deepsense.ai
BIG DATA SCIENCE

UNIVERSITY OF WARSAW

UNIVERSITY OF OXFORD

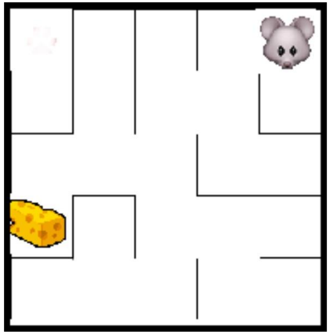
Machine Learning and Logic

Henryk Michalewski
Google Brain, Mountain View, 28th of November 2018

what is reinforcement learning?

"... neuroscience was ... instrumental in erecting a **second pillar of contemporary AI**, stimulating the emergence of the field of **reinforcement learning** (RL). RL methods address the problem of *how to maximize future reward by mapping states in the environment to actions* and are among the most widely used tools in AI research. ..., RL methods were originally inspired by research into **animal learning**."

- Neuroscience-Inspired Artificial Intelligence, Demis Hassabis, Dharshan Kumaran, Christopher Summerfield and Matthew Botvinick, Neuron 2017



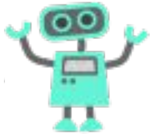
RL crash course

maximize future reward of the agent by mapping states in the ***environment*** to actions

RL crash course

maximize future reward of the ***agent*** by mapping states in the *environment* to actions

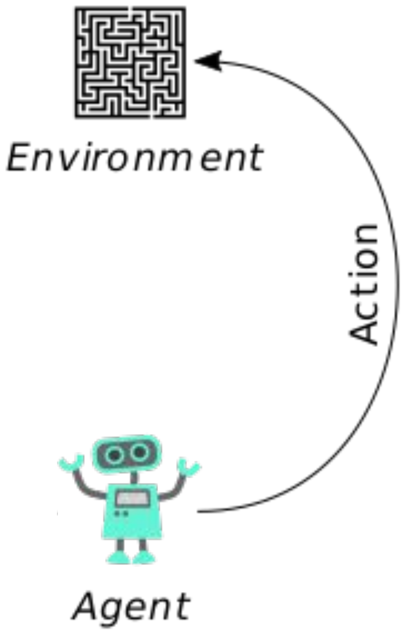
Environment



Agent

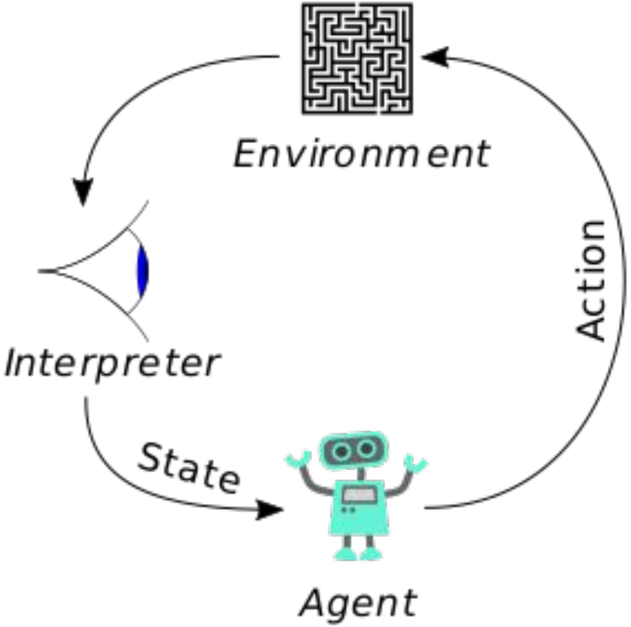
RL crash course

maximize future reward of the *agent* by mapping states in the *environment* to ***actions***



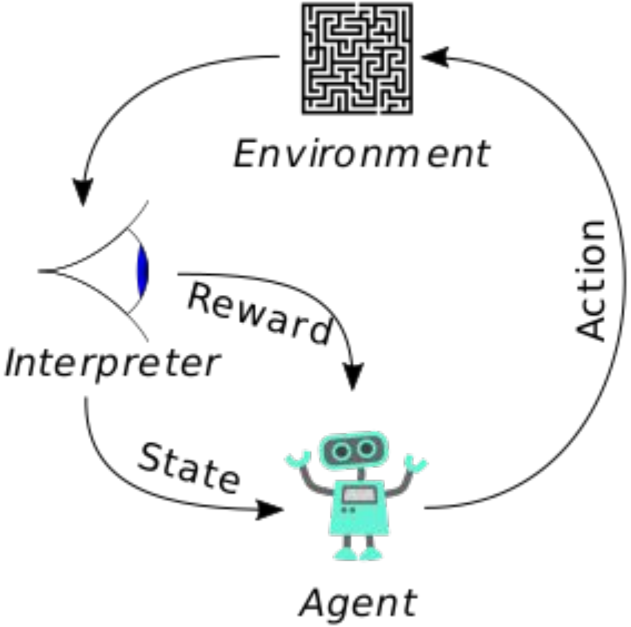
RL crash course

maximize future reward of the *agent* by mapping ***states*** in the *environment* to *actions*



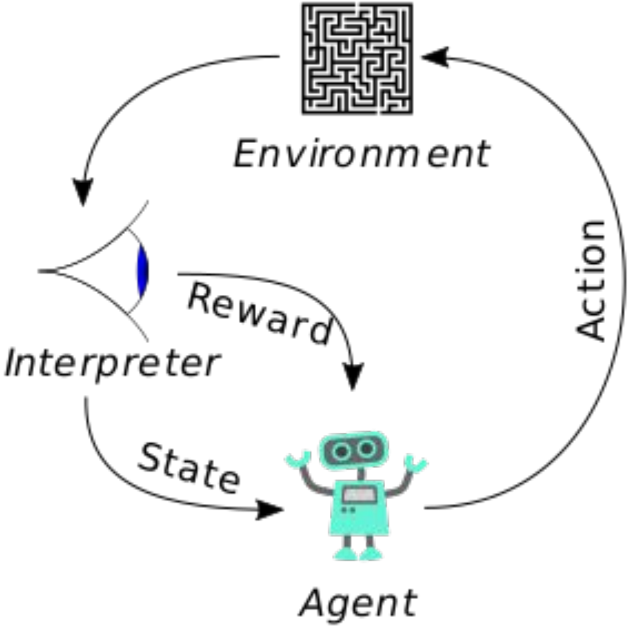
RL crash course

maximize future ***reward*** of the *agent* by mapping ***states*** in the *environment* to *actions*



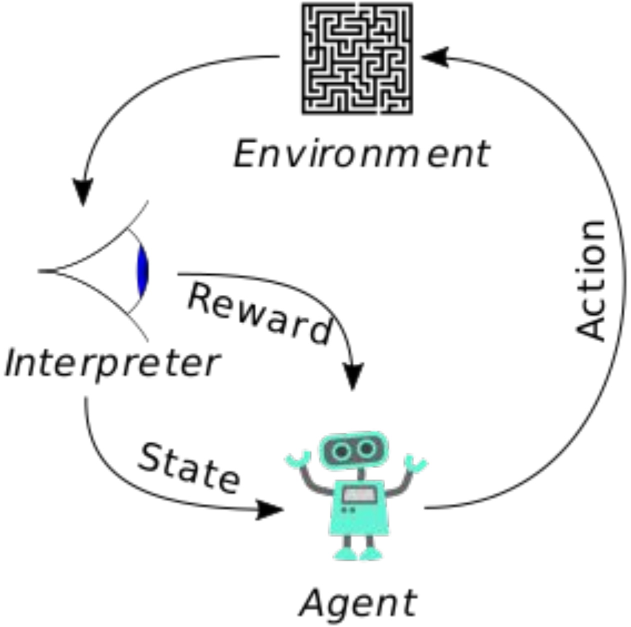
RL crash course

maximize future ***reward*** of the *agent* by mapping *states* in the *environment* to *actions*



RL crash course - some math ;)

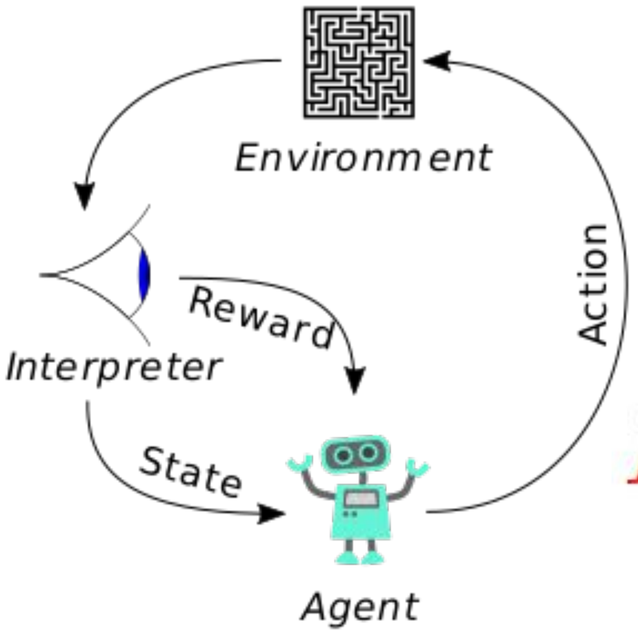
maximize future ***reward*** of the *agent* by mapping *states* in the *environment* to *actions*



$$S \text{ – state space}$$
$$A \text{ – action space}$$
$$\pi : S \mapsto A \text{ – policy}$$

RL crash course - some math ;)

maximize future ***reward*** of the *agent* by mapping *states* in the *environment* to *actions*



$$S - \text{state space}$$

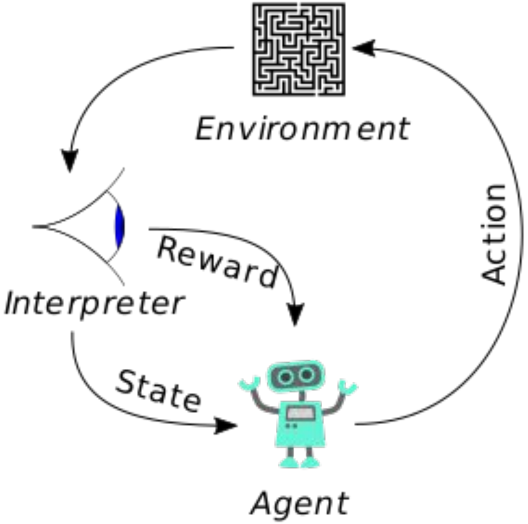
$$A - \text{action space}$$

$$\pi : S \mapsto A - \text{policy}$$

$$R_\pi = \mathbb{E}_\pi (r_1 + r_2 + r_3 + \ldots)$$

RL crash course - some math ;)

maximize future ***reward*** of the *agent* by mapping *states* in the *environment* to *actions*



$$S \text{ – state space}$$

$$A \text{ – action space}$$

$$\pi : S \mapsto A \text{ – policy}$$

$$R_\pi = \mathbb{E}_\pi (r_1 + r_2 + r_3 + \ldots)$$

$$\textbf{Goal: } \text{find } \pi_0 \text{ such that } R_{\pi_0} \text{ is maximal}$$

RL - part for adults

Goal: find π_0 such that R_{π_0} is maximal

$$R_\pi = \mathbb{E}_\pi \left(r_1 + r_2 + r_3 + \ldots \right)$$

RL - part for adults

Goal: find π_0 such that R_{π_0} is maximal

$$R_\pi = \mathbb{E}_\pi \left(r_1 + r_2 + r_3 + \ldots \right)$$

$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau)]$$

RL - part for adults

Goal: find π_0 such that R_{π_0} is maximal

$$R_\pi = \mathbb{E}_\pi \left(r_1 + r_2 + r_3 + \ldots \right)$$

$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau)]$$

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta} [\nabla_\theta \log(\pi_\theta(\tau)) \; r(\tau)].$$

RL - part for adults

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log(\pi_\theta(\tau))\, r(\tau)]$$

$$\hat{g} = \nabla_\theta \mathbb{E}_{(s_t,a_t)\sim\pi_0}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_0(a_t|s_t)}\widehat{A}_{\pi_0}(s_t,a_t)\right] = \mathbb{E}_{(s_t,a_t)\sim\pi_0}\left[\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\pi_0(a_t|s_t)}\widehat{A}_{\pi_0}(s_t,a_t)\right]$$

$$\begin{aligned}&\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1},\\ &\text{where} \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)\end{aligned}$$

RL - part for adults

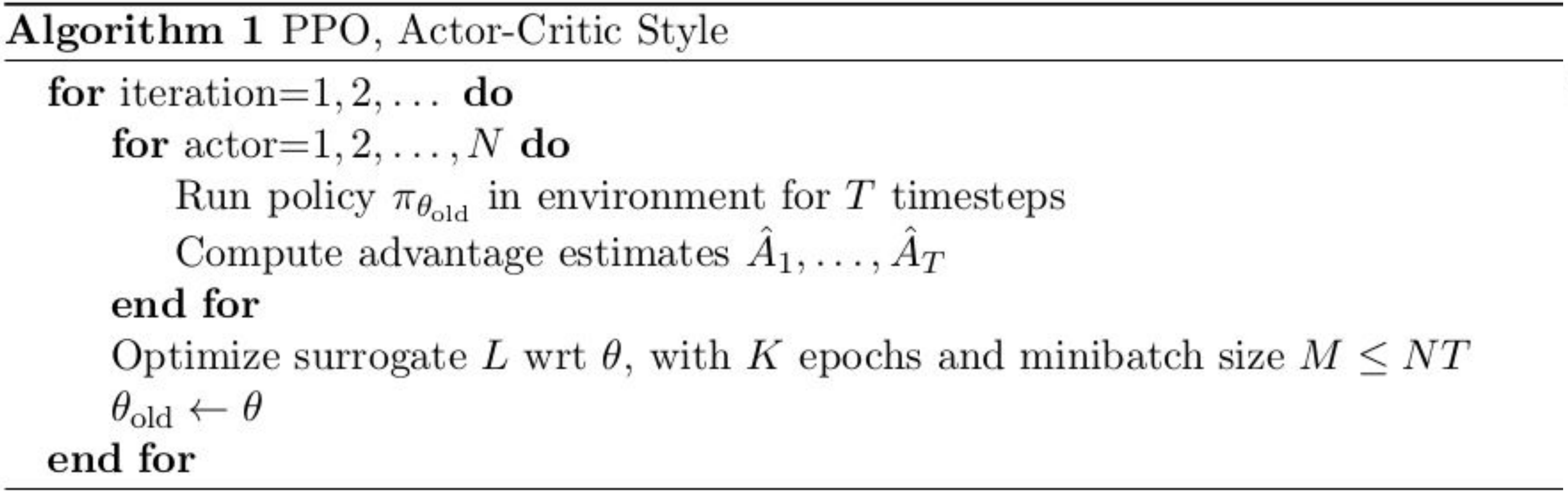
An innovation from the PPO paper:

$$\max_{\theta}\ \mathbb{E}_{(s_t,a_t)\sim\pi}\left[\min\left(\operatorname{clip}\left(\rho_t, 1-\varepsilon, 1+\varepsilon\right)\widehat{A}_{\pi}(s_t,a_t),\ \rho_t\widehat{A}_{\pi}(s_t,a_t)\right)\right]$$

