

---

# HyperTree Proof Search for Neural Theorem Proving

---

**Guillaume Lample<sup>\*†</sup>   Marie-Anne Lachaux<sup>\*†</sup>   Thibaut Lavril<sup>\*†</sup>   Xavier Martinet<sup>\*†</sup>**

**Amaury Hayat<sup>§</sup>   Gabriel Ebner<sup>‡</sup>   Aurélien Rodriguez<sup>†</sup>   Timothée Lacroix<sup>\*†</sup>**

Solves some (10) International Mathematical Olympiad problems!!

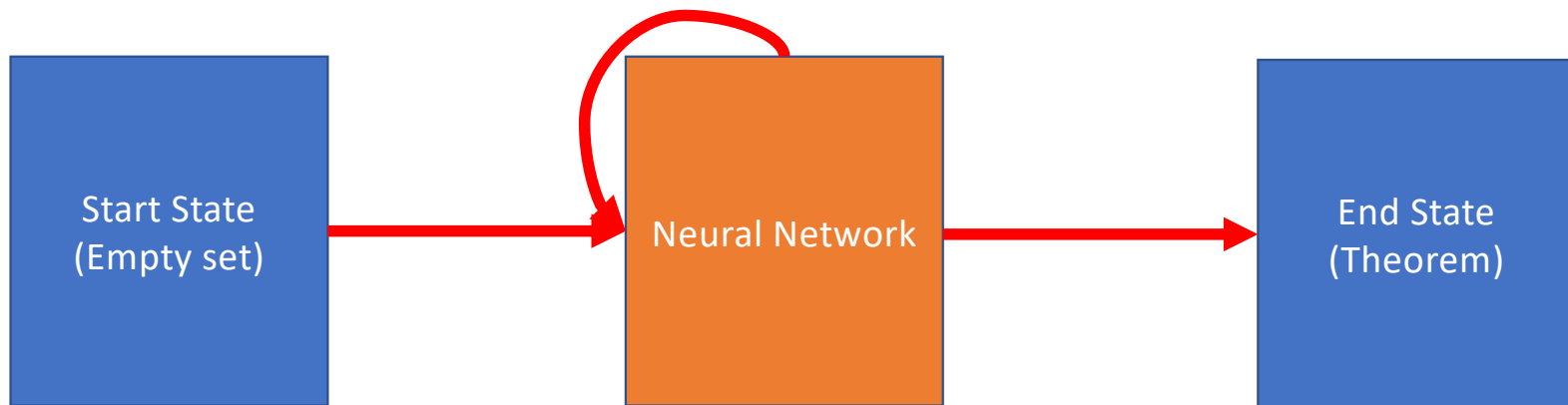
Interpreted by:  
John Tan Chong Min

# Motivation

- Theorem proving is not natural to human thinking
- Requires exploration in proof space, and tries to match a sequence of steps to reach the end goal
- Why not use an algorithm that searches smartly (e.g. AlphaZero) to do theorem proving?

