Neural branching heuristics for SAT solving

Sebastian Jaszczur, Michał Łuszczyk, Henryk Michalewski
University of Warsaw

AITP 2019, bit.ly/neurheur-aitp2019

Overview

1. Introduction
2. The neural network
3. Experiments
4. Conclusions

1. Introduction. DPLL algorithm

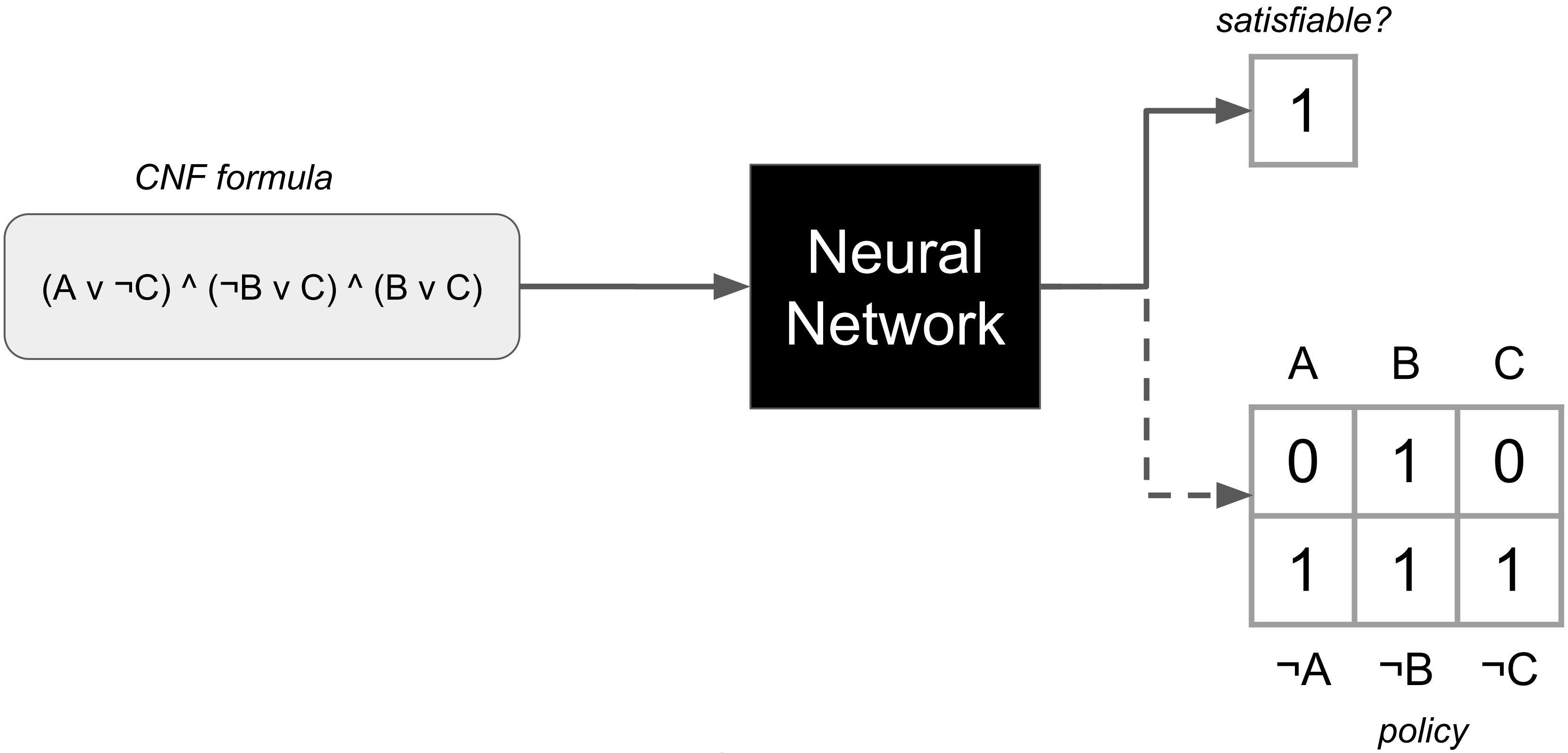
DPLL – basic backtracking algorithm for SAT solving. For illustration purposes.

function DPLL(Φ)
    $\Phi \leftarrow \texttt{simplify}(\Phi)$
    if Φ is trivially satisfiable **then return** True
    if Φ is trivially unsatisfiable **then return** False
    $literal \leftarrow \boxed{\texttt{choose-literal}}(\Phi)$
    if DPLL($\Phi \wedge literal$) **then return** True
    if DPLL($\Phi \wedge \neg literal$) **then return** True
    return False

Our work applies to CDCL, too!