where

$$\rho_t = \frac{\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)}$$

RL - part for adults

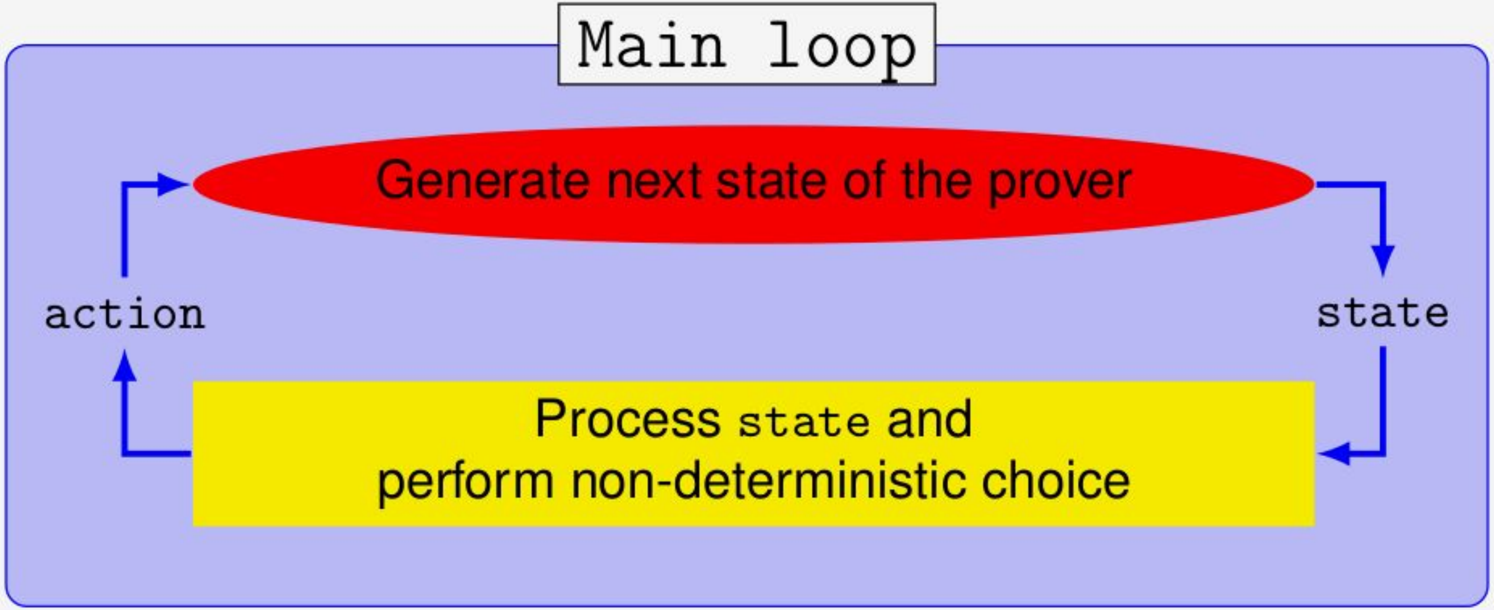


Algorithm 1 PPO, Actor-Critic Style

```
for iteration=1, 2, ... do
    for actor=1, 2, ..., $N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

Curriculum learning and theorem proving

(joint work with Adrián Csiszárik, Cezary Kaliszyk, Josef Urban, Zsolt Zombori)



Main loop

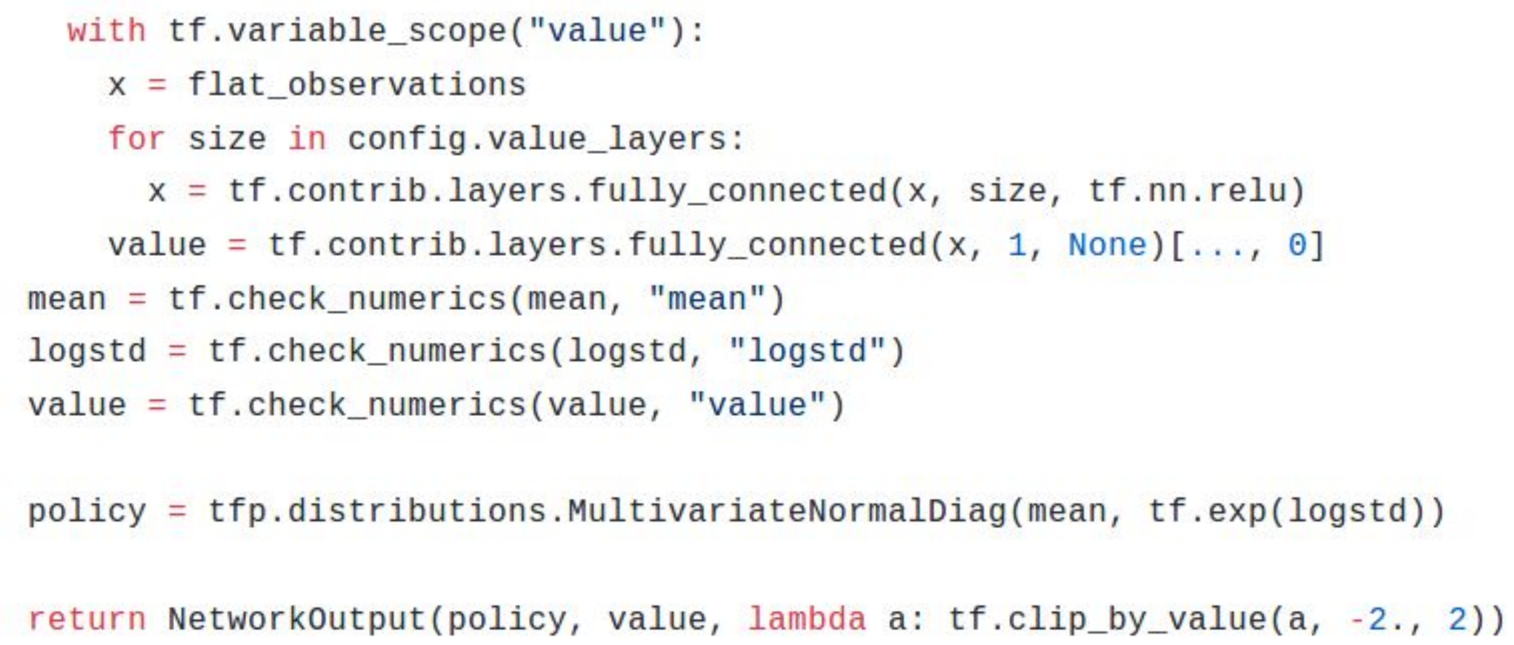
Generate next state of the prover

state

Process `state` and
perform non-deterministic choice

action

Figure 1: The general prover loop.



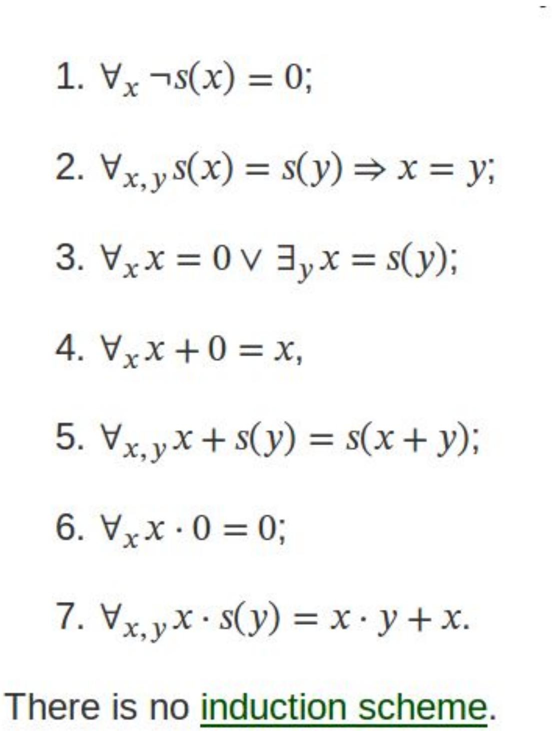
```
  with tf.variable_scope("value"):
    x = flat_observations
    for size in config.value_layers:
      x = tf.contrib.layers.fully_connected(x, size, tf.nn.relu)
    value = tf.contrib.layers.fully_connected(x, 1, None)[..., 0]
mean = tf.check_numerics(mean, "mean")
logstd = tf.check_numerics(logstd, "logstd")
value = tf.check_numerics(value, "value")

policy = tfp.distributions.MultivariateNormalDiag(mean, tf.exp(logstd))

return NetworkOutput(policy, value, lambda a: tf.clip_by_value(a, -2., 2))
```

https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/models/research/rl.py

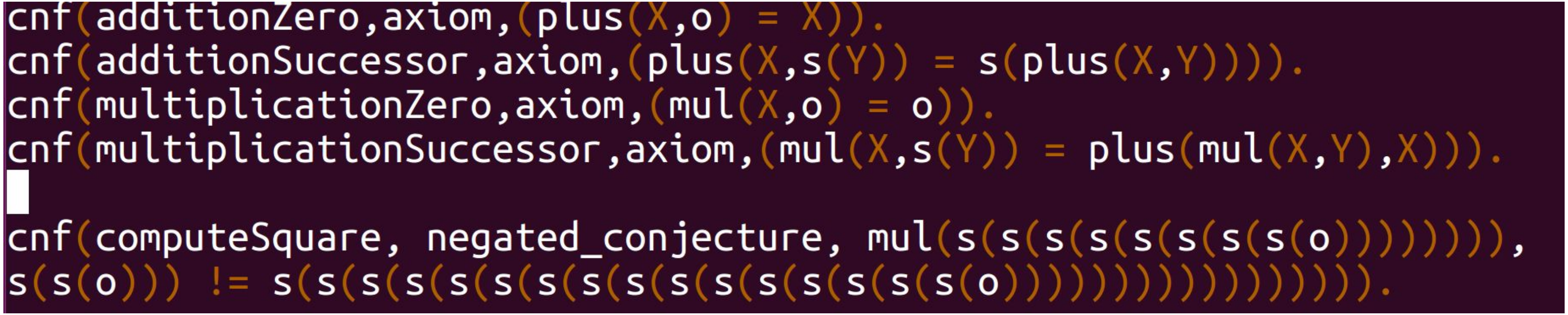
How to translate theorem proving to games?



1. $\forall_x \neg s(x) = 0;$
2. $\forall_{x,y} s(x) = s(y) \Rightarrow x = y;$
3. $\forall_x x = 0 \vee \exists_y x = s(y);$
4. $\forall_x x + 0 = x,$
5. $\forall_{x,y} x + s(y) = s(x + y);$
6. $\forall_x x \cdot 0 = 0;$
7. $\forall_{x,y} x \cdot s(y) = x \cdot y + x.$

There is no induction scheme.

How to translate theorem proving to games?



```
cnf(additionZero,axiom,(plus(X,o) = X)).
cnf(additionSuccessor,axiom,(plus(X,s(Y)) = s(plus(X,Y)))).
cnf(multiplicationZero,axiom,(mul(X,o) = o)).
cnf(multiplicationSuccessor,axiom,(mul(X,s(Y)) = plus(mul(X,Y),X))).

cnf(computeSquare, negated_conjecture, mul(s(s(s(s(s(s(s(s(o)))))))),
s(s(o))) != s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(o))))))))))))))))).
```

Approach

The RL workshop at NIPS 2018 (joint work with Michał Garmulewicz and Piotr Miłoś)

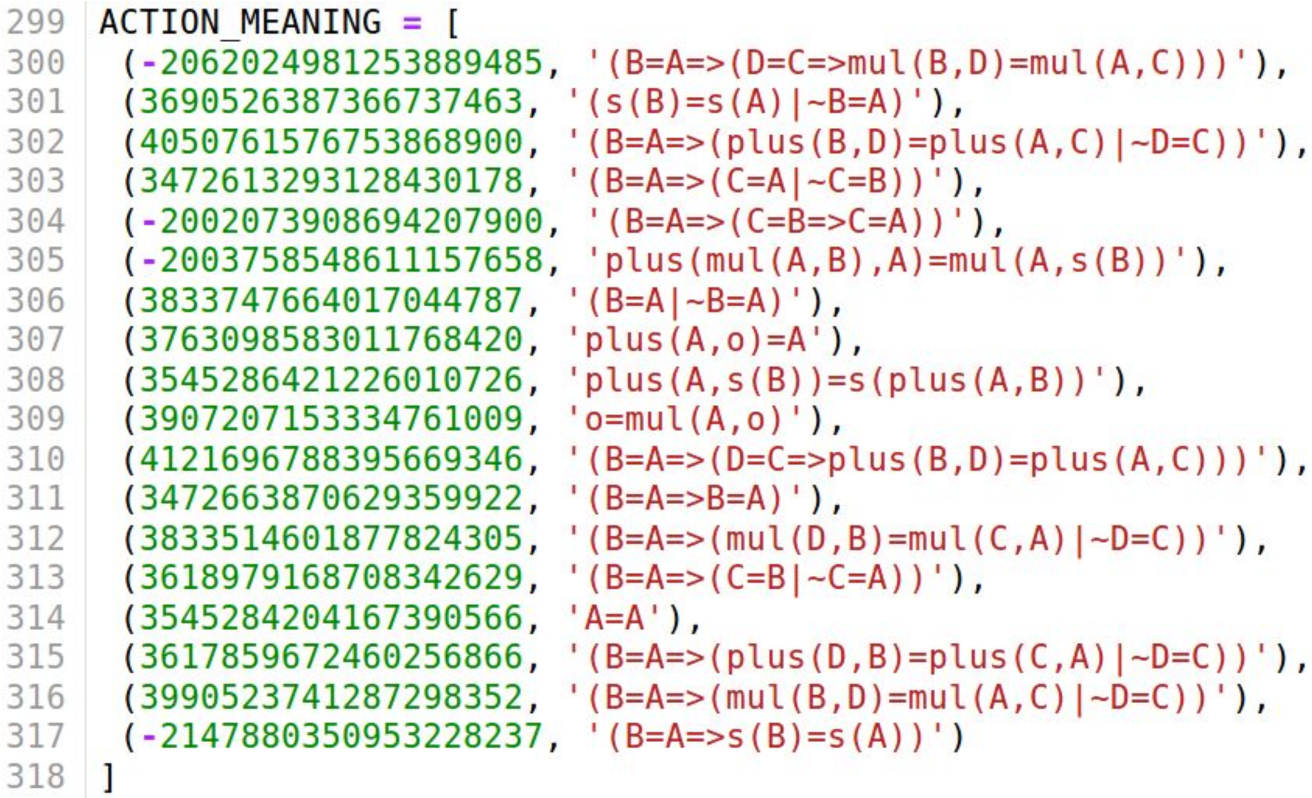
Use curriculum learning to counterbalance sparse rewards

- Take a few given proof trajectories
- Start exploration from the end of the trajectory
- Gradually move backwards towards the beginning of the proof

Handling the the discrete action space (**two options**):

- Use fixed action space: restricted to a narrow type of problems. **A quick route to strong results, however in general our action space is infinite discrete.**
- Action selector model: policy receives all available actions and returns probabilities. **Can generalize to yet unseen actions.**

How to translate theorem proving to games?



```
ACTION_MEANING = [
 (-2062024981253889485, '(B=A=>(D=C=>mul(B,D)=mul(A,C)))'),
 (3690526387366737463, '(s(B)=s(A)|~B=A)'),
 (4050761576753868900, '(B=A=>(plus(B,D)=plus(A,C)|~D=C))'),
 (3472613293128430178, '(B=A=>(C=A|~C=B))'),
 (-2002073908694207900, '(B=A=>(C=B=>C=A))'),
 (-2003758548611157658, 'plus(mul(A,B),A)=mul(A,s(B))'),
 (3833747664017044787, '(B=A|~B=A)'),
 (3763098583011768420, 'plus(A,o)=A'),
 (3545286421226010726, 'plus(A,s(B))=s(plus(A,B))'),
 (3907207153334761009, 'o=mul(A,o)'),
 (4121696788395669346, '(B=A=>(D=C=>plus(B,D)=plus(A,C)))'),
 (3472663870629359922, '(B=A=>B=A)'),
 (3833514601877824305, '(B=A=>(mul(D,B)=mul(C,A)|~D=C))'),
 (3618979168708342629, '(B=A=>(C=B|~C=A))'),
 (3545284204167390566, 'A=A'),
 (3617859672460256866, '(B=A=>(plus(D,B)=plus(C,A)|~D=C))'),
 (3990523741287298352, '(B=A=>(mul(B,D)=mul(A,C)|~D=C))'),
 (-2147880350953228237, '(B=A=>s(B)=s(A))')
]
```

Benefits of action selector model

- Returns a probability distribution (vs scoring function)
- Cannot generalize to yet unseen actions (vs fixed action space model)
- Can handle large action spaces (provided they are not all available together)
- Can make decisions based on other available actions
- Exploits the features of actions
    - Actions and spaces are embedded into the same feature space (manual features, widely used in the ATP community)
    - The network can meaningfully process the concatenation of state-action feature vectors.

Results

Trained and tested on Robinson arithmetic problems

RL method - variations of the Actor-Critic meta-algorithm:

- Fixed action space model trained on 7*2=14 (51 steps) managed to prove all multiplications up to 9*9 (1201 steps)
- Action selector model trained on 7+5=12 (17 steps) managed to prove all additions up to 9+9

What is curriculum learning in the context of reinforcement learning?

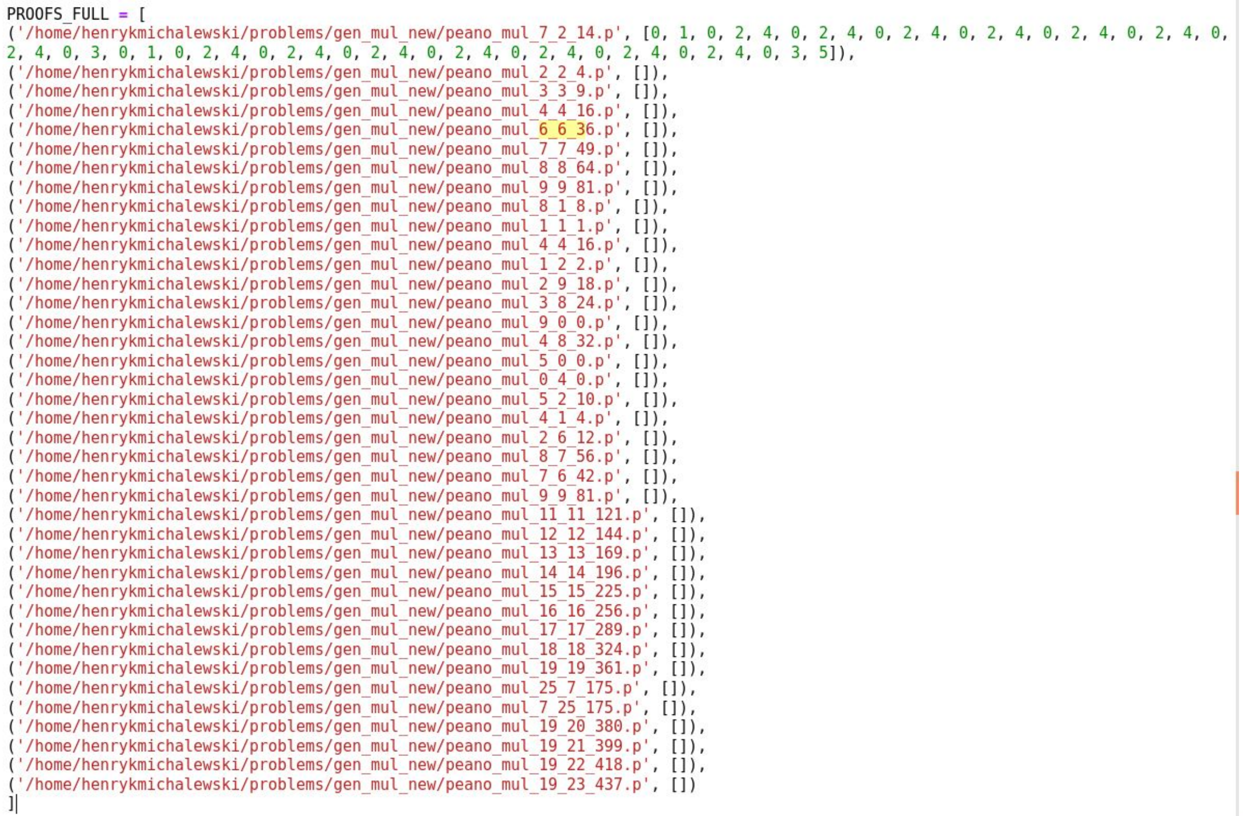
We have a proof trajectory:

```
('/home/henrykmichalewski/problems/gen_mul_new/pe
ano_mul_7_2_14.p', [0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2,
4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2,
4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 5]),
```

Curriculum learning recipe:

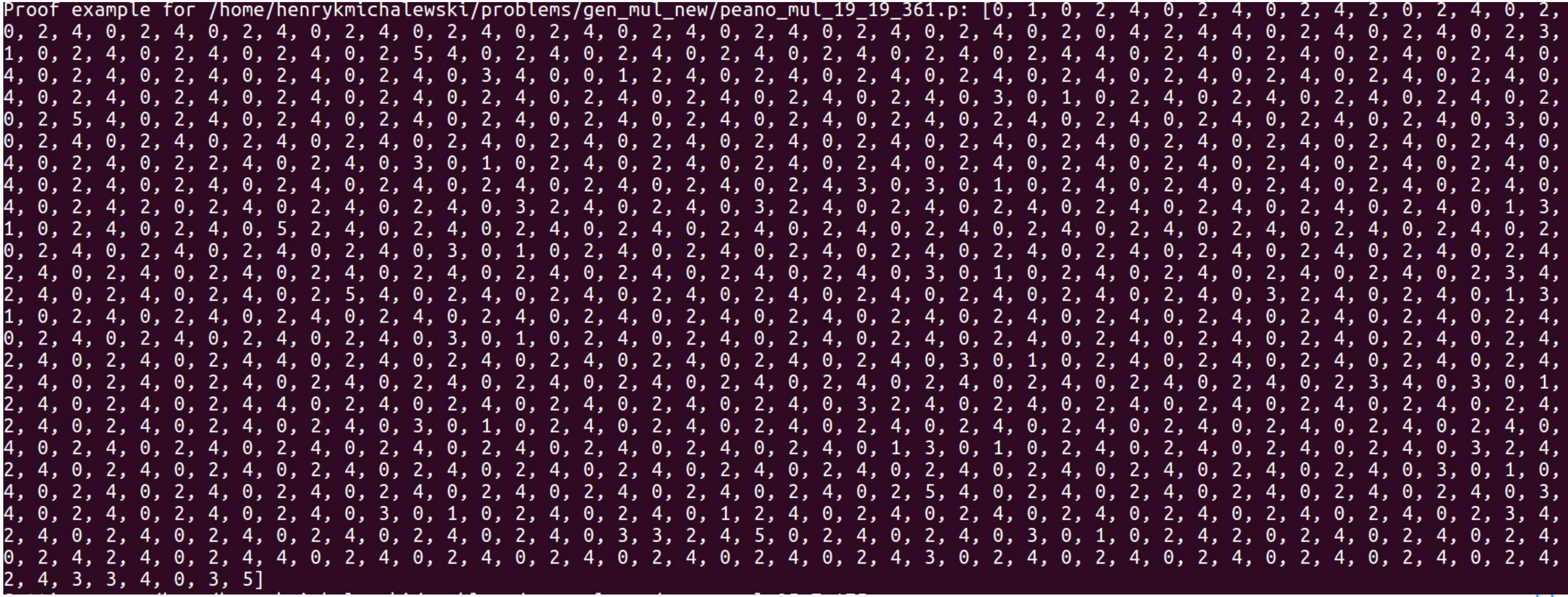
1. **Start close to the end of trajectory**
2. **Gradually raise the bar**
3. **Check whether it generalize to other problems**

Example of a curriculum:



```
PROOFS_FULL = [
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_7_2_14.p', [0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0,
2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 5]),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_2_2_4.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_3_3_9.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_4_4_16.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_6_6_36.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_7_7_49.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_8_8_64.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_9_9_81.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_8_1_8.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_1_1_1.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_4_4_16.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_1_2_2.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_2_9_18.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_3_8_24.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_9_0_0.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_4_8_32.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_5_0_0.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_0_4_0.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_5_2_10.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_4_1_4.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_2_6_12.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_8_7_56.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_7_6_42.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_9_9_81.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_11_11_121.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_12_12_144.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_13_13_169.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_14_14_196.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_15_15_225.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_16_16_256.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_17_17_289.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_18_18_324.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_19_19_361.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_25_7_175.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_7_25_175.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_19_20_380.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_19_21_399.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_19_22_418.p', []),
('/home/henrykmichalewski/problems/gen_mul_new/peano_mul_19_23_437.p', [])
]
```

From a proof of length 51 we are learning a proof of length 1201:



```
Proof example for /home/henrykmichalewski/problems/gen_mul_new/peano_mul_19_19_361.p: [0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 2, 0, 2, 4, 0, 2,
0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 0, 4, 2, 4, 4, 0, 2, 4, 0, 2, 4, 0, 2, 3,
1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 5, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0,
4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 4, 0, 0, 1, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0,
4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2,
0, 2, 5, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0,
0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0,
4, 0, 2, 4, 0, 2, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0,
4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 3, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0,
4, 0, 2, 4, 2, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 2, 4, 0, 2, 4, 0, 3, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 1, 3,
1, 0, 2, 4, 0, 2, 4, 0, 5, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2,
0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4,
2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 3, 4,
2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 5, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 2, 4, 0, 2, 4, 0, 1, 3,
1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4,
0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4,
2, 4, 0, 2, 4, 0, 2, 4, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4,
2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 3, 4, 0, 3, 0, 1,
2, 4, 0, 2, 4, 0, 2, 4, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4,
2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0,
4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 1, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 2, 4,
2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0,
4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 5, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3,
4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 0, 2, 4, 0, 1, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 3, 4,
2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 3, 3, 2, 4, 5, 0, 2, 4, 0, 2, 4, 0, 3, 0, 1, 0, 2, 4, 2, 0, 2, 4, 0, 2, 4, 0, 2, 4,
0, 2, 4, 2, 4, 0, 2, 4, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 3, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4, 0, 2, 4,
2, 4, 3, 3, 4, 0, 3, 5]
```

SQL queries and machine learning

Joint project with Michael Benedikt (work in progress)

Input: a query Q, a set of views/view definitions

Output: a query using only the views that is equivalent to Q

Input: a query `Q`, a set of views/view definitions

Output: a query using only the views that is equivalent to `Q`

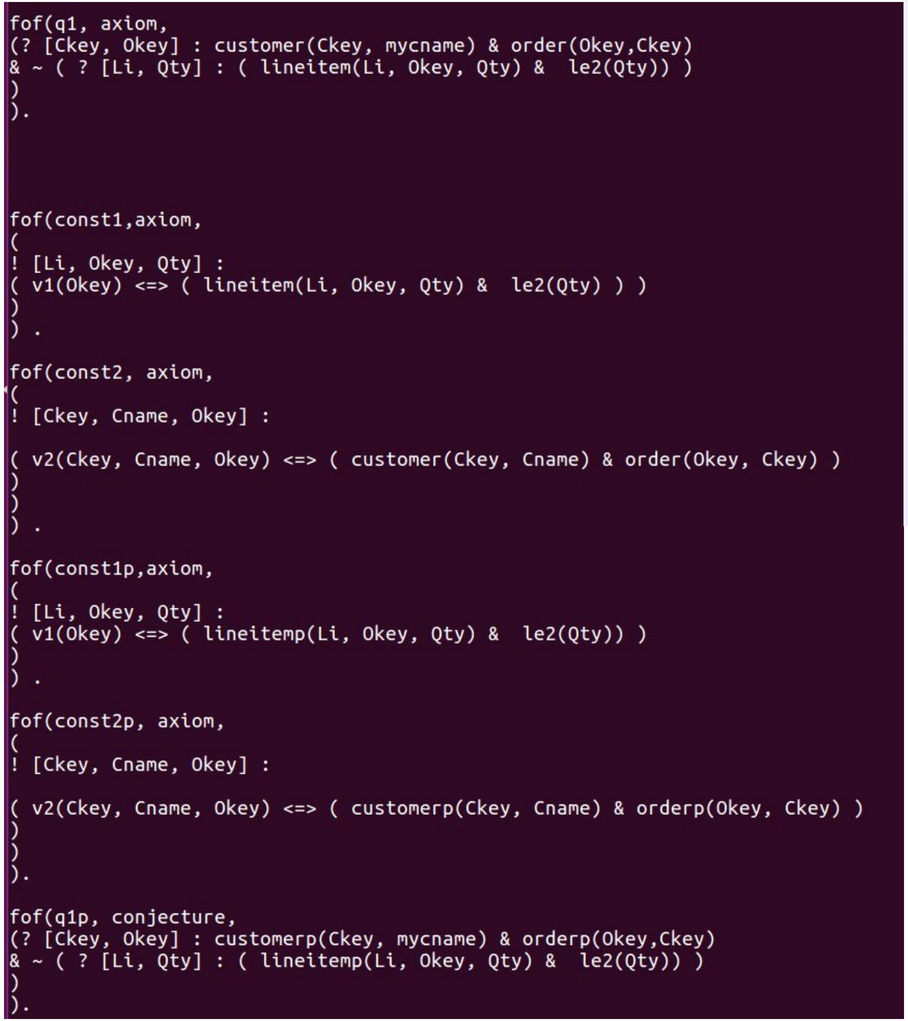
Query:

```
SELECT DISTINCT c.custname
FROM Customer c, Orders o
WHERE c.custkey =o.custkey
AND NOT EXISTS
(SELECT li.quantity FROM
LineItem li WHERE
li.quantitity<2
AND li.orderkey=o.orderkey)
```

Views:

```
V1 (orderkey,...) = SELECT * FROM
LineItem li WHERE li.quantity<2

V2 (custkey, custname, orderkey) =
SELECT * FROM Customers c, Orders o
WHERE c.custkey=o.custkey
```



```
fof(q1, axiom,
(? [Ckey, Okey] : customer(Ckey, mycname) & order(Okey,Ckey)
& ~ ( ? [Li, Qty] : ( lineitem(Li, Okey, Qty) &  le2(Qty)) )
)
).




fof(const1,axiom,
(
! [Li, Okey, Qty] :
( v1(Okey) <=> ( lineitem(Li, Okey, Qty) &  le2(Qty) ) )
)
) .

fof(const2, axiom,
(
! [Ckey, Cname, Okey] :

( v2(Ckey, Cname, Okey) <=> ( customer(Ckey, Cname) & order(Okey, Ckey) )
)
)
) .

fof(const1p,axiom,
(
! [Li, Okey, Qty] :
( v1(Okey) <=> ( lineitemp(Li, Okey, Qty) &  le2(Qty)) )
)
) .

fof(const2p, axiom,
(
! [Ckey, Cname, Okey] :

( v2(Ckey, Cname, Okey) <=> ( customerp(Ckey, Cname) & orderp(Okey, Ckey) )
)
)
).

fof(q1p, conjecture,
(? [Ckey, Okey] : customerp(Ckey, mycname) & orderp(Okey,Ckey)
& ~ ( ? [Li, Qty] : ( lineitemp(Li, Okey, Qty) &  le2(Qty)) )
)
).
```

Query:

```
SELECT DISTINCT c.custname
FROM Customer c, Orders o
WHERE c.custkey =o.custkey
AND NOT EXISTS
 (SELECT li.quantity FROM
LineItem li WHERE
li.quantitity<2
AND li.orderkey=o.orderkey)
```

Views:

```
V1 (orderkey,...) = SELECT * FROM
LineItem li WHERE li.quantity<2

V2 (custkey, custname, orderkey) =
SELECT * FROM Customers c, Orders o
WHERE c.custkey=o.custkey
```

Query:

```
SELECT DISTINCT c.custname
FROM Customer c, Orders o
WHERE c.custkey =o.custkey
AND NOT EXISTS
(SELECT li.quantity FROM
LineItem li WHERE
li.quantitity<2
AND li.orderkey=o.orderkey)
```

Views:

```
V1 (orderkey,...) = SELECT * FROM
LineItem li WHERE li.quantity<2

V2 (custkey, custname, orderkey) =
SELECT * FROM Customers c, Orders o
WHERE c.custkey=o.custkey
```

Rewriting:

```
SELECT DISTINCT v2.custname FROM V2 v2
WHERE v2.orderkey NOT IN
(SELECT orderkey FROM V1)
```

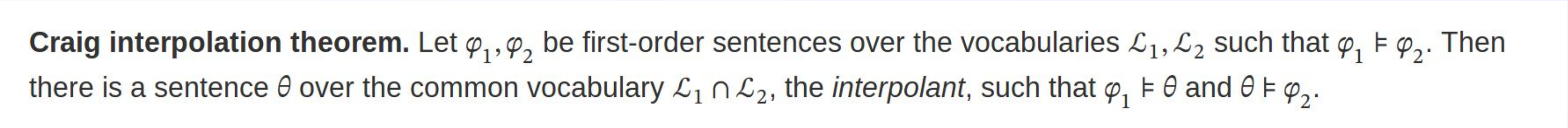
Query:

```
SELECT DISTINCT c.custname
FROM Customer c, Orders o
WHERE c.custkey =o.custkey
AND NOT EXISTS
(SELECT li.quantity FROM
LineItem li WHERE
li.quantitity<2
AND li.orderkey=o.orderkey)
```

Views:

```
V1 (orderkey,...) = SELECT * FROM
LineItem li WHERE li.quantity<2

V2 (custkey, custname, orderkey) =
SELECT * FROM Customers c, Orders o
WHERE c.custkey=o.custkey
```

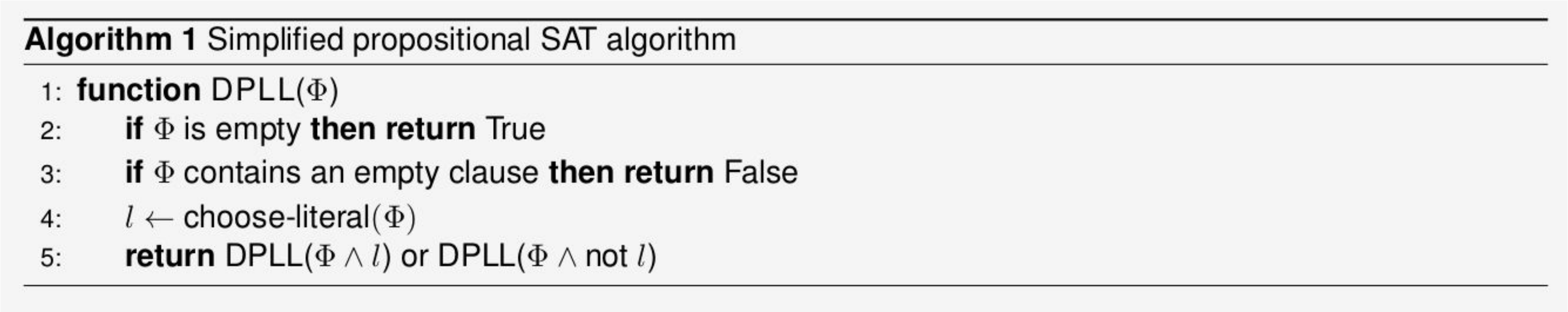


Craig interpolation theorem. Let φ_1, φ_2 be first-order sentences over the vocabularies $\mathcal{L}_1, \mathcal{L}_2$ such that $\varphi_1 \vDash \varphi_2$. Then there is a sentence θ over the common vocabulary $\mathcal{L}_1 \cap \mathcal{L}_2$, the *interpolant*, such that $\varphi_1 \vDash \theta$ and $\theta \vDash \varphi_2$.

```
SELECT DISTINCT v2.custname FROM V2 v2
WHERE v2.orderkey NOT IN
(SELECT orderkey FROM V1)
```

Neural architectures for SAT

Joint work with Sebastian Jaszczur, Michał Łuszczyk



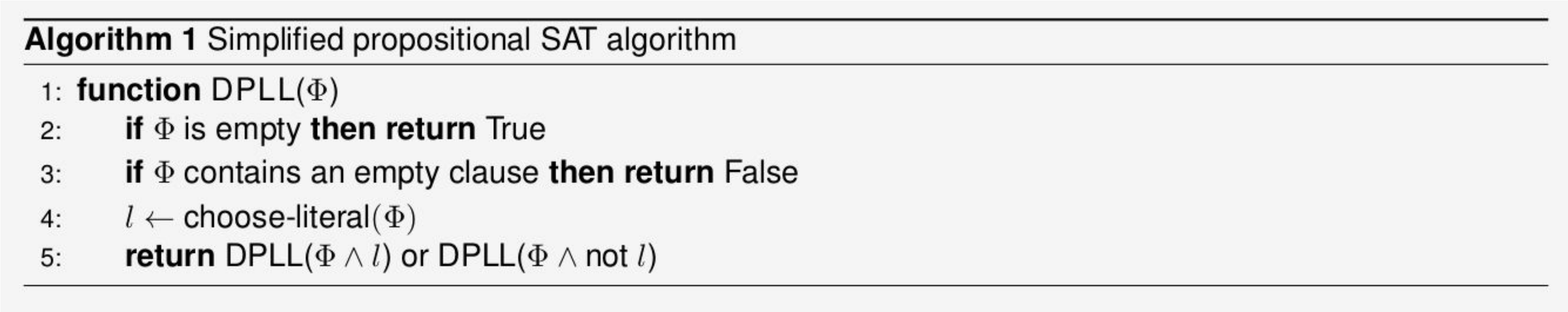
Algorithm 1 Simplified propositional SAT algorithm

```
function DPLL($\Phi$)
    if $\Phi$ is empty then return True
    if $\Phi$ contains an empty clause then return False
    $l \leftarrow$ choose-literal($\Phi$)
    return DPLL($\Phi \wedge l$) or DPLL($\Phi \wedge$ not $l$)
```

In this experiment we deal with formulas given in the conjunctive normal form (CNF):

$$
\begin{aligned}
&(x_1 \vee x_2 \vee \ldots \vee x_n) \wedge\\
&(y_1 \vee x_2 \vee \ldots \vee x_n) \wedge\\
&(x_1 \vee y_2 \vee \ldots \vee x_n) \wedge\\
&(y_1 \vee y_2 \vee \ldots \vee x_n) \wedge \ldots \wedge\\
&(x_1 \vee x_2 \vee \ldots \vee y_n) \wedge\\
&(y_1 \vee x_2 \vee \ldots \vee y_n) \wedge\\
&(x_1 \vee y_2 \vee \ldots \vee y_n) \wedge\\
&(y_1 \vee y_2 \vee \ldots \vee y_n);
\end{aligned}
$$

$$
\begin{aligned}
&(l_1 \vee l_2 \vee x_2) \wedge\\
&(\neg x_2 \vee l_3 \vee x_3) \wedge\\
&(\neg x_3 \vee l_4 \vee x_4) \wedge \cdots \wedge\\
&(\neg x_{n-3} \vee l_{n-2} \vee x_{n-2}) \wedge\\
&(\neg x_{n-2} \vee l_{n-1} \vee l_n)
\end{aligned}
$$



Algorithm 1 Simplified propositional SAT algorithm

```
function DPLL($\Phi$)
    if $\Phi$ is empty then return True
    if $\Phi$ contains an empty clause then return False
    $l \leftarrow$ choose-literal($\Phi$)
    return DPLL($\Phi \wedge l$) or DPLL($\Phi \wedge$ not $l$)
```

In this experiment we want to do things which are sound and simple from engineering point of view:

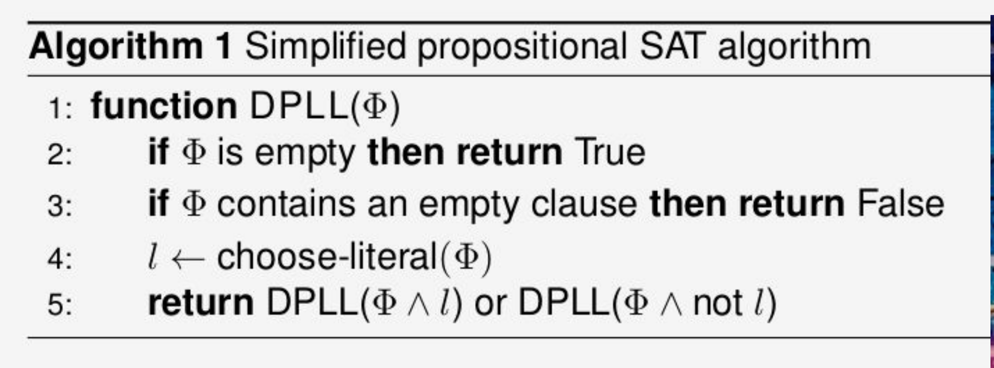
Done:

- Very simple implementation of DPLL for prototyping (allows for various heuristics)
- Generators random, but non-trivial instances
- Various neural architectures compared using the above infrastructure
- Generic distributed training on a cluster

Not done:

- Integration with a more sophisticated solver
- Training/testing on large and very large instances
- Implementation of DPLL in tf (aka on GPU)
- TPU-compliant implementation of NeuroDPLL

Goal: Learned policy or embedding which outdoes basic heuristics and generalizes to large instances than these in training



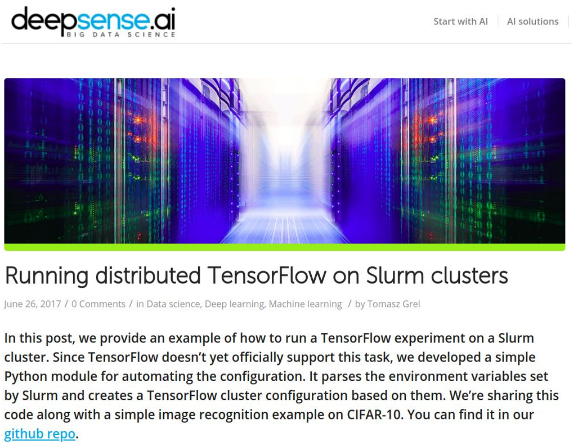
Algorithm 1 Simplified propositional SAT algorithm