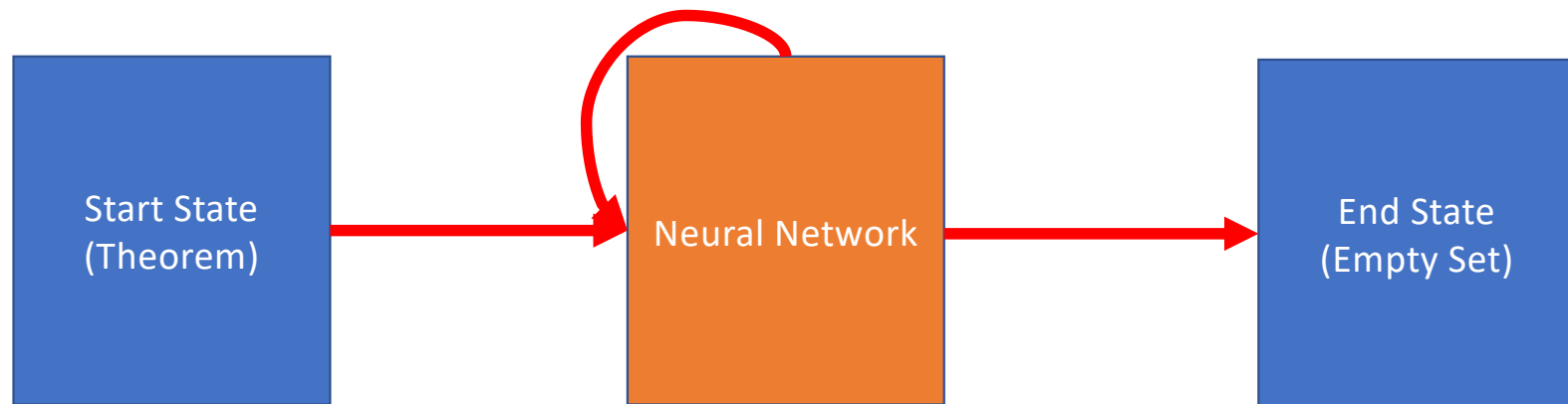
# Traditional Process

- Hard to know when theorem is reached as there could be many valid formulations of the theorem



# Backwards Process

- When the theorem has sufficiently been proved such that every sub-part is proved, we know the theorem is solved

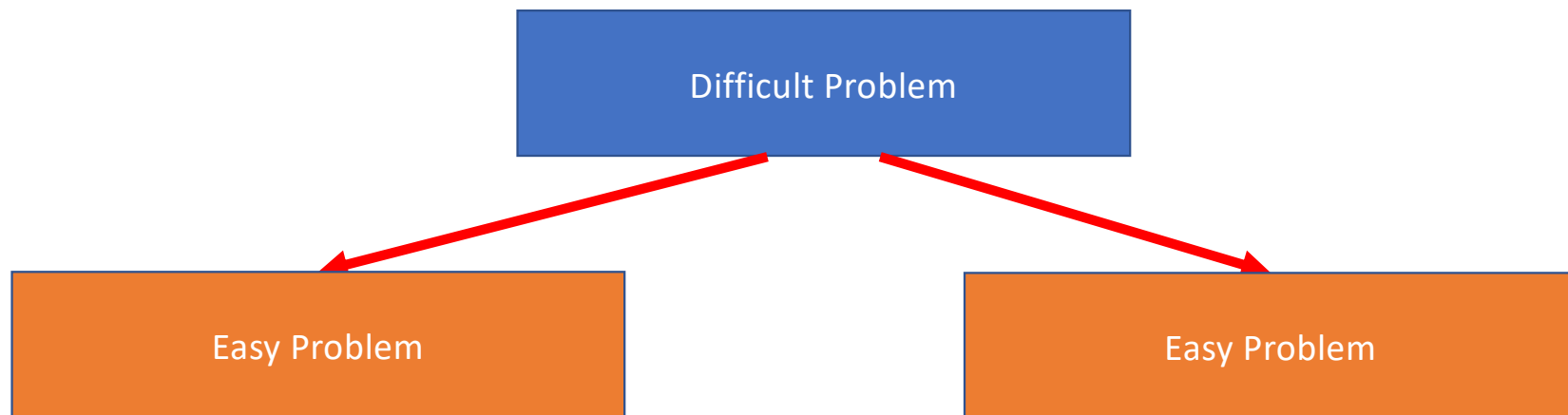


# Tactics

- Think of it as performing a function onto a particular state of the theorem, to get the next state of the theorem
- Using the tactic can make it easier or harder to solve!!
- Common tactics:
  - Intro (introduce a hypothesis)
  - Apply (apply a hypothesis to simplify the goal)
  - Assumption (solve by the assumption of the hypothesis)
  - Split (split the proof into the left and right parts)

# Insight

- Rather than solving a problem at once, break it down into sub-goals
- Make each step of incomplete proof as a node in the graph, evaluate its goodness



