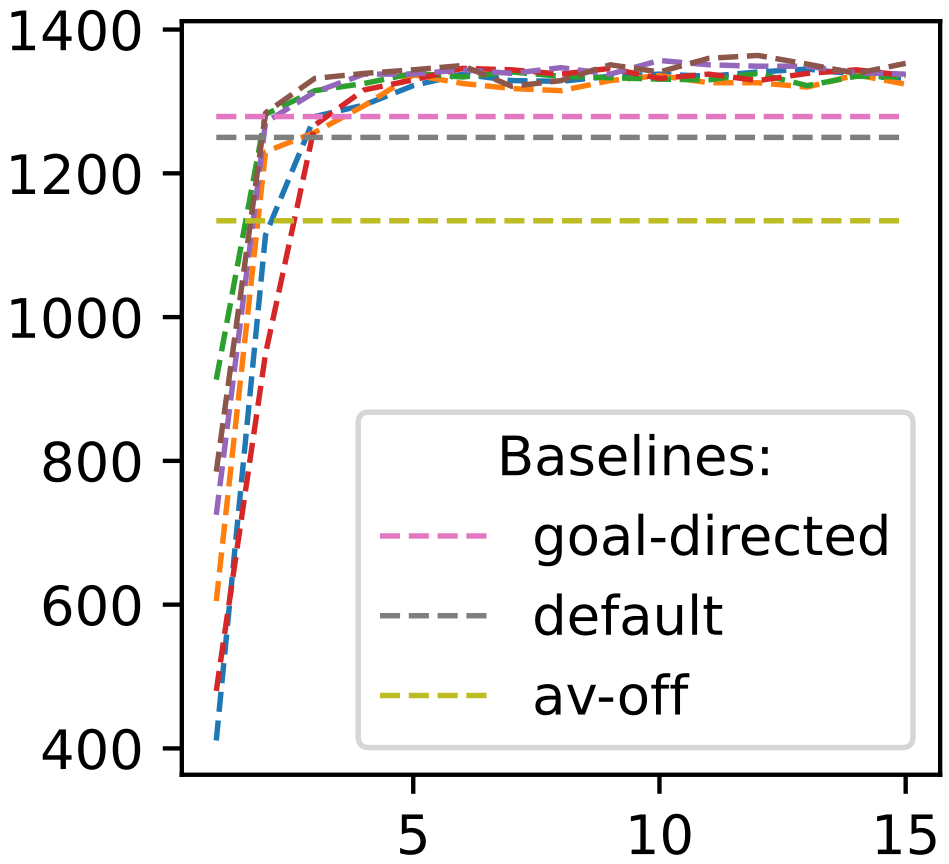


Observations:

- random initialization $\Rightarrow$ different start performance
- hidden layer size 16, 32, 64, 128, 256 all similar performance

A first (modest) ML-based improvement of a SoTA ATP on TPTP!

Training with the RL-inspired Operator



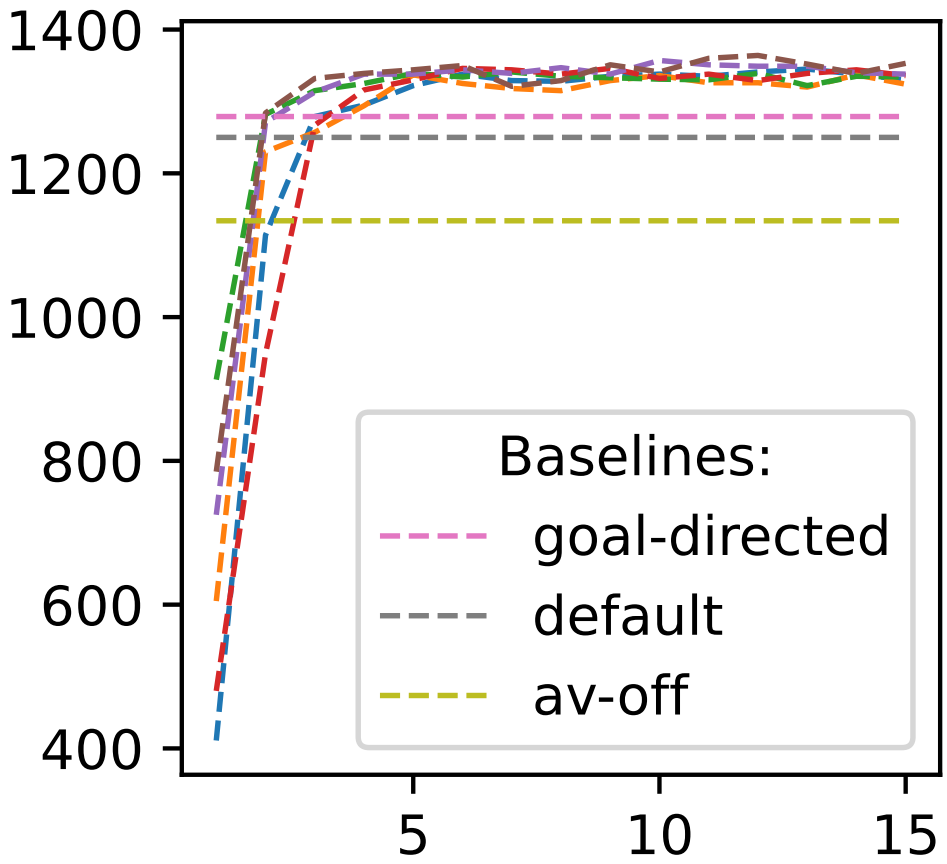
Observations:

- random initialization $\Rightarrow$ different start performance
- hidden layer size 16, 32, 64, 128, 256 all similar performance

A first (modest) ML-based improvement of a SoTA ATP on TPTP!

Future work: ready for experiments with richer feature sets!

Training with the RL-inspired Operator



Observations:

- random initialization $\Rightarrow$ different start performance
- hidden layer size 16, 32, 64, 128, 256 all similar performance

A first (modest) ML-based improvement of a SoTA ATP on TPTP!

Future work: ready for experiments with richer feature sets!

Thank you!