```
function DPLL($\Phi$)
    if $\Phi$ is empty then return True
    if $\Phi$ contains an empty clause then return False
    $l \leftarrow$ choose-literal($\Phi$)
    return DPLL($\Phi \wedge l$) or DPLL($\Phi \wedge$ not $l$)
```

In this experiment we want to do things which are sound and simple from engineering point of view:

Done:

- Very simple implementation of DPLL for prototyping (allows for various heuristics)
- Generators random, but trivial instances
- Various neural architectures compared using the above infrastructure
- Generic distributed training on a cluster

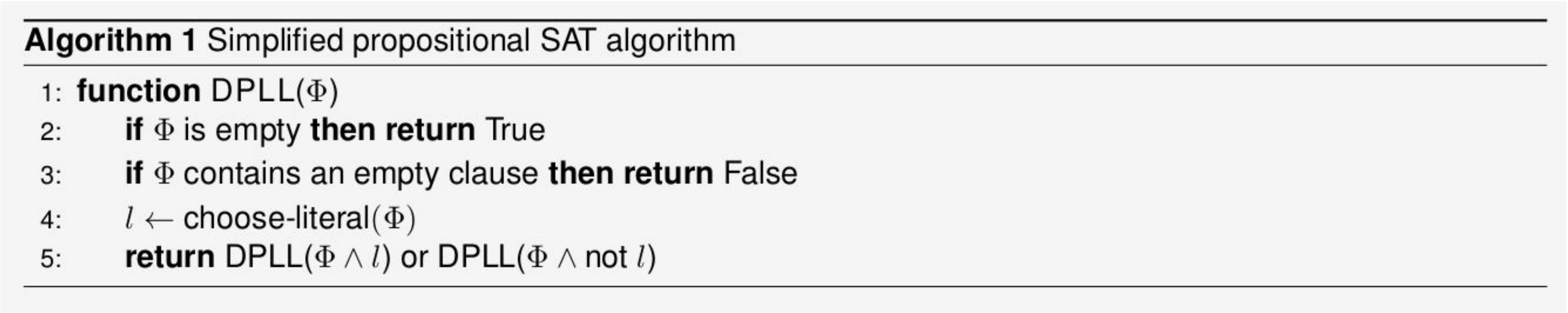


Running distributed TensorFlow on Slurm clusters

June 26, 2017 / 0 Comments / in Data science, Deep learning, Machine learning / by Tomasz Grel

In this post, we provide an example of how to run a TensorFlow experiment on a Slurm cluster. Since TensorFlow doesn't yet officially support this task, we developed a simple Python module for automating the configuration. It parses the environment variables set by Slurm and creates a TensorFlow cluster configuration based on them. We're sharing this code along with a simple image recognition example on CIFAR-10. You can find it in our github repo.

Goal: Learned policy or embedding which outdoes basic heuristics and generalizes to large instances than these in training



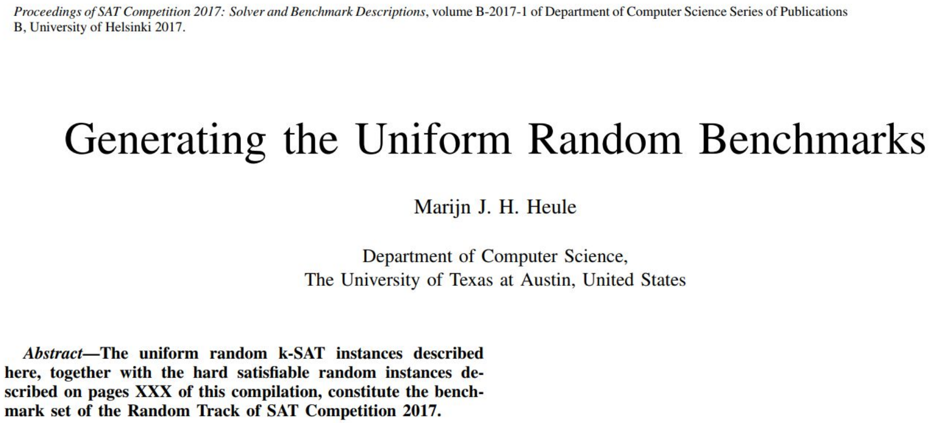
Algorithm 1 Simplified propositional SAT algorithm

```
function DPLL($\Phi$)
    if $\Phi$ is empty then return True
    if $\Phi$ contains an empty clause then return False
    $l \leftarrow$ choose-literal($\Phi$)
    return DPLL($\Phi \wedge l$) or DPLL($\Phi \wedge$ not $l$)
```

In this experiment we want to do things which are sound and simple from engineering point of view:

Goal: Learned policy or embedding which outdoes basic heuristics and generalizes to large instances than these in training

Long-term goal: learn the distributions in the random SAT benchmark



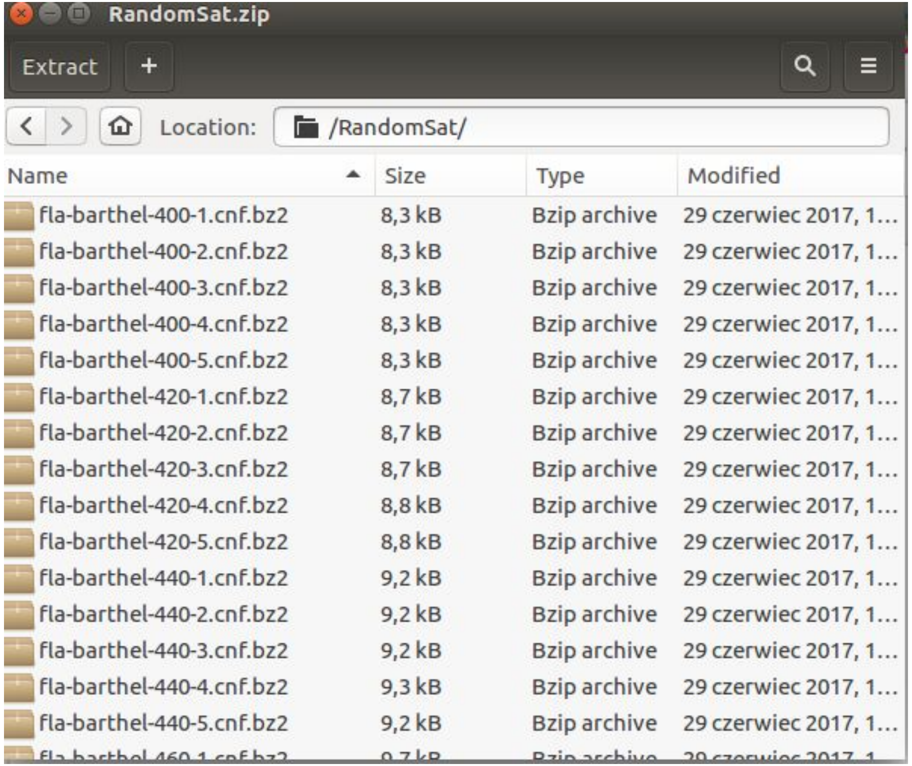
Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions, volume B-2017-1 of Department of Computer Science Series of Publications B, University of Helsinki 2017.

Generating the Uniform Random Benchmarks

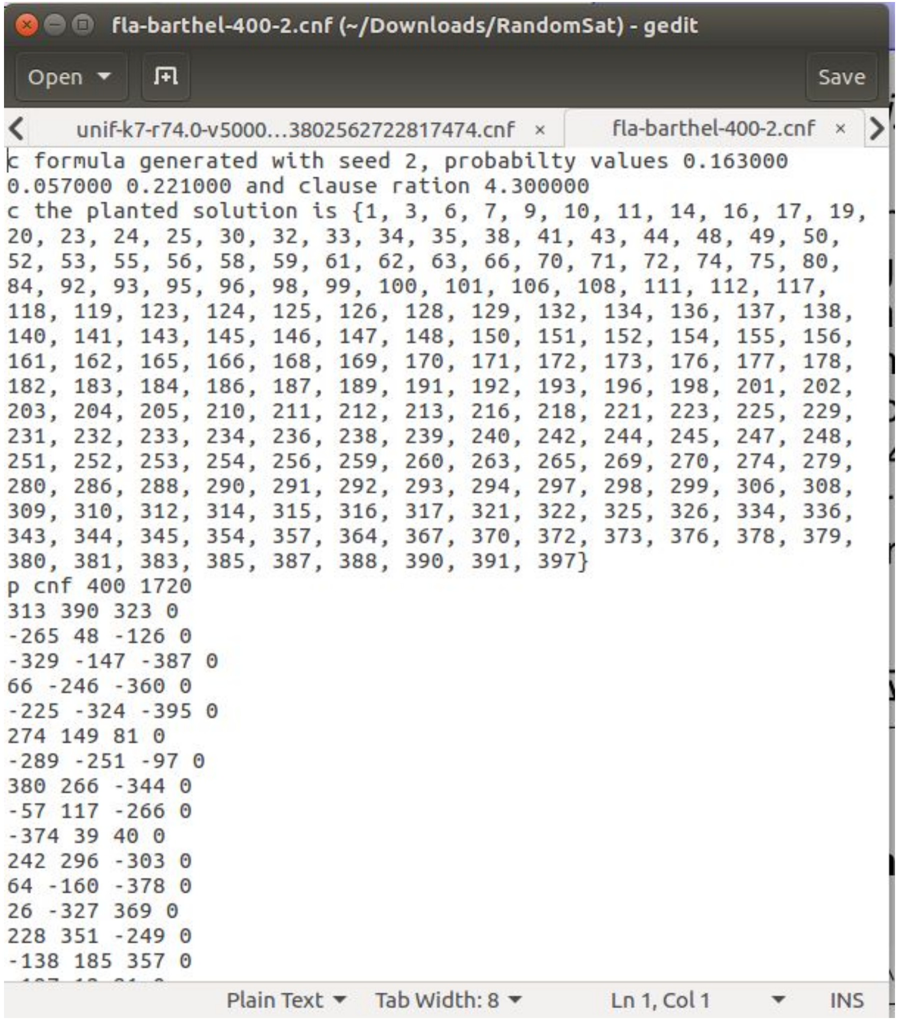
Marijn J. H. Heule

Department of Computer Science,
The University of Texas at Austin, United States

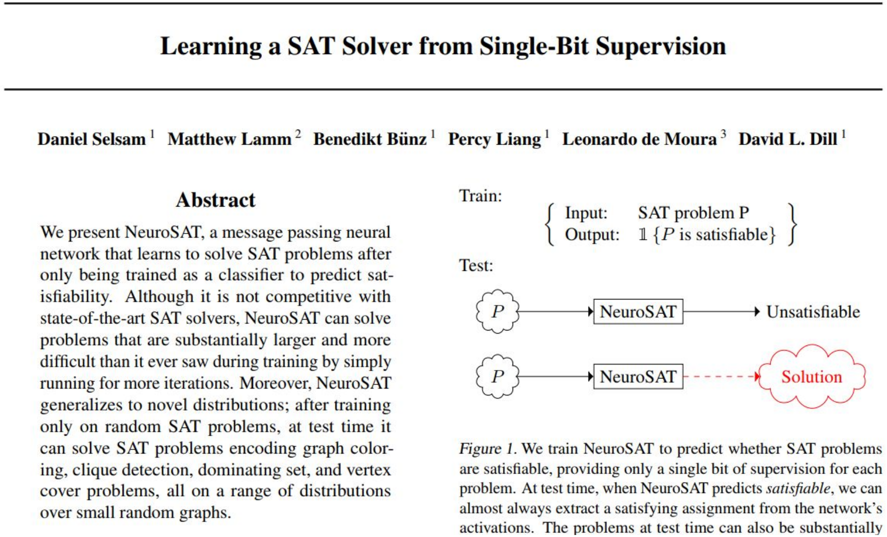
***Abstract*—The uniform random k-SAT instances described here, together with the hard satisfiable random instances described on pages XXX of this compilation, constitute the benchmark set of the Random Track of SAT Competition 2017.**



Name	Size	Type	Modified
fla-barthel-400-1.cnf.bz2	8,3 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-400-2.cnf.bz2	8,3 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-400-3.cnf.bz2	8,3 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-400-4.cnf.bz2	8,3 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-400-5.cnf.bz2	8,3 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-420-1.cnf.bz2	8,7 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-420-2.cnf.bz2	8,7 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-420-3.cnf.bz2	8,7 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-420-4.cnf.bz2	8,8 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-420-5.cnf.bz2	8,8 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-440-1.cnf.bz2	9,2 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-440-2.cnf.bz2	9,2 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-440-3.cnf.bz2	9,2 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-440-4.cnf.bz2	9,3 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-440-5.cnf.bz2	9,2 kB	Bzip archive	29 czerwiec 2017, 1...
fla-barthel-460-1.cnf.bz2	9,7 kB	Bzip archive	29 czerwiec 2017, 1...


fla-barthel-400-2.cnf (~/Downloads/RandomSat) - gedit
Open
Save
unif-k7-r74.0-v5000...3802562722817474.cnf
fla-barthel-400-2.cnf
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

```
c formula generated with seed 2, probabilty values 0.163000
0.057000 0.221000 and clause ration 4.300000
c the planted solution is {1, 3, 6, 7, 9, 10, 11, 14, 16, 17, 19,
20, 23, 24, 25, 30, 32, 33, 34, 35, 38, 41, 43, 44, 48, 49, 50,
52, 53, 55, 56, 58, 59, 61, 62, 63, 66, 70, 71, 72, 74, 75, 80,
84, 92, 93, 95, 96, 98, 99, 100, 101, 106, 108, 111, 112, 117,
118, 119, 123, 124, 125, 126, 128, 129, 132, 134, 136, 137, 138,
140, 141, 143, 145, 146, 147, 148, 150, 151, 152, 154, 155, 156,
161, 162, 165, 166, 168, 169, 170, 171, 172, 173, 176, 177, 178,
182, 183, 184, 186, 187, 189, 191, 192, 193, 196, 198, 201, 202,
203, 204, 205, 210, 211, 212, 213, 216, 218, 221, 223, 225, 229,
231, 232, 233, 234, 236, 238, 239, 240, 242, 244, 245, 247, 248,
251, 252, 253, 254, 256, 259, 260, 263, 265, 269, 270, 274, 279,
280, 286, 288, 290, 291, 292, 293, 294, 297, 298, 299, 306, 308,
309, 310, 312, 314, 315, 316, 317, 321, 322, 325, 326, 334, 336,
343, 344, 345, 354, 357, 364, 367, 370, 372, 373, 376, 378, 379,
380, 381, 383, 385, 387, 388, 390, 391, 397}
p cnf 400 1720
313 390 323 0
-265 48 -126 0
-329 -147 -387 0
66 -246 -360 0
-225 -324 -395 0
274 149 81 0
-289 -251 -97 0
380 266 -344 0
-57 117 -266 0
-374 39 40 0
242 296 -303 0
64 -160 -378 0
26 -327 369 0
228 351 -249 0
-138 185 357 0
```



Learning a SAT Solver from Single-Bit Supervision

Daniel Selsam[1] Matthew Lamm[2] Benedikt Bünz[1] Percy Liang[1] Leonardo de Moura[3] David L. Dill[1]

Abstract

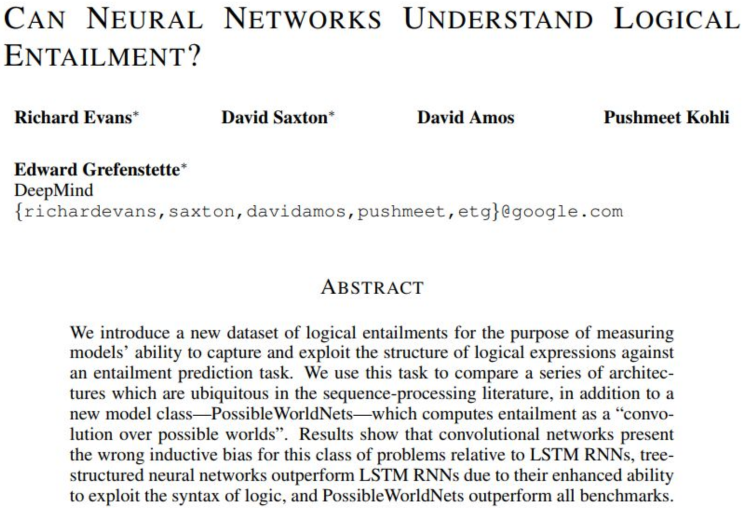
We present NeuroSAT, a message passing neural network that learns to solve SAT problems after only being trained as a classifier to predict satisfiability. Although it is not competitive with state-of-the-art SAT solvers, NeuroSAT can solve problems that are substantially larger and more difficult than it ever saw during training by simply running for more iterations. Moreover, NeuroSAT generalizes to novel distributions; after training only on random SAT problems, at test time it can solve SAT problems encoding graph coloring, clique detection, dominating set, and vertex cover problems, all on a range of distributions over small random graphs.

Train:

$$\left\{ \begin{array}{ll} \text{Input:} & \text{SAT problem P} \\ \text{Output:} & \mathbb{1}\{P \text{ is satisfiable}\} \end{array} \right\}$$

Test:

Figure 1. We train NeuroSAT to predict whether SAT problems are satisfiable, providing only a single bit of supervision for each problem. At test time, when NeuroSAT predicts *satisfiable*, we can almost always extract a satisfying assignment from the network's activations. The problems at test time can also be substantially



Can Neural Networks Understand Logical Entailment?

Richard Evans* David Saxton* David Amos Pushmeet Kohli

Edward Grefenstette*
DeepMind
{richardevans,saxton,davidamos,pushmeet,etg}@google.com

Abstract

We introduce a new dataset of logical entailments for the purpose of measuring models' ability to capture and exploit the structure of logical expressions against an entailment prediction task. We use this task to compare a series of architectures which are ubiquitous in the sequence-processing literature, in addition to a new model class—PossibleWorldNets—which computes entailment as a "convolution over possible worlds". Results show that convolutional networks present the wrong inductive bias for this class of problems relative to LSTM RNNs, tree-structured neural networks outperform LSTM RNNs due to their enhanced ability to exploit the syntax of logic, and PossibleWorldNets outperform all benchmarks.

DPLL + some embedding