# Sample Theorem

```
1  import data.nat.basic
2  ∨ example (m n k : ℕ) (h₀ : n ≤ m) : n + k ≤ m + k := begin
3    induction k,
4    ∨ {
5      exact h₀
6    },
7    ∨ {
8      rw nat.succ_le_succ_iff,
9      exact k_ih
10   }
11  end
```

First subgoal :  $n + 0 \leq m + 0$

Second subgoal :  
 $n + k \leq m + k \Rightarrow n + k + 1 \leq m + k + 1$

Figure 1: A simple proof of the statement  $n \leq m \Rightarrow n + k \leq m + k$  in Lean. The *induction* tactic reduces the initial statement to two subgoals, that can be solved independently.

- First subgoal is initial hypothesis
- Second subgoal is inductive step

# Lean

- Lean is a full-fledged programming language with powerful automations for theorem proving
- Uses tactics such as ***ring*** (able to prove goals using manipulations in semirings), ***norm\_num*** (able to prove numerical goals) or ***linarith*** (able to find contradictions in a set of inequalities).

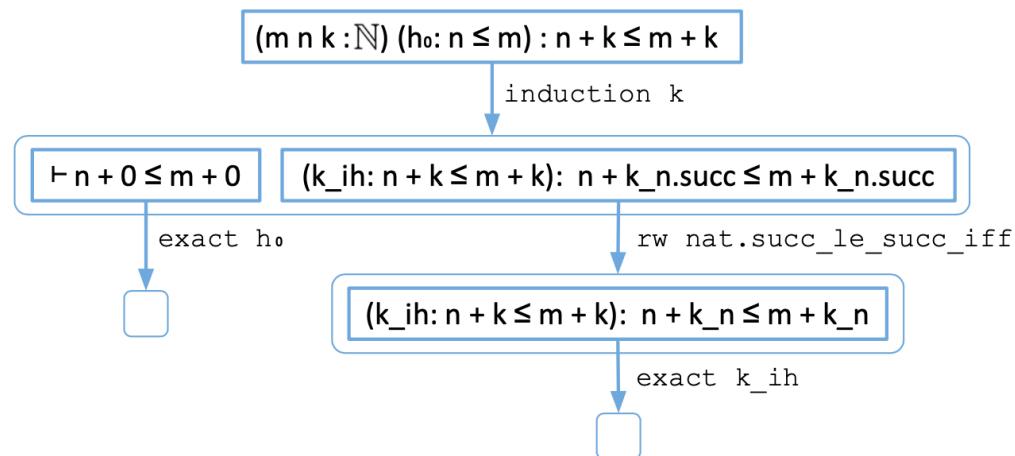
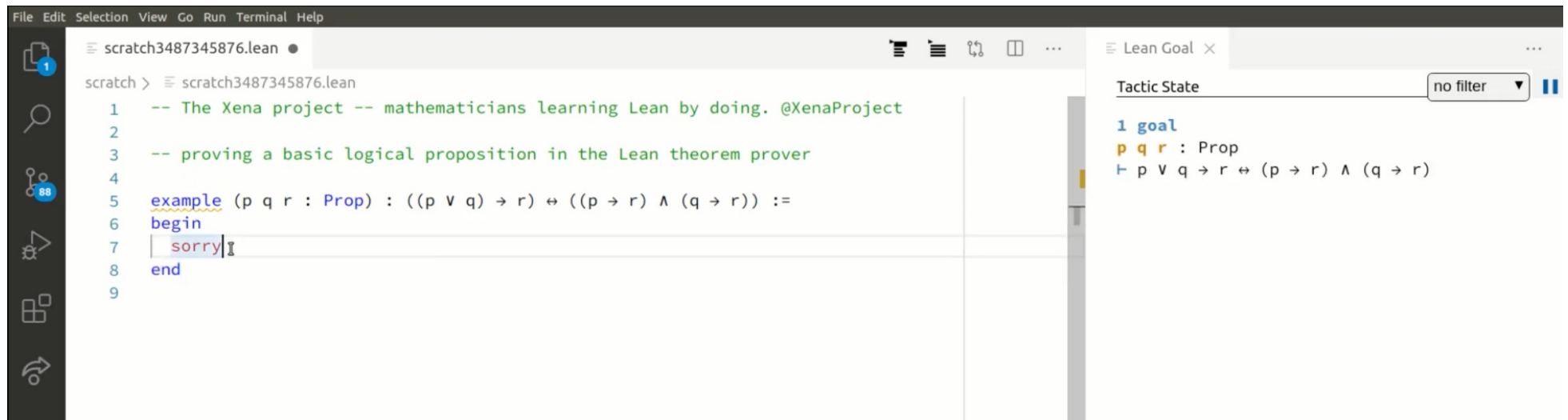