2. Models

Neural network interface



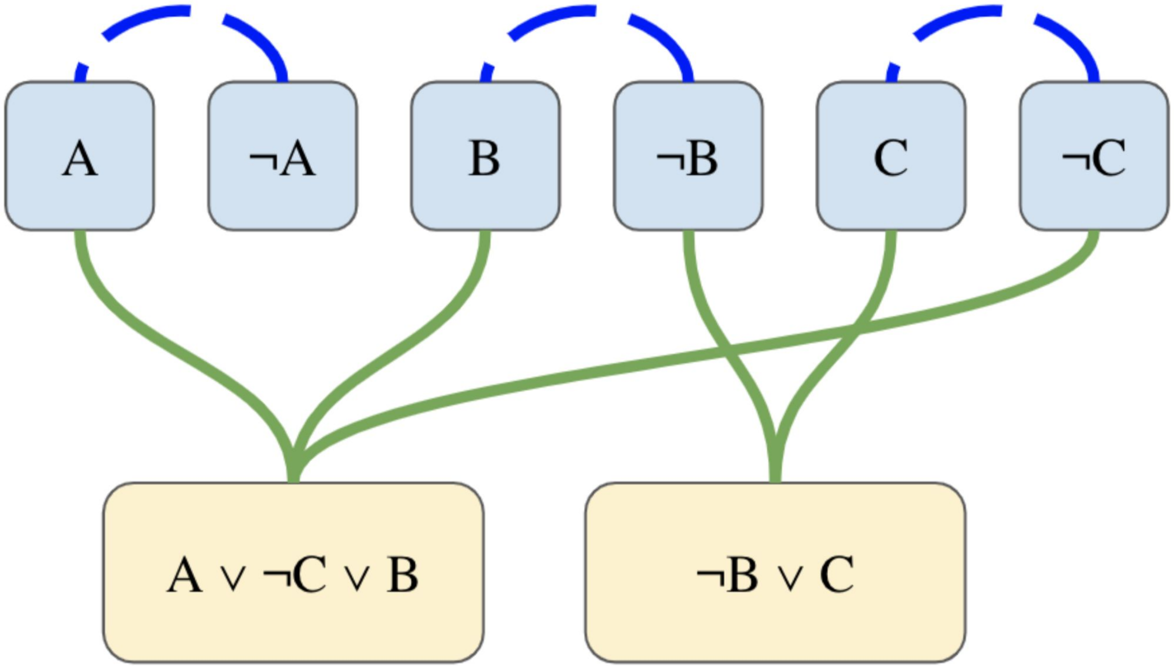
bit.ly/neurheur-aitp2019

CNF invariants

Invariant	TreeNN, LSTM	BOW averaging	Graph NN
“Variable renaming” - invariance	No	No	Yes
“Permutation of literals in clause” - invariance	No	Yes	Yes
“Permutation of clauses in formula” - invariance	No	Yes	Yes
“Negation of all occurrences of variable” - invariance	No	No	Yes

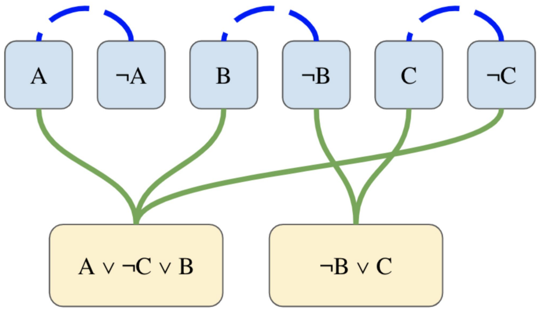
bit.ly/neurheur-aitp2019

CNF formula: graph representation



We can erase the labels and have an equisatisfiable problem!