```
Algorithm DPLL
  Input: A set of clauses $\Phi$.
  Output: A Truth Value.

function DPLL($\Phi$)
   if $\Phi$ is empty
       then return true;
   if $\Phi$ contains an empty clause
       then return false;
//    for every unit clause $l$ in $\Phi$
//       $\Phi \leftarrow$ unit-propagate($l$, $\Phi$);
//   for every literal $l$ that occurs pure in $\Phi$
//      $\Phi \leftarrow$ pure-literal-assign($l$, $\Phi$);
   $l \leftarrow$ choose-literal($\Phi$);
   return DPLL($\Phi \wedge$ l) or DPLL($\Phi \wedge$ not(l));
```

```
function choose-literal($\Phi$):
        for each variable v
                calculate embedding of $\Phi$[v=true]
                calculate embedding of $\Phi$[v=false]

        take literal for which embedding is farthest from False
```

DPLL + some trained policy choose-literal

```
Algorithm DPLL
  Input: A set of clauses $\Phi$.
  Output: A Truth Value.

function DPLL($\Phi$)
   if $\Phi$ is empty
       then return true;
   if $\Phi$ contains an empty clause
       then return false;
//    for every unit clause $l$ in $\Phi$
//        $\Phi \leftarrow$ unit-propagate($l$, $\Phi$);
//   for every literal $l$ that occurs pure in $\Phi$
//       $\Phi \leftarrow$ pure-literal-assign($l$, $\Phi$);
   $l \leftarrow$ choose-literal($\Phi$);
   return DPLL($\Phi \land$ l) or DPLL($\Phi \land$ not(l));
```

Axiomatization table for CNF

Axiom	EqNet	LSTM	NeuroSAT
“Variable renaming” - independent			
“Permutation of literals in clause” - independent			
“Permutation of clauses in formula” - independent			
“Negation of all occurrences of variable” - independent			

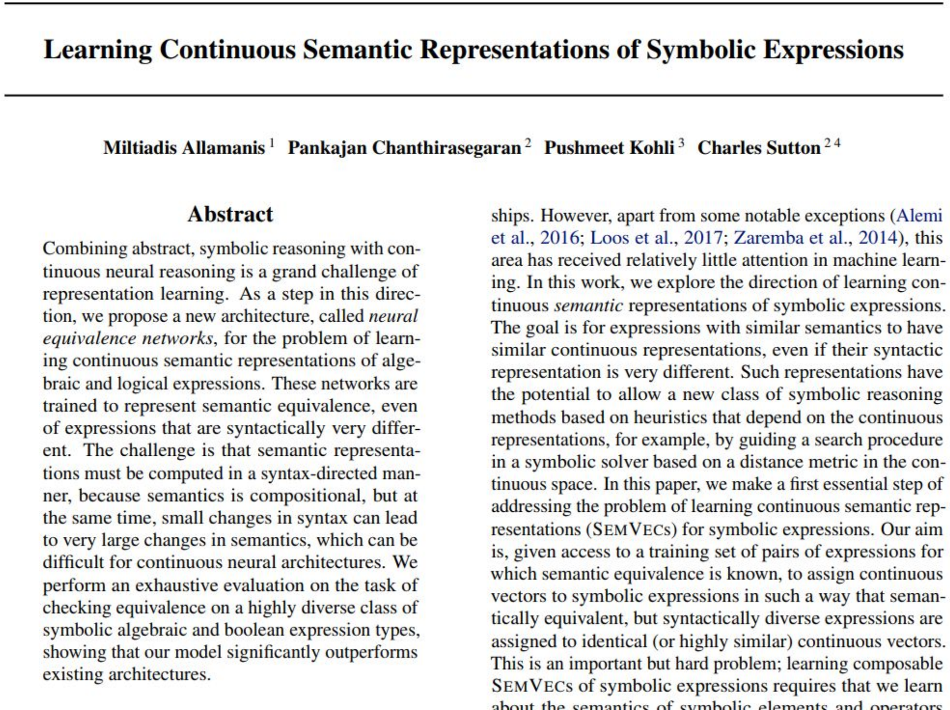
In this presentation I will conflate terms NeuroSAT, NeuroDPLL, GraphSAT, GraphDPLL …
- they all refer to a SAT policy based on a NeuroSAT-style architecture.

EQNET

Learning Continuous Semantic Representations of Symbolic Expressions

Goal: learn a mapping formula -> vector space

Semantically equivalent formulas should be close.



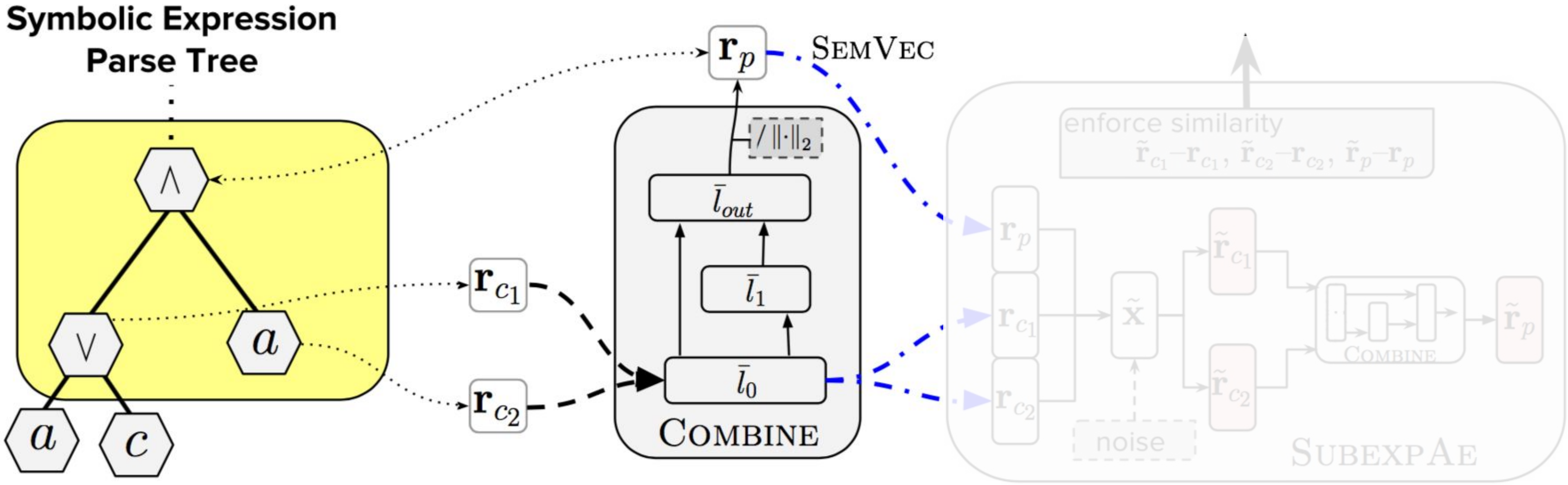
Learning Continuous Semantic Representations of Symbolic Expressions

Miltiadis Allamanis [1] Pankajan Chanthirasegaran [2] Pushmeet Kohli [3] Charles Sutton [2 4]

Abstract

Combining abstract, symbolic reasoning with continuous neural reasoning is a grand challenge of representation learning. As a step in this direction, we propose a new architecture, called *neural equivalence networks*, for the problem of learning continuous semantic representations of algebraic and logical expressions. These networks are trained to represent semantic equivalence, even of expressions that are syntactically very different. The challenge is that semantic representations must be computed in a syntax-directed manner, because semantics is compositional, but at the same time, small changes in syntax can lead to very large changes in semantics, which can be difficult for continuous neural architectures. We perform an exhaustive evaluation on the task of checking equivalence on a highly diverse class of symbolic algebraic and boolean expression types, showing that our model significantly outperforms existing architectures.

ships. However, apart from some notable exceptions (Alemi et al., 2016; Loos et al., 2017; Zaremba et al., 2014), this area has received relatively little attention in machine learning. In this work, we explore the direction of learning continuous *semantic* representations of symbolic expressions. The goal is for expressions with similar semantics to have similar continuous representations, even if their syntactic representation is very different. Such representations have the potential to allow a new class of symbolic reasoning methods based on heuristics that depend on the continuous representations, for example, by guiding a search procedure in a symbolic solver based on a distance metric in the continuous space. In this paper, we make a first essential step of addressing the problem of learning continuous semantic representations (SEMVECs) for symbolic expressions. Our aim is, given access to a training set of pairs of expressions for which semantic equivalence is known, to assign continuous vectors to symbolic expressions in such a way that semantically equivalent, but syntactically diverse expressions are assigned to identical (or highly similar) continuous vectors. This is an important but hard problem; learning composable SEMVECs of symbolic expressions requires that we learn about the semantics of symbolic elements and operators

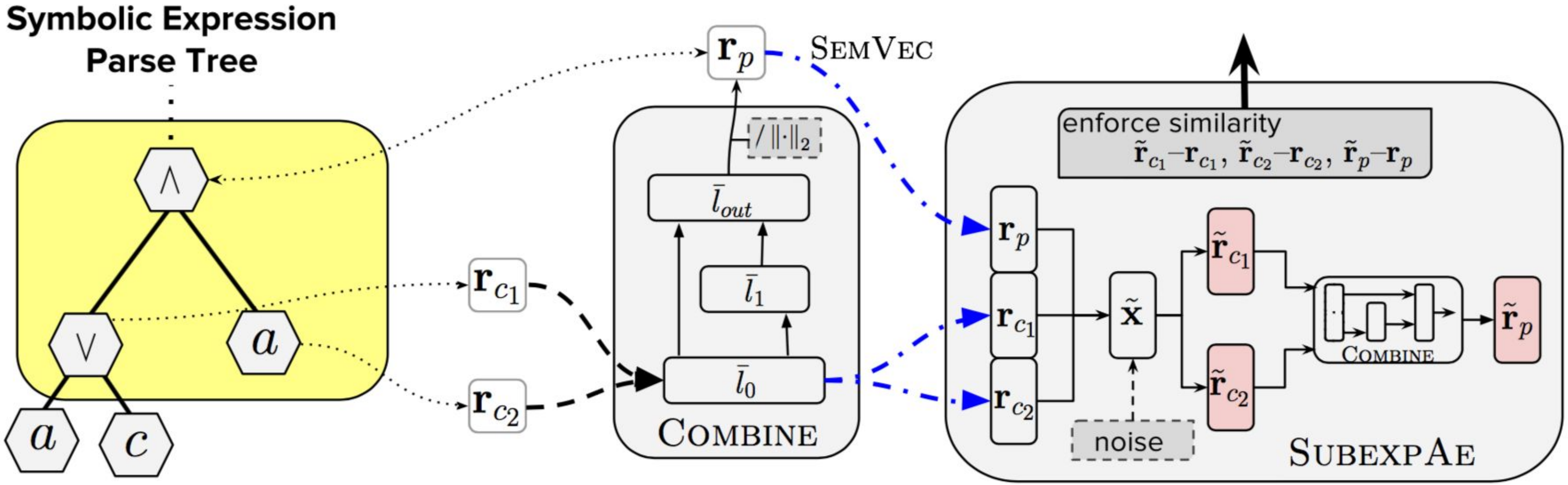


(a) Architectural diagram of EqNets. Example parse tree shown is of the boolean expression $(a \vee c) \wedge a$.

```
TreeNN (current node $n$)
  if $n$ is not a leaf then
    $\mathbf{r}_n \leftarrow$ Combine(TreeNN($c_0$), . . . , TreeNN($c_k$), $\tau_n$),
    where $(c_0, \ldots, c_k) = \mathrm{ch}(n)$
  else
    $\mathbf{r}_n \leftarrow$ LookupLeafEmbedding($\tau_n$)
  return $\mathbf{r}_n$
```

```
Combine ($\mathbf{r}_{c_0}, \ldots, \mathbf{r}_{c_k}, \tau_p$)
  $\bar{l}_0 \leftarrow [\mathbf{r}_{c_0}, \ldots, \mathbf{r}_{c_k}]$
  $\bar{l}_1 \leftarrow \sigma\left(W_{i,\tau_p} \cdot \bar{l}_0\right)$
  $\bar{l}_{out} \leftarrow W_{o0,\tau_p} \cdot \bar{l}_0 + W_{o1,\tau_p} \cdot \bar{l}_1$
  return $\bar{l}_{out} / \left\|\bar{l}_{out}\right\|_2$
```

(b) Combine of EqNet.



(a) Architectural diagram of EqNets. Example parse tree shown is of the boolean expression $(a \vee c) \wedge a$.

```
SubexpAe $(\mathbf{r}_{c_0}, \ldots, \mathbf{r}_{c_k}, \mathbf{r}_p, \tau_p)$
    $\mathbf{x} \leftarrow [\mathbf{r}_{c_0}, \ldots, \mathbf{r}_{c_k}]$
    $\tilde{\mathbf{x}} \leftarrow \tanh\left(W_d \cdot \tanh\left(W_{e,\tau_p} \cdot [\mathbf{r}_p, \mathbf{x}] \cdot \mathbf{n}\right)\right)$
    $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} \cdot \|\mathbf{x}\|_2 \,/\, \|\tilde{\mathbf{x}}\|_2$
    $\tilde{\mathbf{r}}_p \leftarrow \textsc{Combine}(\tilde{\mathbf{x}}, \tau_p)$
    return $-\left(\tilde{\mathbf{x}}^\top \mathbf{x} + \tilde{\mathbf{r}}_p^\top \mathbf{r}_p\right)$
```

(c) Loss function used for subexpression autoencoder

- goal: reversibility
- bottleneck model aka autoencoder
- “not used to compute the parent representation, regulariser only”

EQNET as a SAT oracle

Experiment:

1. trained eqnet: size 10, 5 vars. Operators AND, OR and NOT. Accuracy (claimed and reproduced): 85%.
2. calculate embeddings on a freshly generated set.
3. calculate distances from embedding of False
4. plot one histogram for satisfiable formulas and one from unsatisfiable formulas.

DPLL + eqnet

```
Algorithm DPLL
  Input: A set of clauses Φ.
  Output: A Truth Value.

function DPLL(Φ)
   if Φ is empty
       then return true;
   if Φ contains an empty clause
       then return false;
//    for every unit clause l in Φ
//       Φ ← unit-propagate(l, Φ);
//   for every literal l that occurs pure in Φ
//      Φ ← pure-literal-assign(l, Φ);
   l ← choose-literal(Φ);
   return DPLL(Φ ∧ l) or DPLL(Φ ∧ not(l));
```

```
function choose-literal(Φ):
        for each variable v
                calculate embedding of Φ[v=true]
                calculate embedding of Φ[v=false]