Figure 3: A visualization of the proof-tree for the proof discussed in the introduction in Lean.



# Lean Demo

- Let's see some Lean action
- <https://www.youtube.com/watch?v=POHVMMG7pqE>



The screenshot shows the Lean IDE interface. The main editor displays a Lean file named `scratch3487345876.lean` with the following content:

```
1 -- The Xena project -- mathematicians learning Lean by doing. @XenaProject
2
3 -- proving a basic logical proposition in the Lean theorem prover
4
5 example (p q r : Prop) : ((p ∨ q) → r) ↔ ((p → r) ∧ (q → r)) :=
6 begin
7   sorry
8 end
```

The cursor is positioned at the end of the `sorry` statement on line 7. On the right side, the **Tactic State** panel shows the current goal:

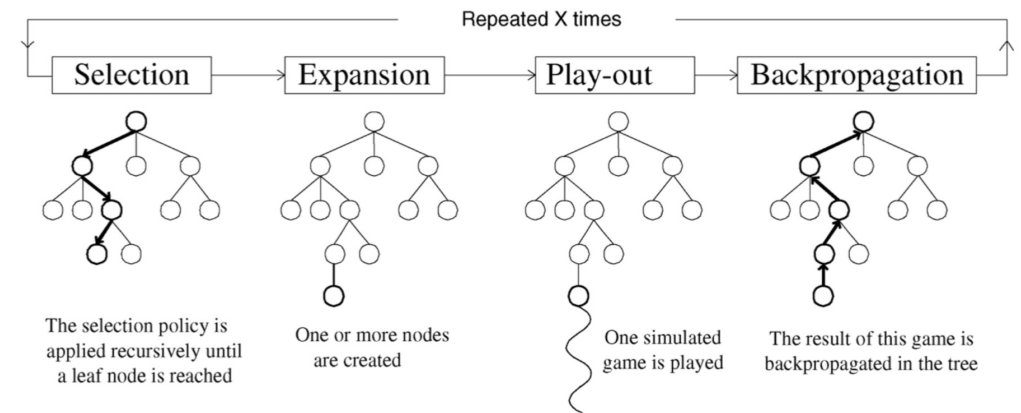
```
1 goal
p q r : Prop
⊢ p ∨ q → r ↔ (p → r) ∧ (q → r)
```

The **Tactic State** panel also includes a filter dropdown set to `no filter` and a pause button.

Recap: AlphaZero

# AlphaZero: Overview

- Uses Monte Carlo Tree Search to improve networks
- **Value Network (Exploit)**: Helps to evaluate each node to guide Monte Carlo Tree Search
- **Policy Network (Explore)**: Helps to guide Monte Carlo Tree Search to select better actions



$$a_t = \underset{a}{\operatorname{argmax}} \left( Q(s_t, a) + U(s_t, a) \right)$$