bit.ly/neurheur-aitp2019

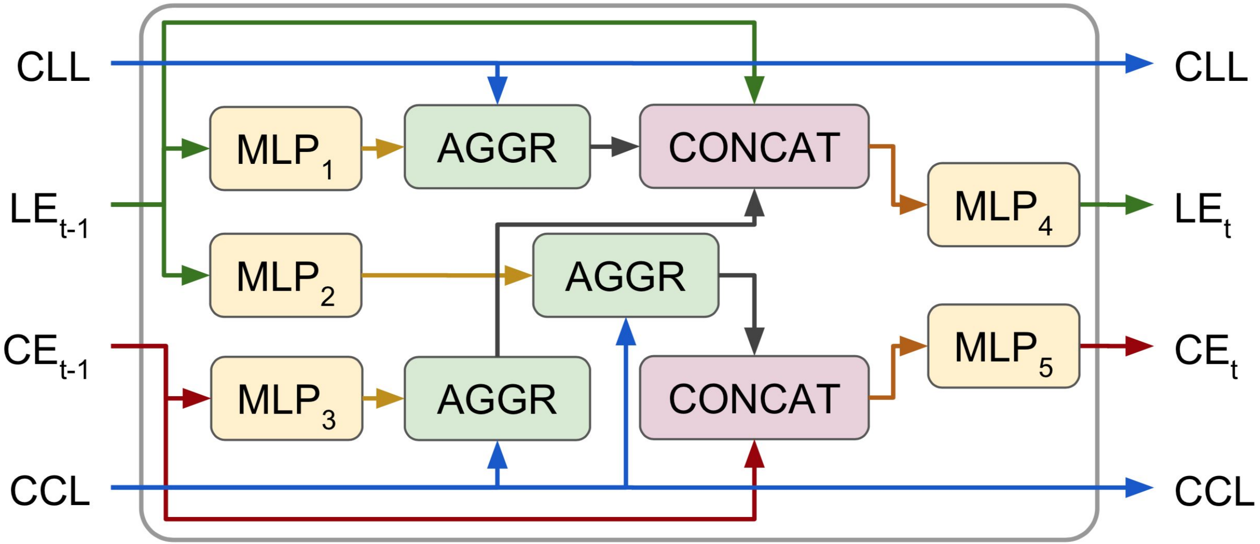


Graph neural network for CNF clauses

Inspired by NeuroSAT

Each vertex stores an embedding. In each iteration, each vertex:

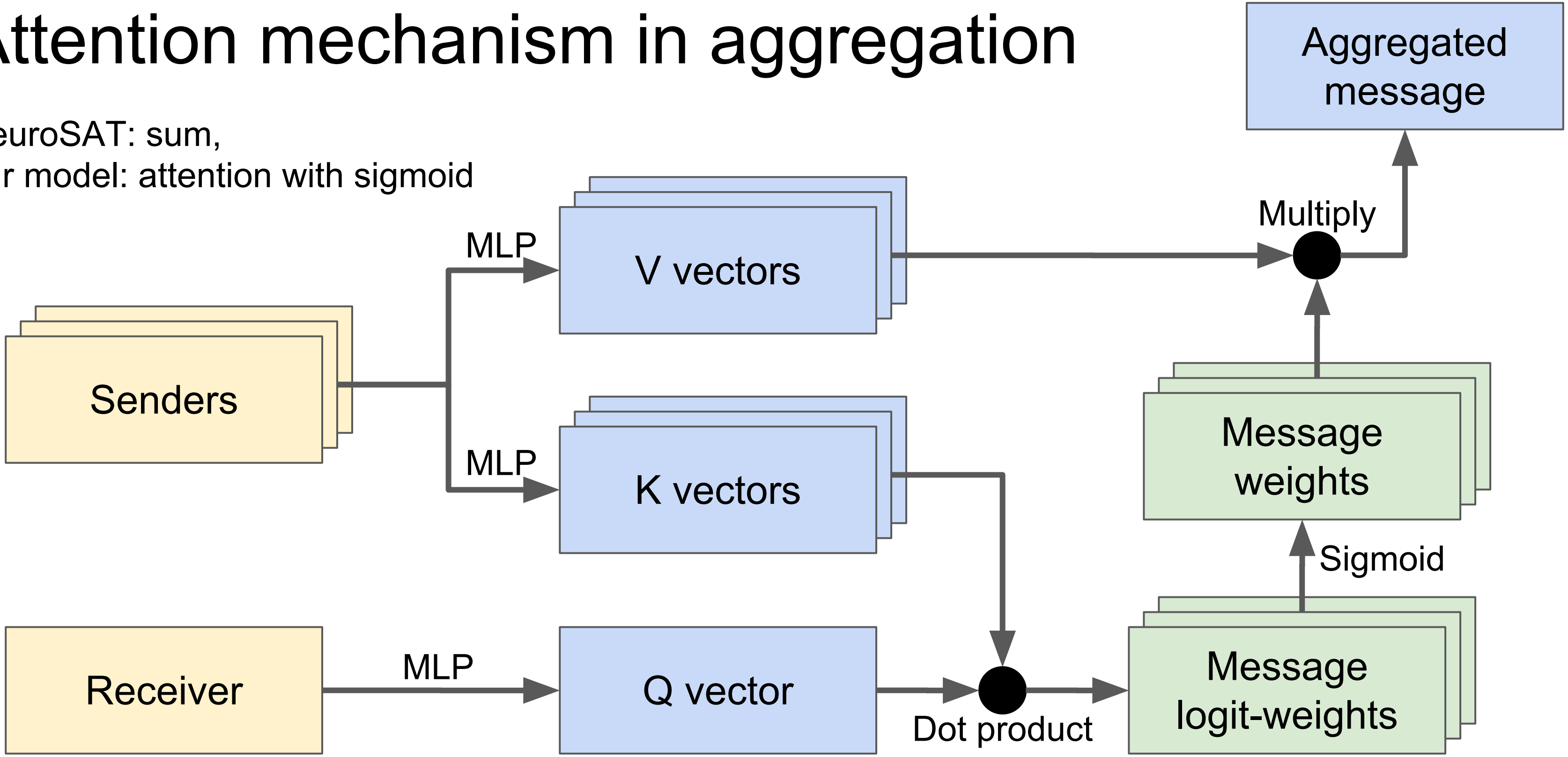
1. calculates a message and sends to neighbours,
2. receives the messages and aggregates,
3. calculates its new embedding based on aggregate and previous embedding.



bit.ly/neurheur-aitp2019

Attention mechanism in aggregation

NeuroSAT: sum,
our model: attention with sigmoid



Dataset *SR*(*x*)

- *x* - number of variables
- CNF distribution introduced in NeuroSAT
- formulas difficult for SAT solvers
- labels: generated with Glucose

Problem difficulty:

	Average number of clauses	Average formula size	Average time for MiniSAT
SR(30):	300±33	1480±175	0.007
SR(110)	1060±50	5100±287	0.137
SR(150)	1450±60	6930±320	3.406