        take literal for which embedding is farthest from False
```

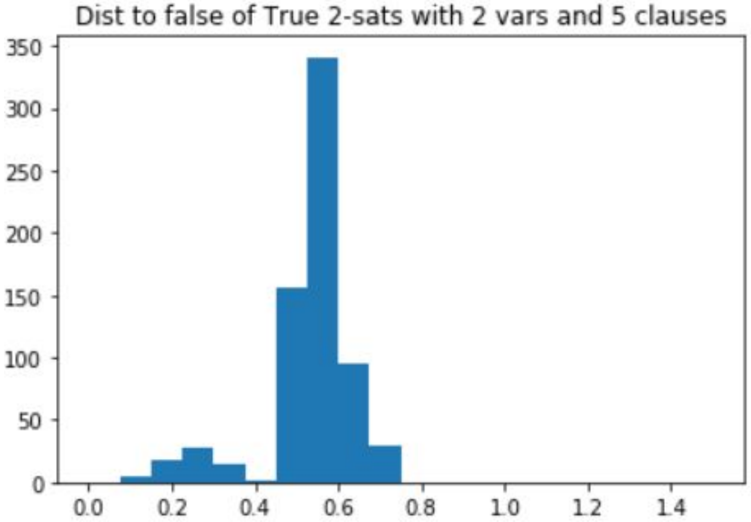
Experiment 2. Notebook link.

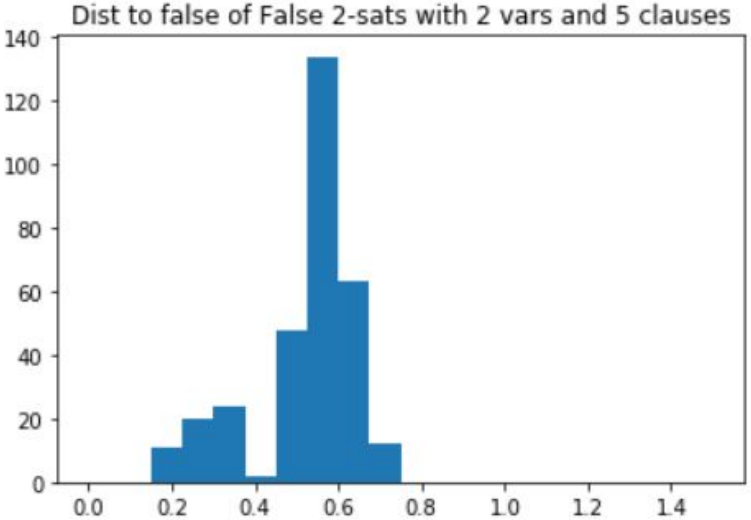
Dataset with random k-SATs:

2-SAT, 2 variables, 5 clauses

#Sat: 686; avg dist: 0.5334; stdev dist: 0.103

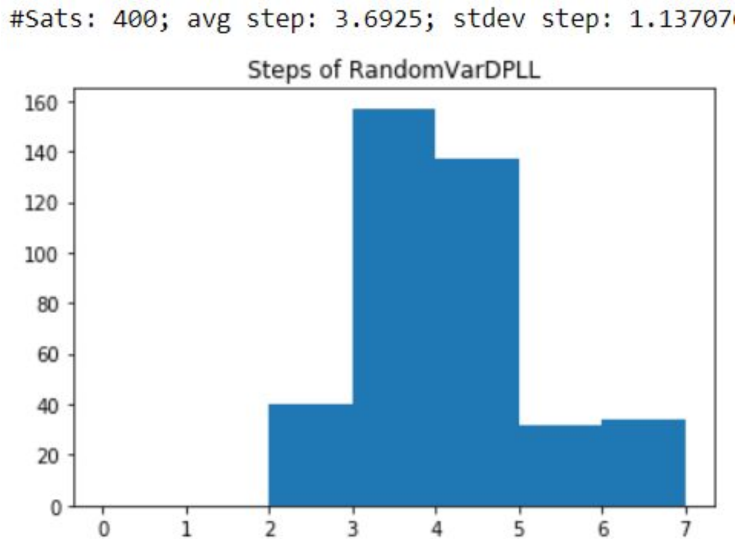
#Unsat: 314; avg dist: 0.5207; stdev dist: 0.124



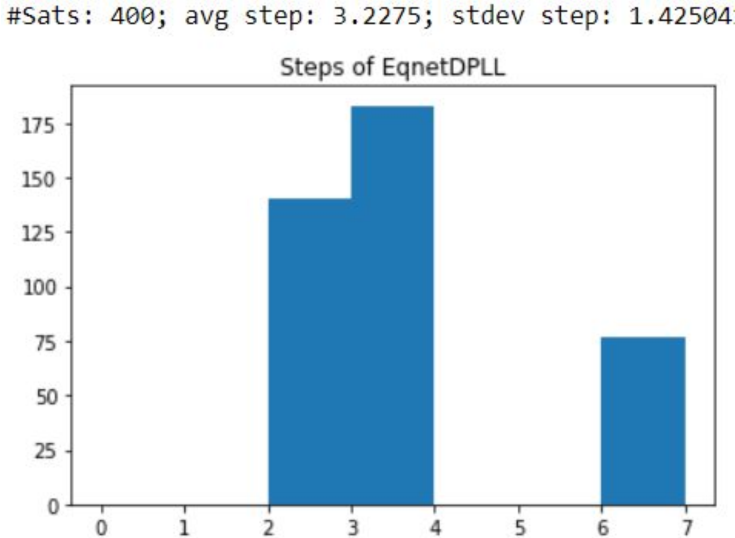


DPLL results: link

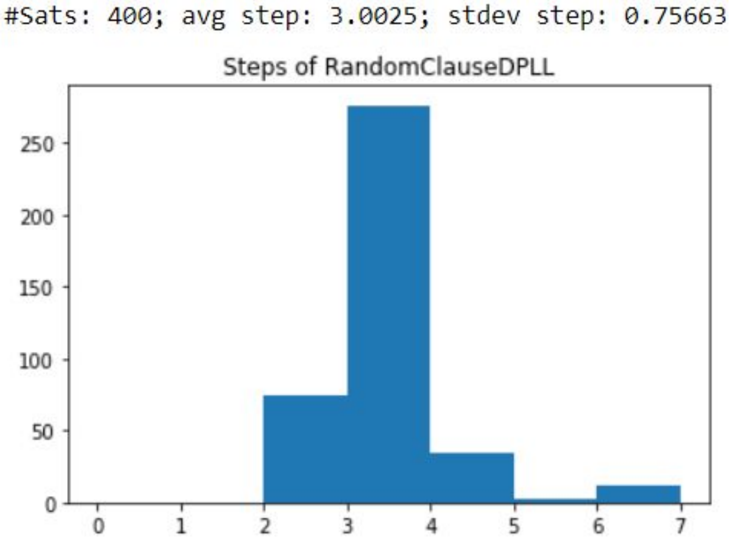
2-SAT,
2 variables,
3 clauses



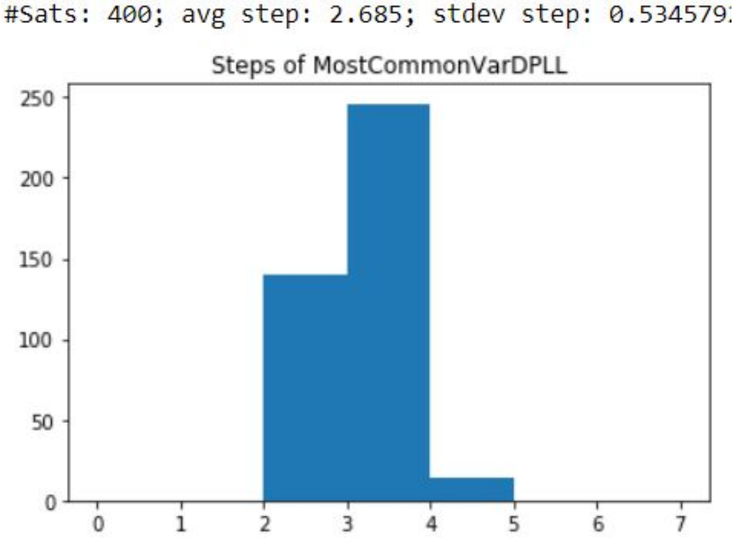
```
#Sats: 400; avg step: 3.6925; stdev step: 1.13707
```



```
#Sats: 400; avg step: 3.2275; stdev step: 1.42504
```



```
#Sats: 400; avg step: 3.0025; stdev step: 0.75663
```

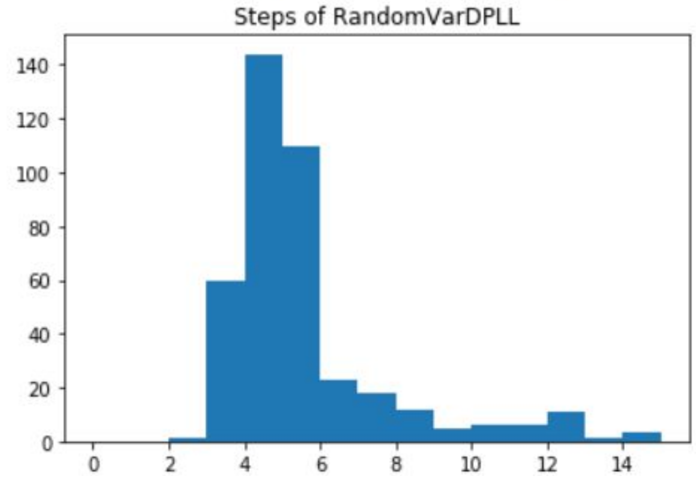


```
#Sats: 400; avg step: 2.685; stdev step: 0.534579
```

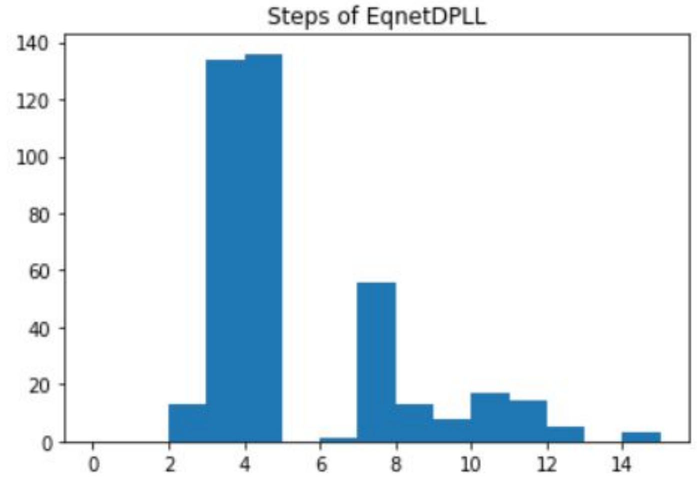
DPLL results: link

3-SAT,
3 variables,
6 clauses

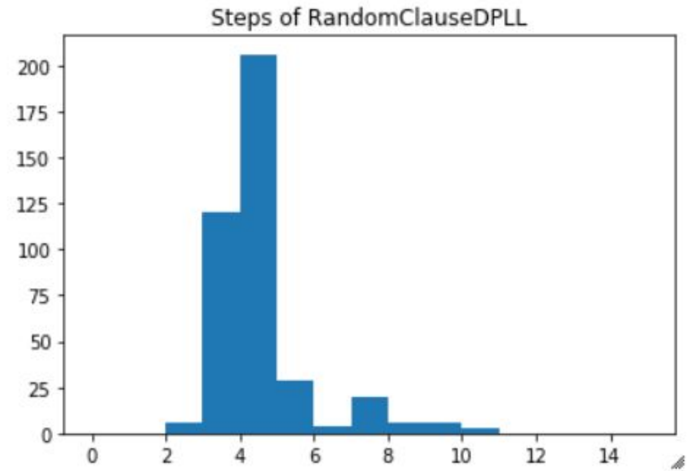
```
#Sats: 400; avg step: 5.0675; stdev step: 2.18241
```



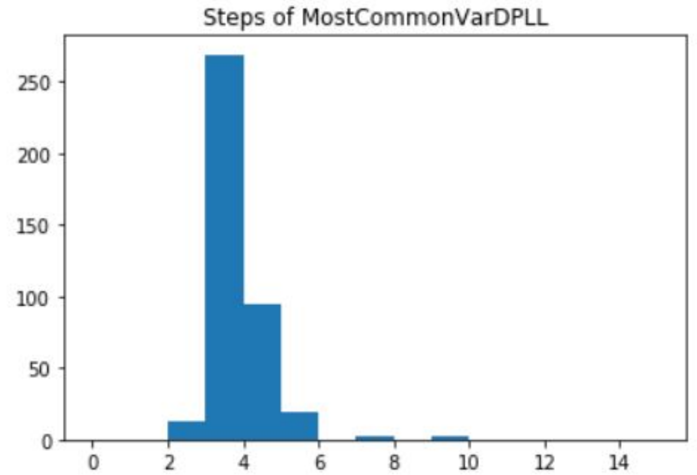
```
#Sats: 400; avg step: 4.93; stdev step: 2.5855560
```



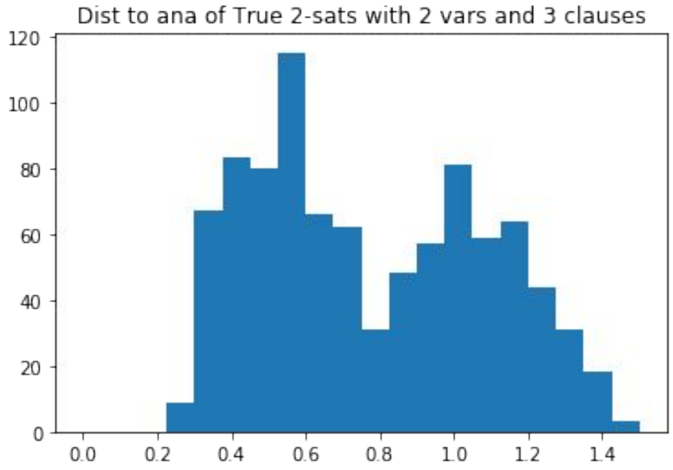
```
#Sats: 400; avg step: 4.0925; stdev step: 1.34124
```

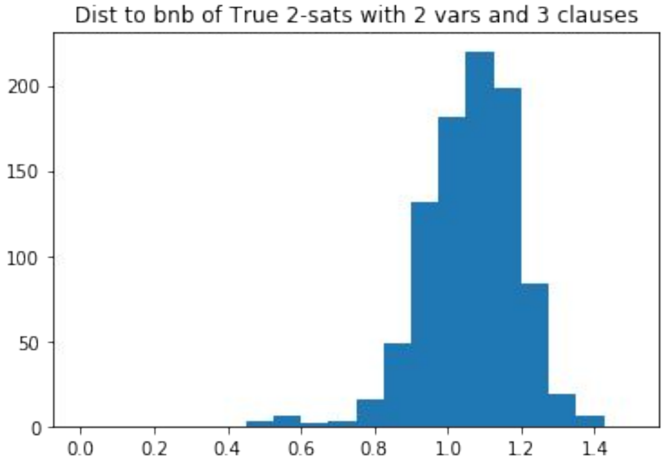


```
#Sats: 400; avg step: 3.35; stdev step: 0.7729812
```



Experiment: a&~a versus b&~b Notebook link





distance from/to	**a&~a**	**b&~b**	**c&~c**
a&~a	0.0	1.14	1.07
b&~b	1.14	0.0	0.9
c&~c	1.07	0.9	0.0

Standard solutions: why not LSTM?

Treat like a sequence: (A v ~C v B) ^ (~B ^ C) ^ (~A v B v D)

- Problems with LSTMs and similar:
    - 😟 Permutation of variables inside a clause changes the result.
    - 😟 Permutation of clauses changes the result.
    - 😟 Negation of all occurences of a variable changes the result.
    - 😟 Definite limit on embedding size, regardless of the size of the formula.

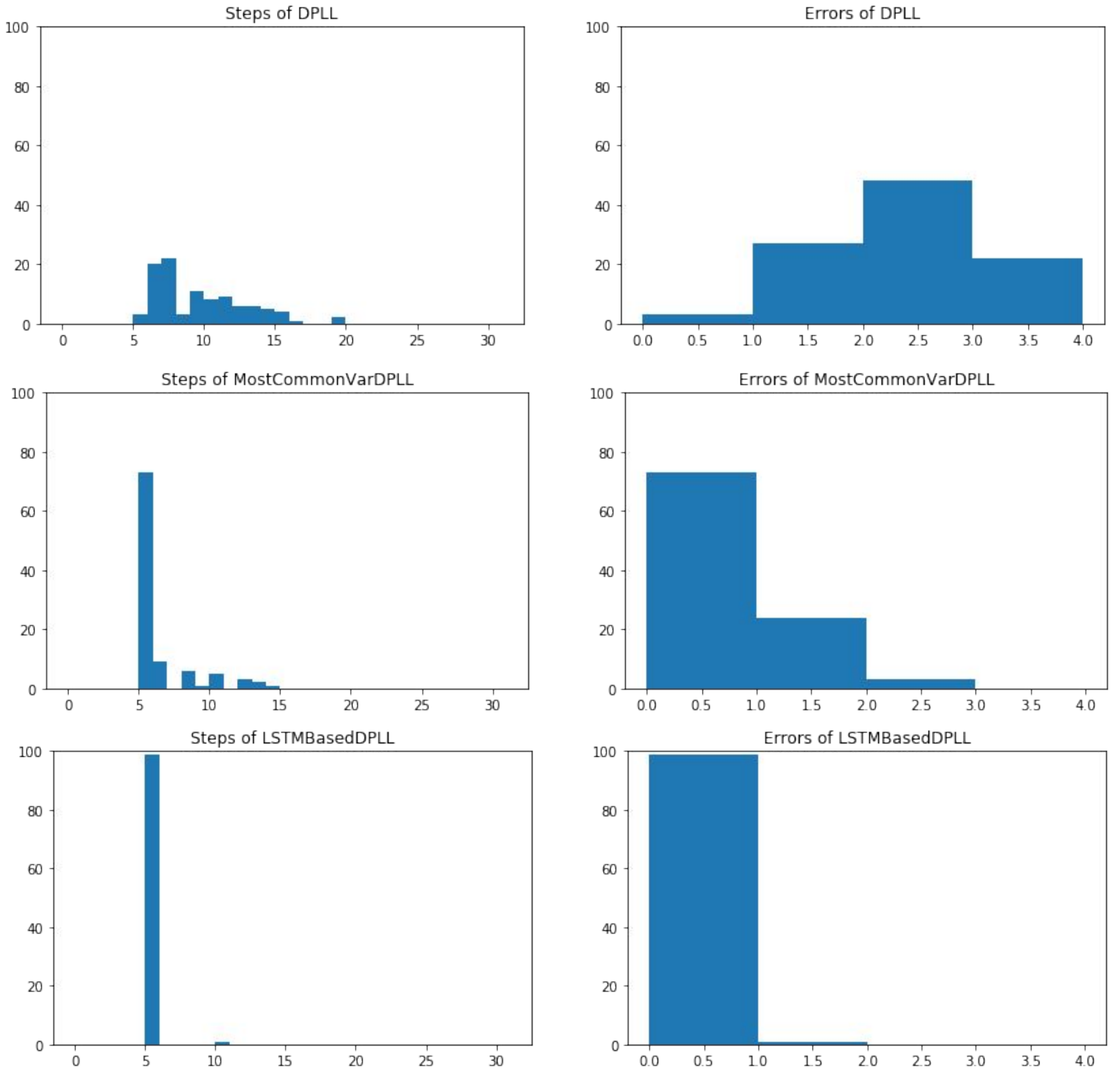
Slides: goo.gl/JyRc15

Experiment 1.

#vars: 4

trained on up to 40 clauses

test on up to 70 clauses

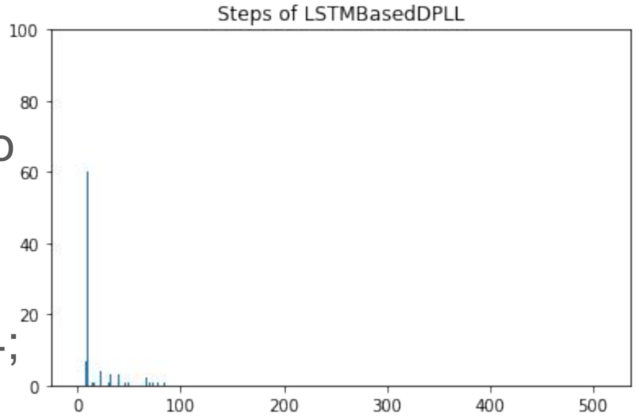


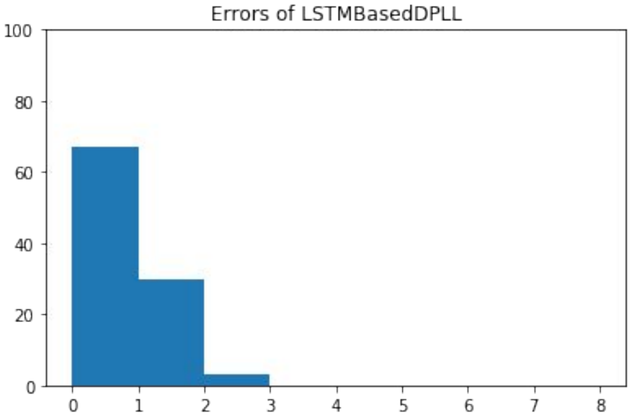
Notebook

Experiment 3.

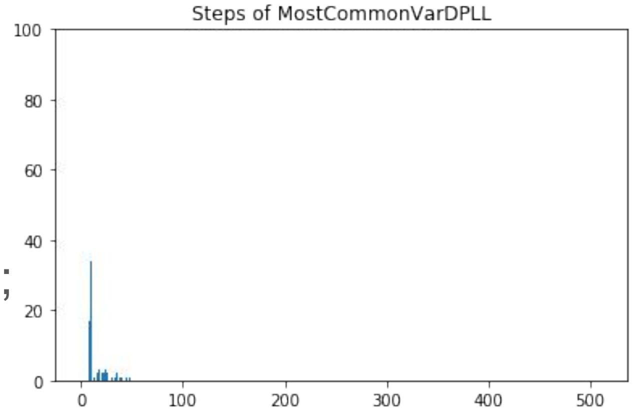
#vars: 8, #clauses: up to 50

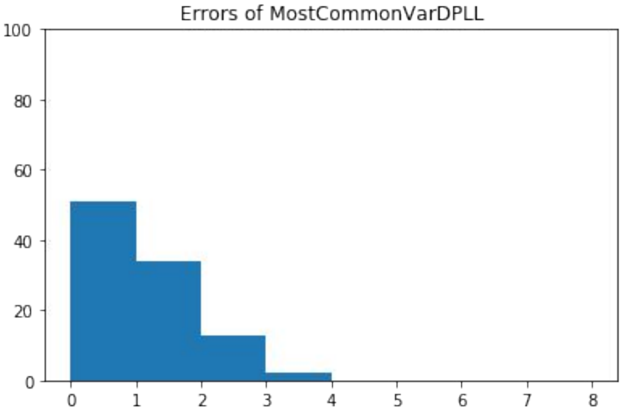
#Sats: 100; avg step: 19.73; stdev step: 19.94; avg error: 0.36; stdev error: 0.54





#Sats: 100; avg step: 15.27; stdev step: 11.05; avg error: 0.66; stdev error: 0.78





Learning a SAT Solver from Single-Bit Supervision

Daniel Selsam [1] Matthew Lamm [2] Benedikt Bünz [1] Percy Liang [1] Leonardo de Moura [3] David L. Dill [1]

Abstract

We present NeuroSAT, a message passing neural network that learns to solve SAT problems after only being trained as a classifier to predict satisfiability. Although it is not competitive with state-of-the-art SAT solvers, NeuroSAT can solve problems that are substantially larger and more difficult than it ever saw during training by simply running for more iterations. Moreover, NeuroSAT generalizes to novel distributions; after training only on random SAT problems, at test time it can solve SAT problems encoding graph coloring, clique detection, dominating set, and vertex cover problems, all on a range of distributions over small random graphs.

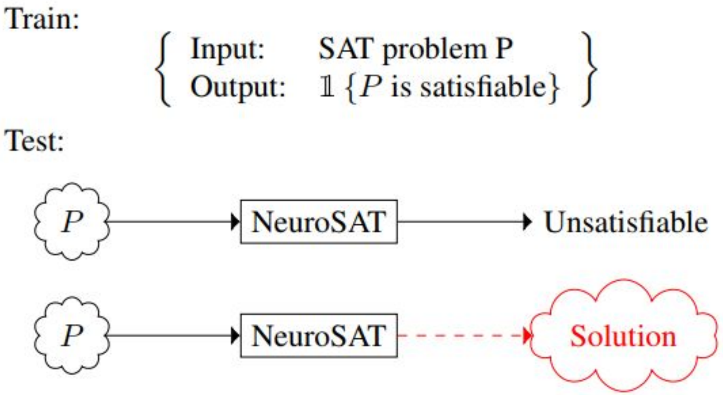


Figure 1. We train NeuroSAT to predict whether SAT problems are satisfiable, providing only a single bit of supervision for each problem. At test time, when NeuroSAT predicts *satisfiable*, we can almost always extract a satisfying assignment from the network's activations. The problems at test time can also be substantially

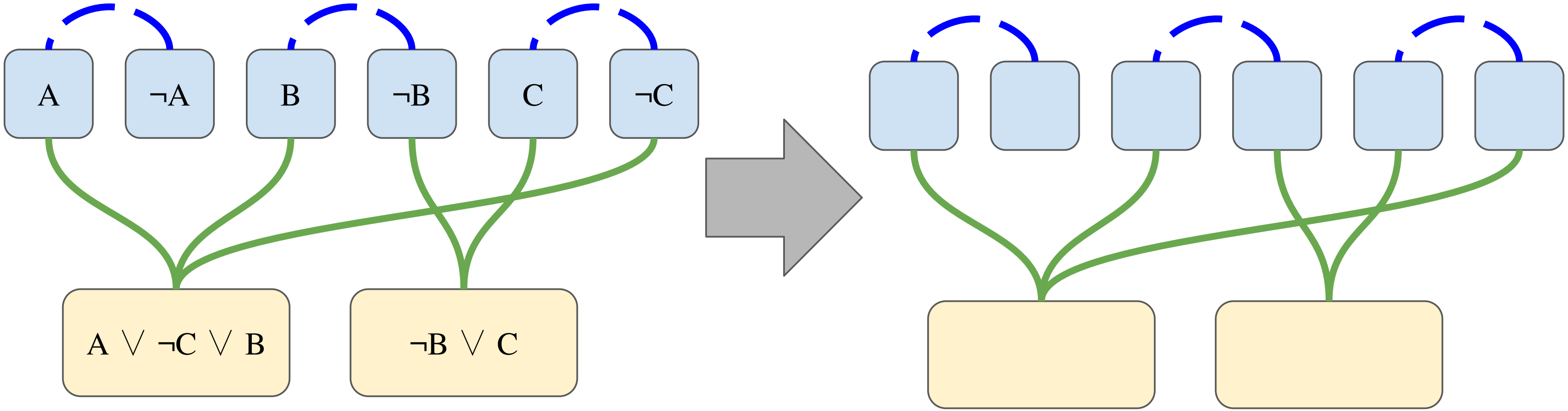
Axiomatization table for CNF

Axiom	EqNet	LSTM	NeuroSAT
“Variable renaming” - independent	No	No	Yes
“Permutation of literals in clause” - independent	No	No*	Yes
“Permutation of clauses in formula” - independent	No	No	Yes
“Negation of all occurrences of variable” - independent	No	No	Yes

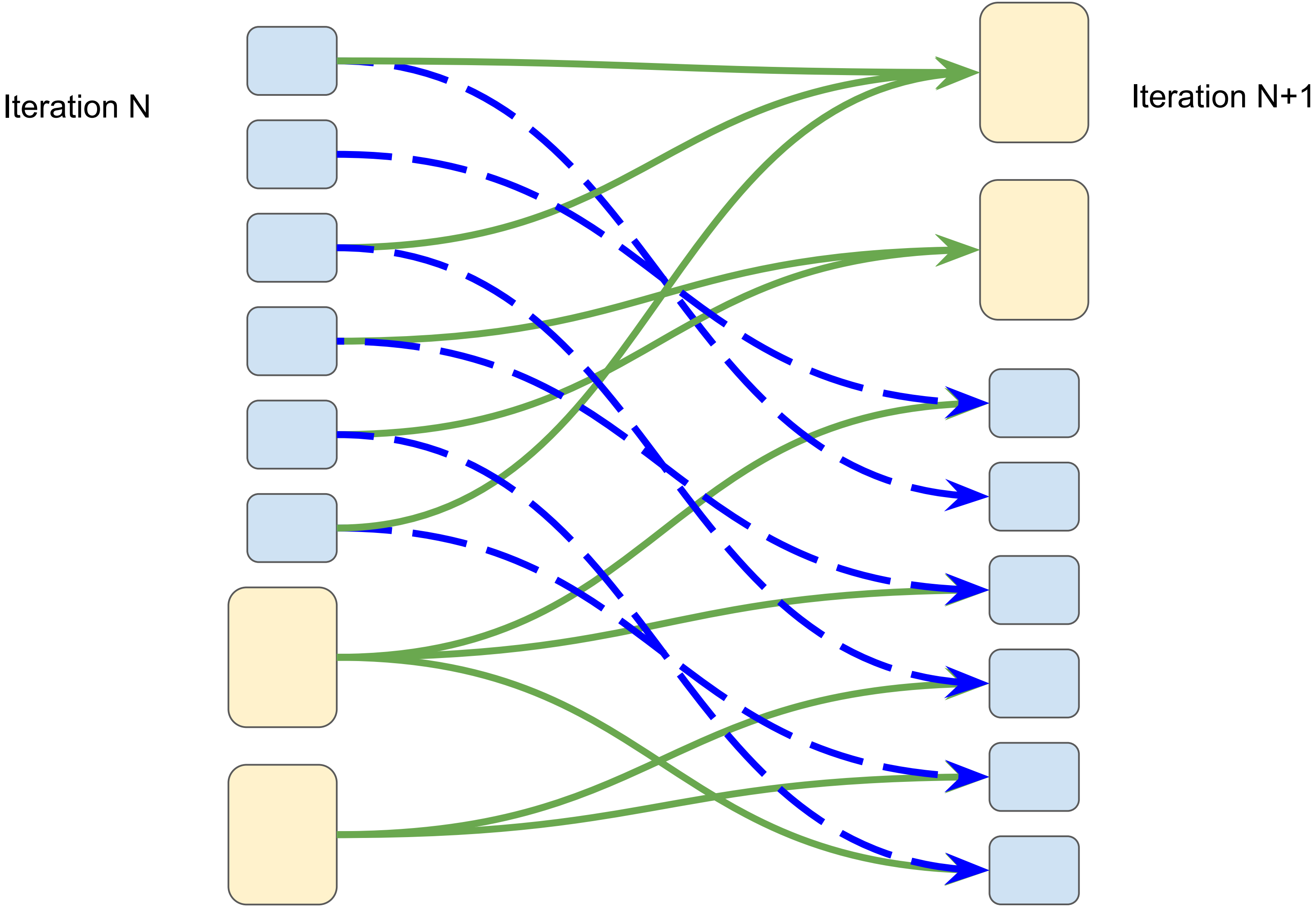
*Unless Bag of Words is used for in-clause literals aggregation.

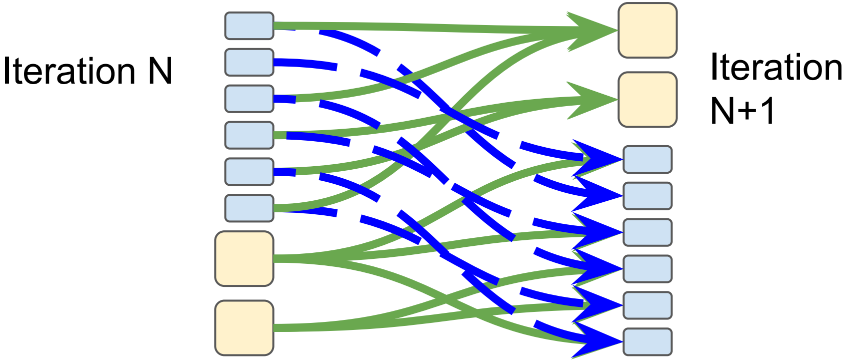
We may forget the names

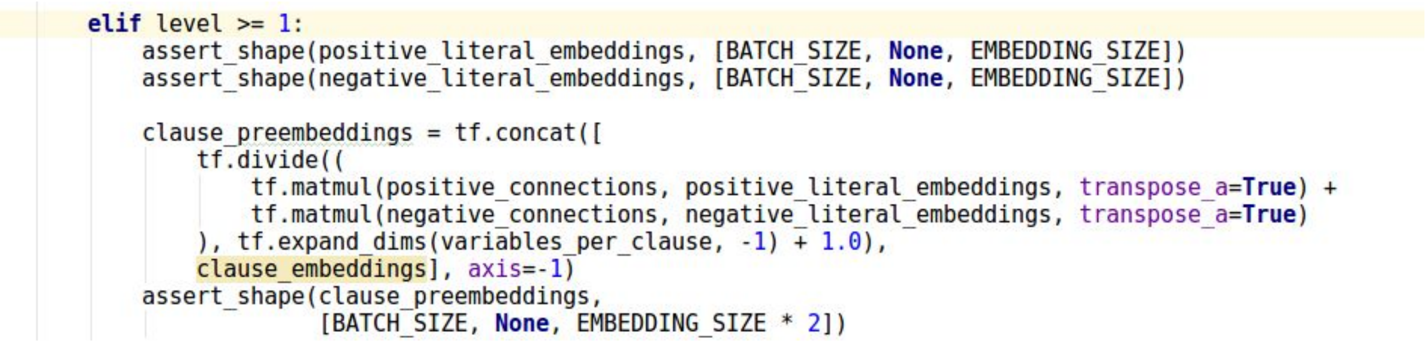
A graph representation without names still has all important information!



Slides: goo.gl/JyRc15

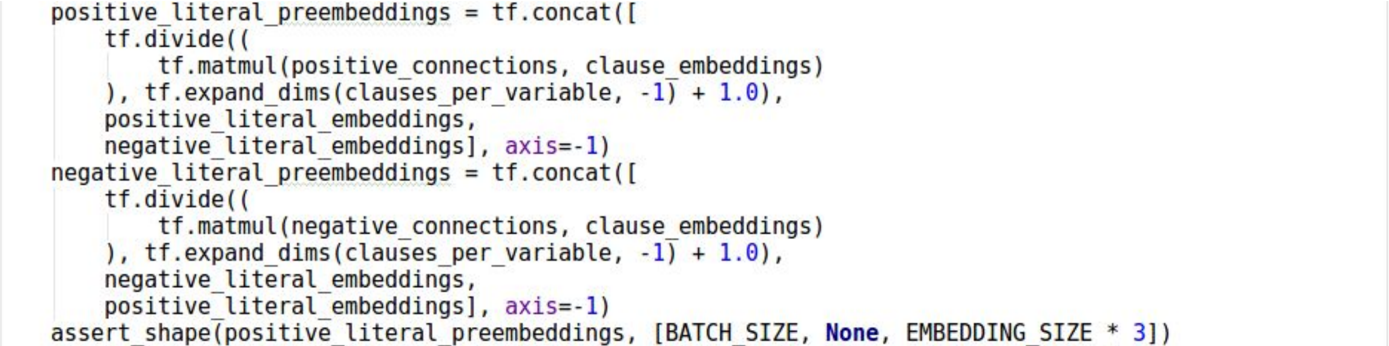

Iteration N
Iteration N+1



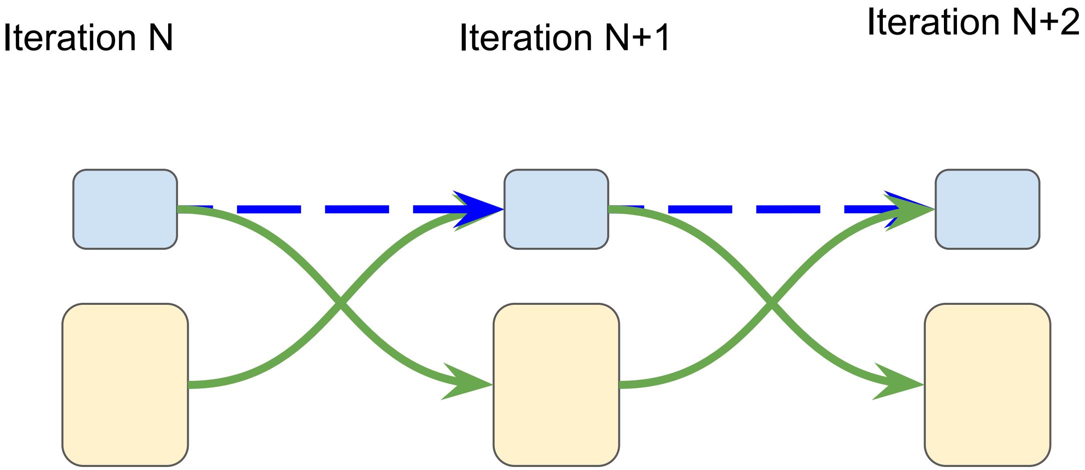


```
elif level >= 1:
    assert_shape(positive_literal_embeddings, [BATCH_SIZE, None, EMBEDDING_SIZE])
    assert_shape(negative_literal_embeddings, [BATCH_SIZE, None, EMBEDDING_SIZE])

    clause_preembeddings = tf.concat([
        tf.divide((
            tf.matmul(positive_connections, positive_literal_embeddings, transpose_a=True) +
            tf.matmul(negative_connections, negative_literal_embeddings, transpose_a=True)
        ), tf.expand_dims(variables_per_clause, -1) + 1.0),
        clause_embeddings], axis=-1)
    assert_shape(clause_preembeddings,
                 [BATCH_SIZE, None, EMBEDDING_SIZE * 2])
```