bit.ly/neurheur-aitp2019

2. Trained models.

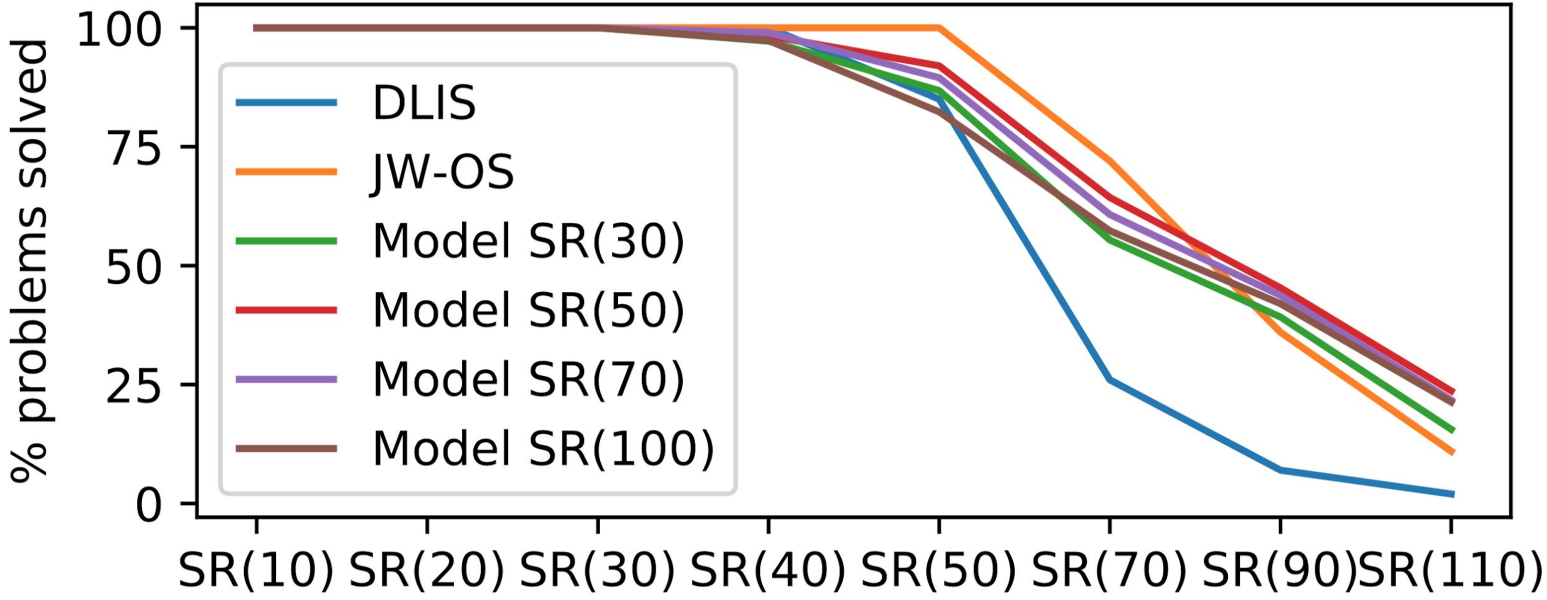
We trained 3-5 models on each of the following datasets.

Problem	Loss	*sat* error	*policy* error	Batch size	Train. steps	Train. time
SR(30)	28.178±0.672	0.084±0.004	0.050±0.002	128	1200K	20h
SR(50)	32.024±0.555	0.233±0.017	0.105±0.006	64	600K	12h
SR(70)	33.010±0.482	0.266±0.033	0.110±0.007	64	600K	22h
SR(100)	34.227±0.127	0.319±0.007	0.123±0.002	32	1200K	28h

Mean and stdev over 3-5 trained models.

3. Experiments

Performance with DPLL compared to *static* heuristics; at step limit 1000

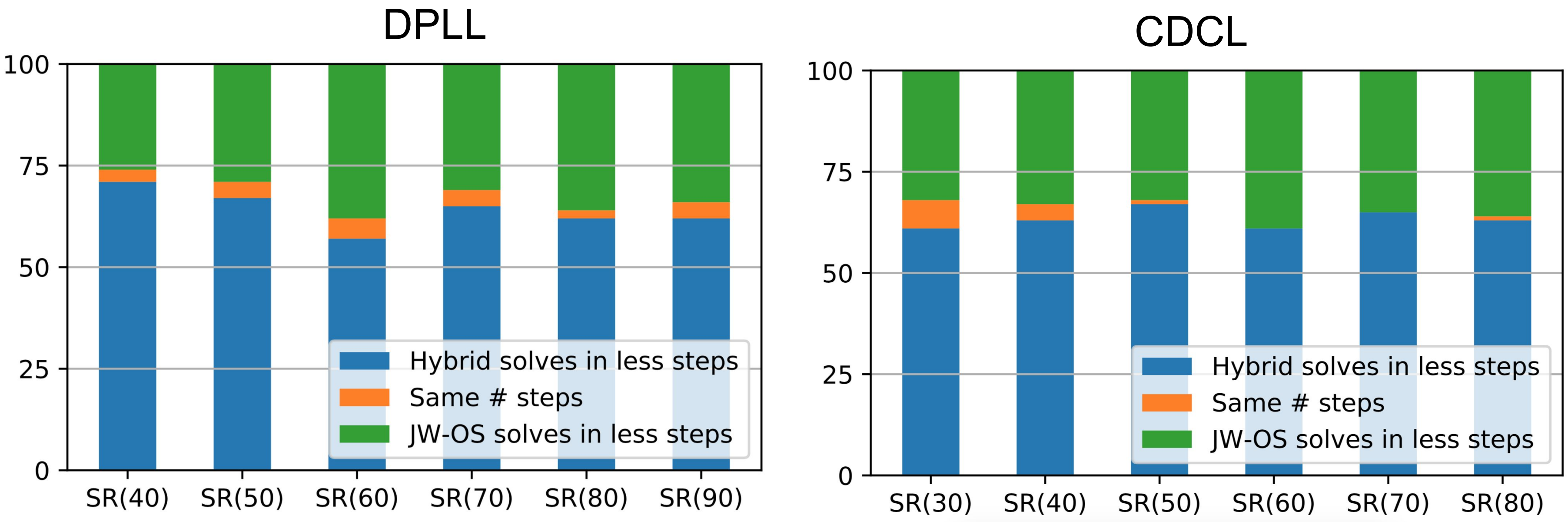


Learned heuristic better than DLIS. JW-OS better at SR(50)-SR(70) and worse at SR(90)-SR(110).