```
positive_literal_preembeddings = tf.concat([
    tf.divide((
        tf.matmul(positive_connections, clause_embeddings)
    ), tf.expand_dims(clauses_per_variable, -1) + 1.0),
    positive_literal_embeddings,
    negative_literal_embeddings], axis=-1)
negative_literal_preembeddings = tf.concat([
    tf.divide((
        tf.matmul(negative_connections, clause_embeddings)
    ), tf.expand_dims(clauses_per_variable, -1) + 1.0),
    negative_literal_embeddings,
    positive_literal_embeddings], axis=-1)
assert_shape(positive_literal_preembeddings, [BATCH_SIZE, None, EMBEDDING_SIZE * 3])
```



Iteration N

Iteration N+1

Iteration N+2

Architecture idea: embeddings get copied from previous iteration

- There are blue connections between negating literals
- There are green connections\between clauses and literals.

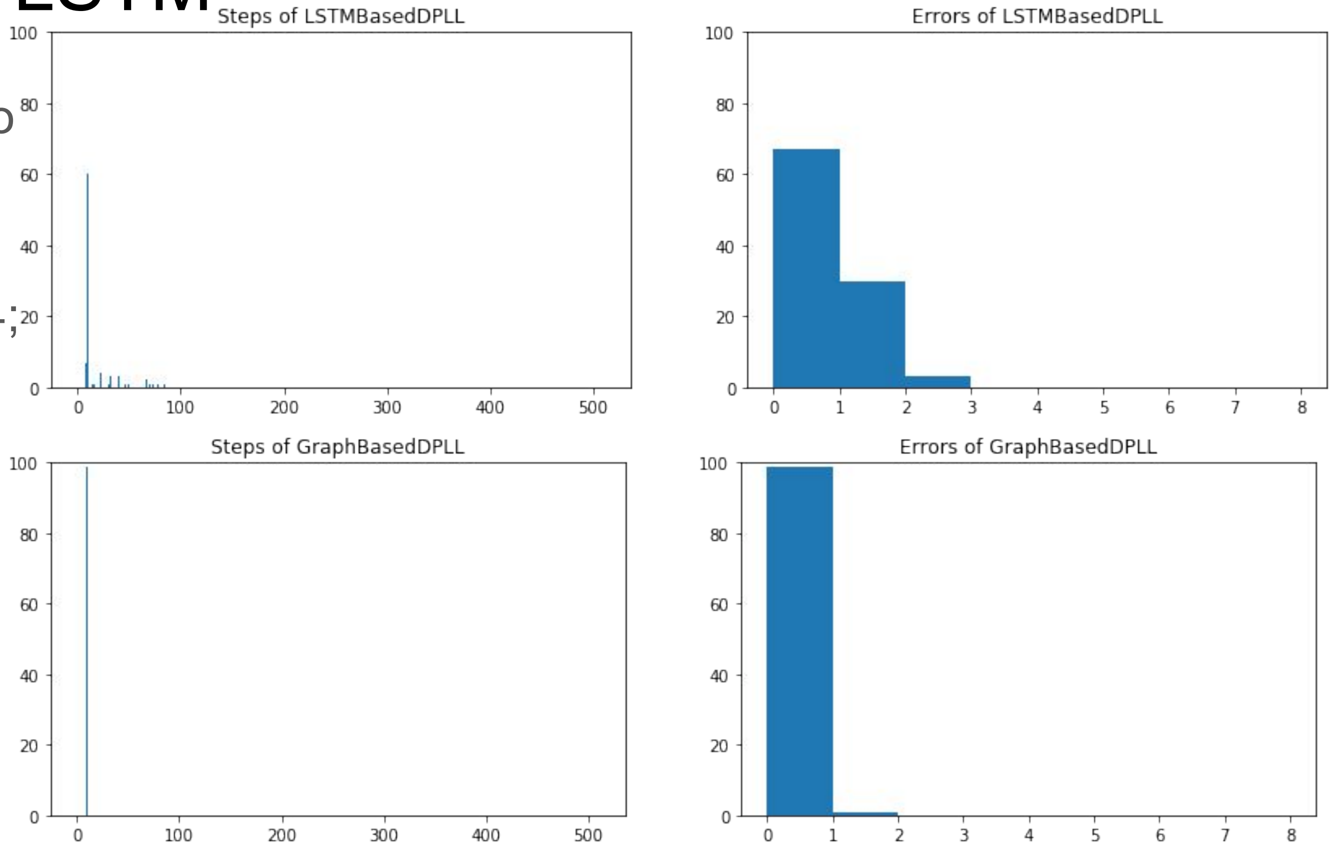
Notebook and Notebook

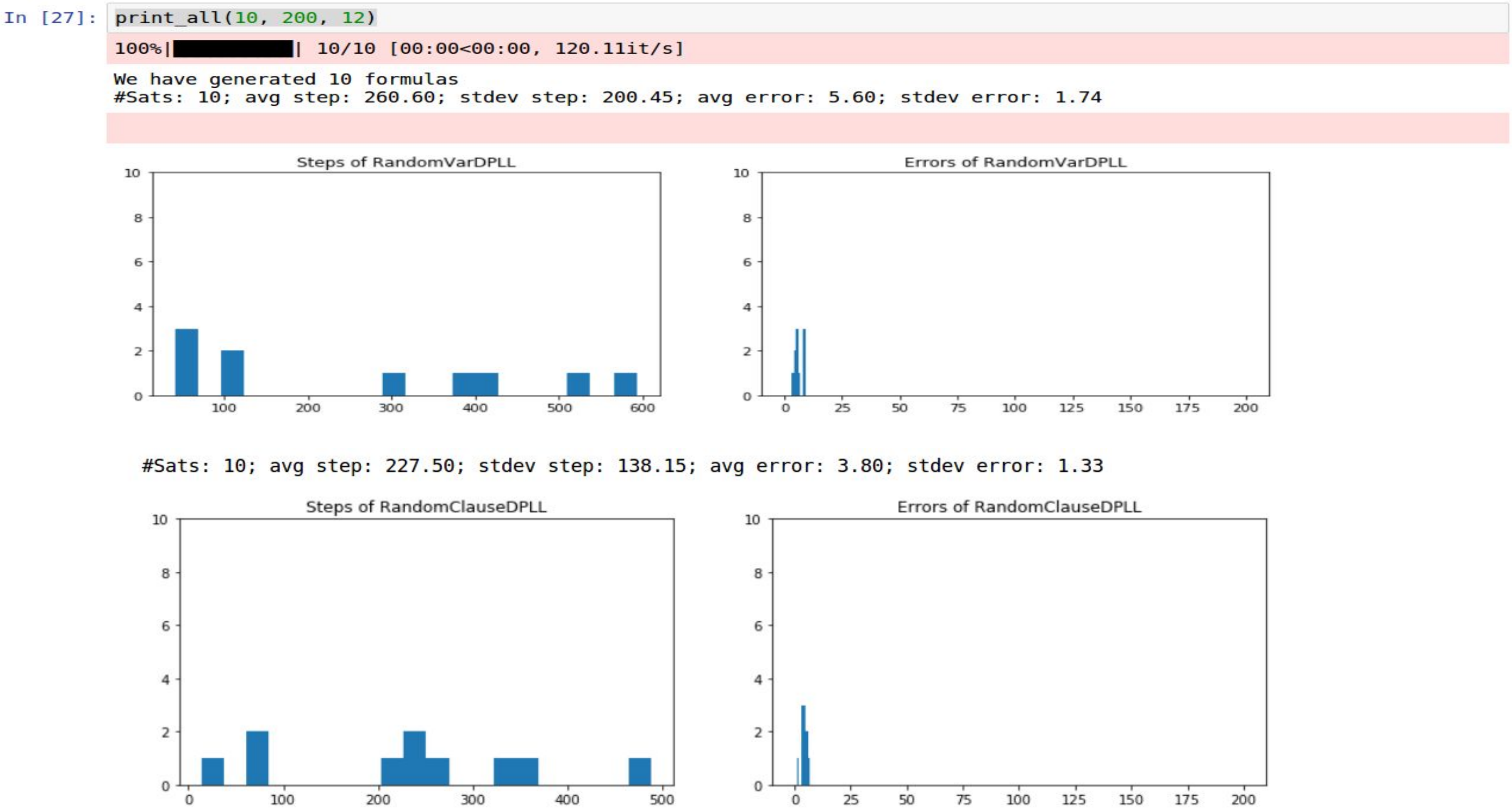
Comparison vs LSTM

#vars: 8, #clauses: up to 50

#Sats: 100; avg step: 19.73; stdev step: 19.94; avg error: 0.36; stdev error: 0.54

#Sats: 100; avg step: 9.01; stdev step: 0.10; avg error: 0.01; stdev error: 0.10





```
In [27]: print_all(10, 200, 12)
```

```
100%|██████████| 10/10 [00:00<00:00, 120.11it/s]
```

```
We have generated 10 formulas
#Sats: 10; avg step: 260.60; stdev step: 200.45; avg error: 5.60; stdev error: 1.74
```

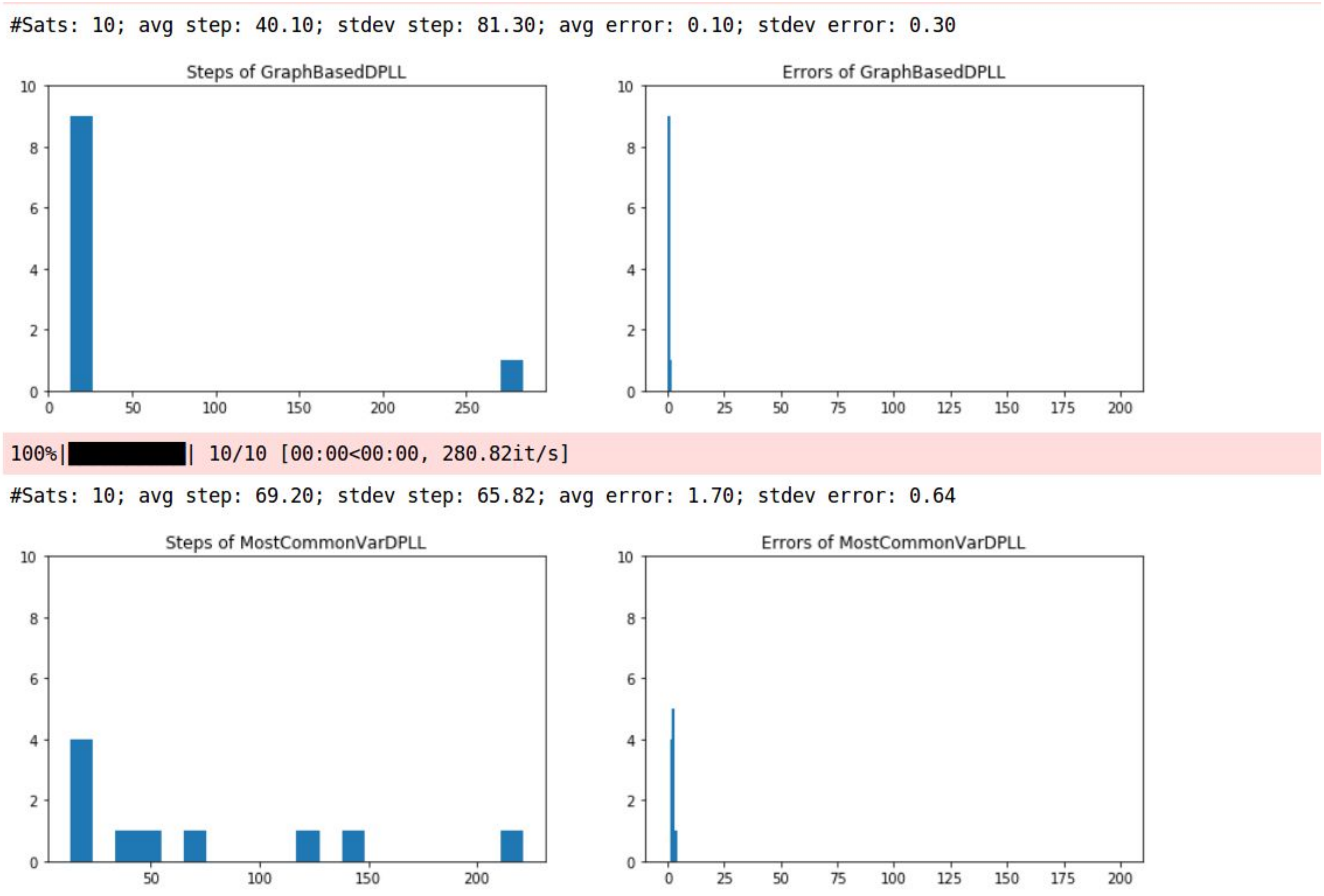
Steps of RandomVarDPLL

Errors of RandomVarDPLL

```
#Sats: 10; avg step: 227.50; stdev step: 138.15; avg error: 3.80; stdev error: 1.33
```

Steps of RandomClauseDPLL

Errors of RandomClauseDPLL



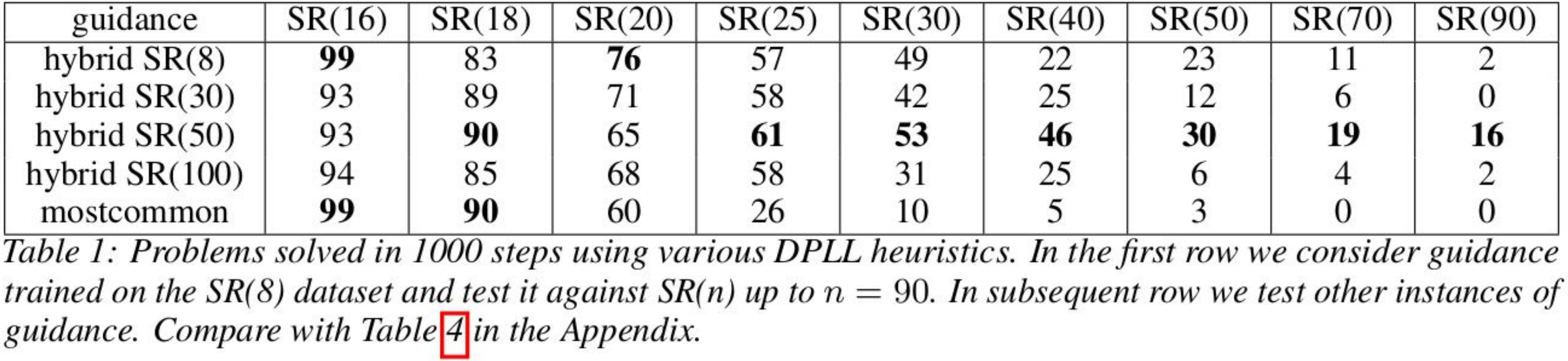
```
#Sats: 10; avg step: 40.10; stdev step: 81.30; avg error: 0.10; stdev error: 0.30
```

```
100%|██████████| 10/10 [00:00<00:00, 280.82it/s]
#Sats: 10; avg step: 69.20; stdev step: 65.82; avg error: 1.70; stdev error: 0.64
```

nodes	samples per hour	samples per node	days to 1e8
8	28693	3587	145
16	58080	3630	72
32	108672	3396	38
64	189504	2961	22
128	352683	2755	12
256	691328	2701	6

problem	Glucose 3	MiniSat 2.2
SR(10)	0.003	0.002
SR(30)	0.009	0.007
SR(50)	0.021	0.015
SR(70)	0.031	0.024
SR(90)	0.052	0.053
SR(110)	0.171	0.137
SR(130)	0.1238	0.504
SR(150)	6.091	3.406

Figure 2: Performance of our multinode training architecture tested using up to 256 servers each equipped with Xeon E5-2680v3@2,5 GHz. The days *column shows how many days of computations are required to generate and train the message passing architecture on 100 million samples drawn from the SR(100) distribution. The table on the right shows the amount of time in seconds required by Glucose 3 [AS09] and Minisat 2.2 [ES03] to solve a randomly sampled problem from SR(n) family (see Section A for more details about SR(n) problems). Solvers were accessed using the PySAT interface [IMM18].*



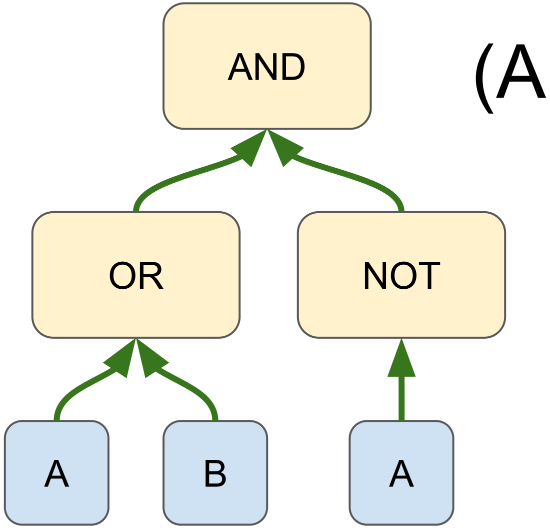
guidance	SR(16)	SR(18)	SR(20)	SR(25)	SR(30)	SR(40)	SR(50)	SR(70)	SR(90)
hybrid SR(8)	**99**	83	**76**	57	49	22	23	11	2
hybrid SR(30)	93	89	71	58	42	25	12	6	0
hybrid SR(50)	93	**90**	65	**61**	**53**	**46**	**30**	**19**	**16**
hybrid SR(100)	94	85	68	58	31	25	6	4	2
mostcommon	**99**	**90**	60	26	10	5	3	0	0

Table 1: Problems solved in 1000 steps using various DPLL heuristics. In the first row we consider guidance trained on the SR(8) dataset and test it against SR(n) up to $n = 90$. In subsequent row we test other instances of guidance. Compare with Table 4 in the Appendix.

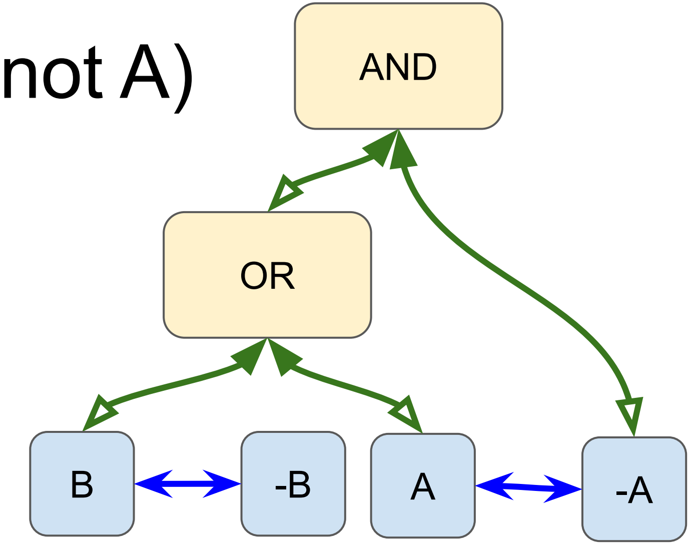
https://www.mimuw.edu.pl/~henrykm/pubs_2018/neural_guidance_sat.pdf

Generalisation of GraphSAT to non-CNF

(A or B) and (not A)

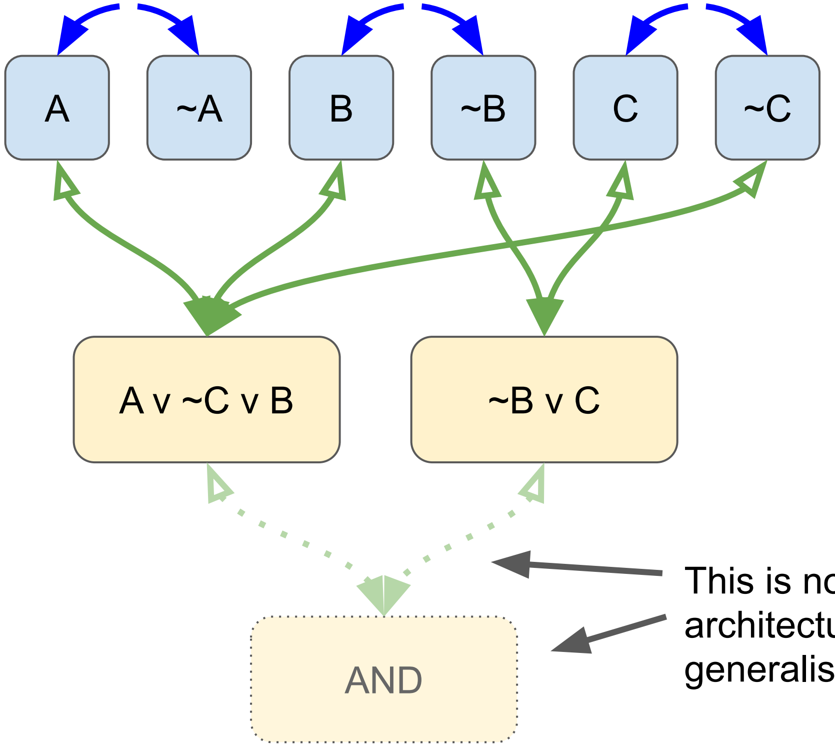


Eqnet/TreeNN:
one-way edges.



GraphSAT: two-way edges,
symmetric or asymmetric.

Current graphSAT - AND across clauses is implicit.



This is not present in current architecture, but would be in generalised architecture.