bit.ly/neurheur-aitp2019

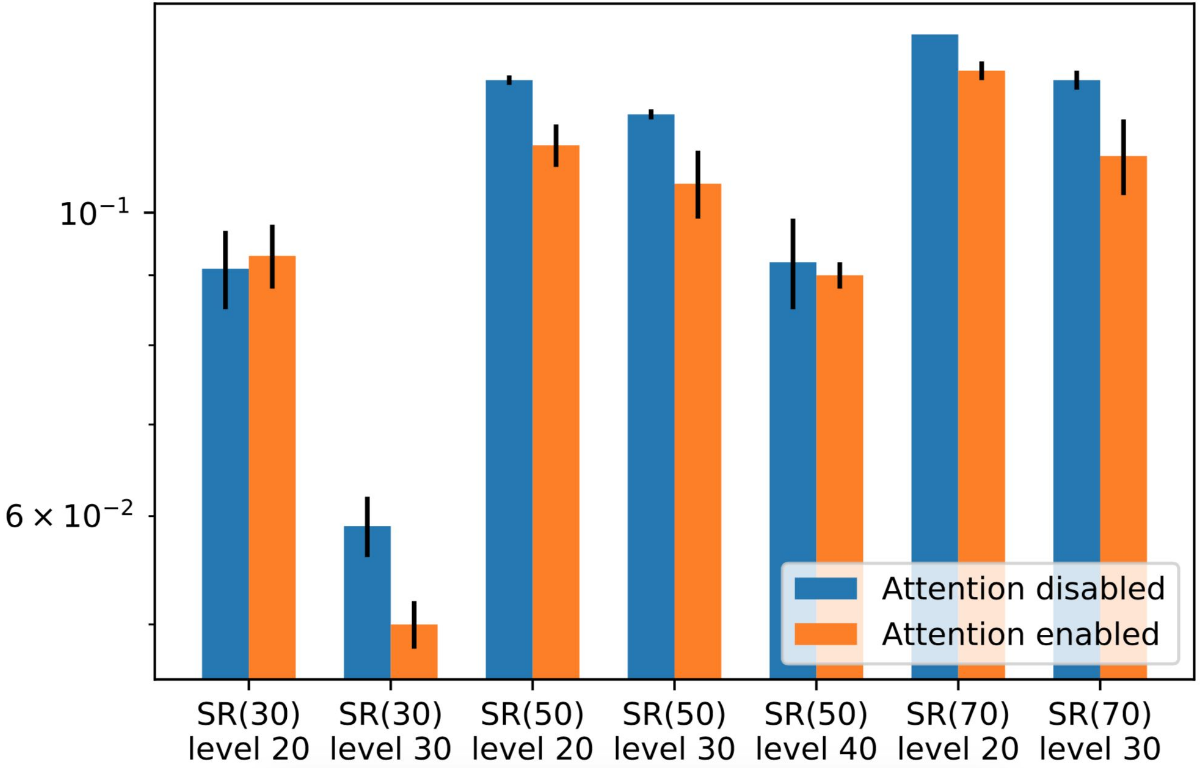
Hybrid vs JW-OS guiding DPLL and CDCL

Hybrid - use Model SR(50) if *sat probability* > 0.3, otherwise fallbacks to JW-OS.
Why? 1. Faster 2. NN *policy* isn't trained on unsatisfiable formulas.



bit.ly/neurheur-aitp2019

Attention experiment



y axis: policy error, average and std over 3-5 runs

- Attention significantly better,
- except for SR(30) l20 and SR(50) l40.

bit.ly/neurheur-aitp2019

Our contributions

- Learned branching heuristic using a GNN
- Modifying the NeuroSAT-inspired network with attention

Future work

- Optimise for the number of steps.
- Curriculum learning.
- Integrate with restart policy and clause learning and forgetting decision.
- Compare to VSIDS and other dynamic heuristics.
- Unsat certificates.

Our workshop paper: bit.ly/neurheur-iclr2019

```
(e|~f|d|a|~b|~j)&(~e|g|b)&(f|j|g)&(e|~f|j|~i)&(~j|f|~d)&(~c|~f|~h)&(~a|~j|~d)&(~j|h|~b|
~f)&(~h|~c|i|~b|j|e)&(~c|~e|a|b|i)&(~f|b|g|~i)&(~a|i|h|f|c)&(~d|f|g|~c|a)&(a|d|~c|~h)&(
c|~a|~h|d)&(f|~i|~a)&(d|b|f|~g|a)&(e|d|c|~f|~j)&(~f|d|~a)&(~c|~g|~d|~f|j)&(f|e|j)&(d|c|
~a|f|~j|e)&(i|~c|d|~j|h|b|a|~g|~f)&(j|i|~d)&(e|f|d)&(f|g|~a)&(~b|h|d)&(j|f|~b|g)&(~e|~g|
~h)&(e|h|~c|~b|d|a|g)&(~e|~j|~i|d|f)&(h|e|~g|d)&(~f|h|c|g|~j|b|d|~i|~a)&(~d|~j|~h|i|~c
)&(~a|~c|~d)&(d|~a|f)&(a|h|d)&(~b|~g|~f|~i|a|d)&(~a|~f|~g|j)&(e|~b|j|f)&(i|~g|~f|~e)&
(a|b|~h|~j|~f|~i)&(a|j|c)&(g|j|~c|~f)&(~e|d|~f|~c)&(j|~c|f)&(e|i|~f|j|b|g)&(~d|a|c|e)&(~b
|~a|~h)&(a|~j|h|e|i|c)&(~d|~a|g|h|~c|~f|j)&(~h|~e|~d)&(h|~j|~b|~f|~e|~g|d)&(~i|~j|a|d
)&(~i|~h|~f|~b|c)&(~f|e|a)&(j|b|~i)&(~c|j|~i|a|~h|e|d)&(~i|~h|c|~e|j|~d|~g)&(b|~c|j|i)&(
e|~d|~a|~g|~h)&(c|~b|i|~h)&(~j|h|c)&(~c|i|g)&(f|a|~i)&(~e|h|~j|~c|~d)&(a|b|~f)&(j|~h|
i|d)&(~d|a|e)&(h|~j|d)&(d|~f|~h|~e|~b)&(~b|~f|j|~c|h|~i)&(~b|f|~j|d|~g)&(~a|i|~b|~c)&
(d|e|~c|j|~h)&(~b|~d|~i|~j|~a)&(f|j|i|h|~g|d|~a|~c)&(g|f|~e)&(~e|~h|d|j)&(i|j|~a|g|~e|~
h)&(~e|~g|j)&(c|~h|e|~j|~d)&(i|~f|~j)&(b|~h|~g|j|f|i|e|~d)&(~j|e|~i|~a|d|~g)
```

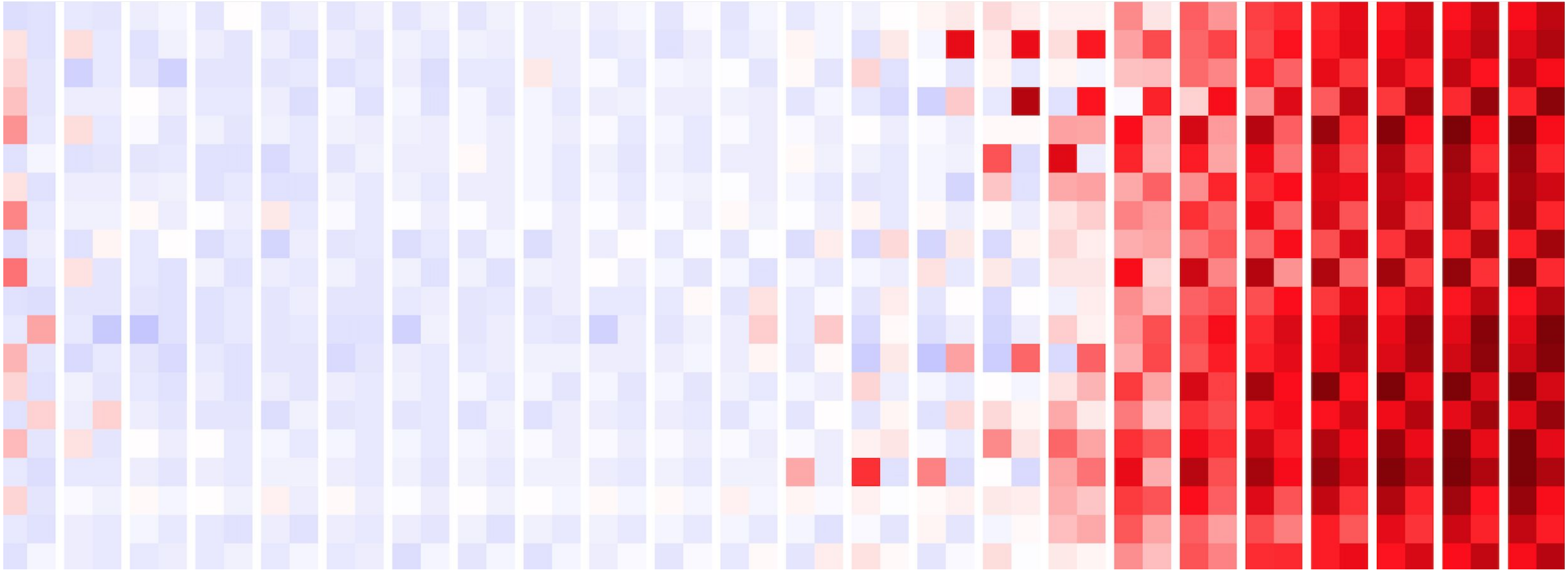
Why sigmoidal attention?

- Attention with sigmoid resembles aggregation with sum while attention with softmax resembles aggregation with average,
- average loses important information e.g. it cannot count the neighbours.

Why we compare steps rather than time

- Time optimisations are possible (parallel execution, simpler model, non-Python implementation), it’s engineering work,
- bigger models determines the upper bound,
- we expect better time at sufficiently large instances:
    - actually, we’re better than some heuristics now.

How NeuroSAT works *x* axis - iteration number, *sat* probability at each literal node



bit.ly/neurheur-aitp2019

Usage

- We take a formula, and we predict SAT probability and policy probability
- SAT probability, computed for whole formula:
  - Ground truth is "is whole formula satisfiable?" - like NeuroSAT
  - We do linear regression on every node's embedding. We output sigmoid of sum of the results of those linear regression.
- Policy probability, computed separately for each literal:
  - Ground truth is "is there a solution to this formula with this literal?".
  - We do logistic regression on every literal node's embedding.