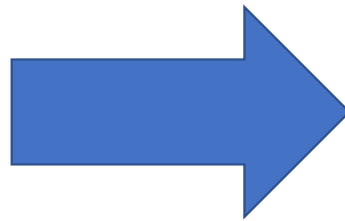
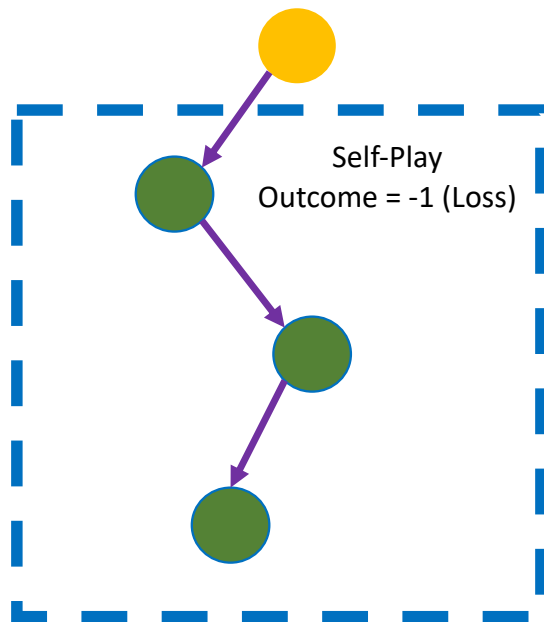
Exploit (Value Network)

Explore (Policy Network)

$$U(s, a) = c_{\text{puct}} P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)}$$

# Value Network (Depth)

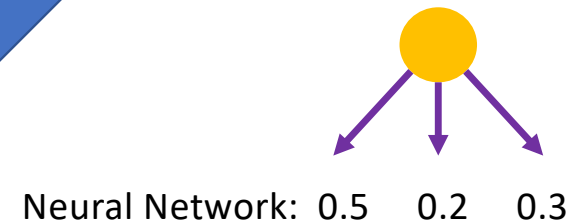
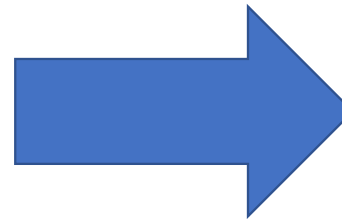
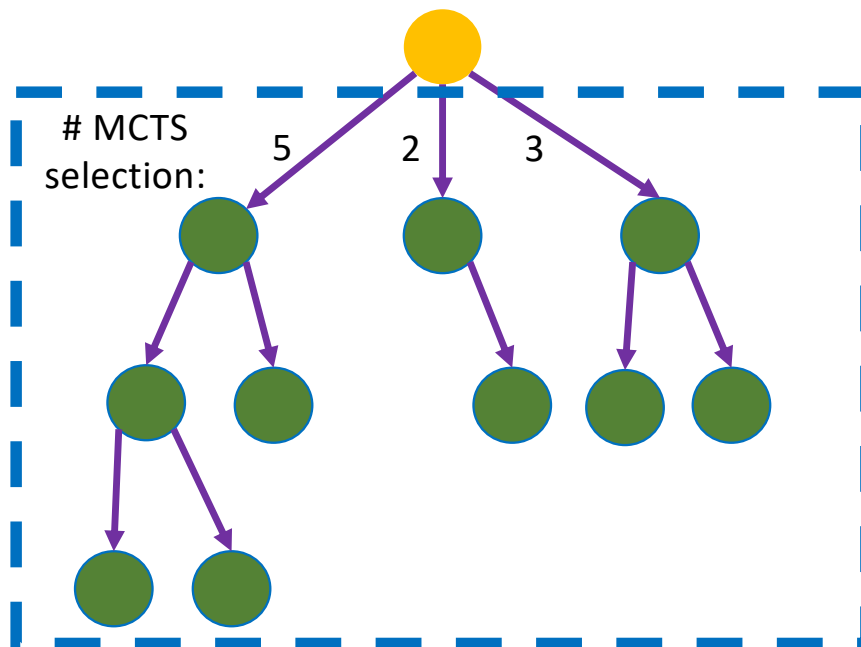
- Look ahead without simulation!
- Even more accurate than Monte Carlo rollouts



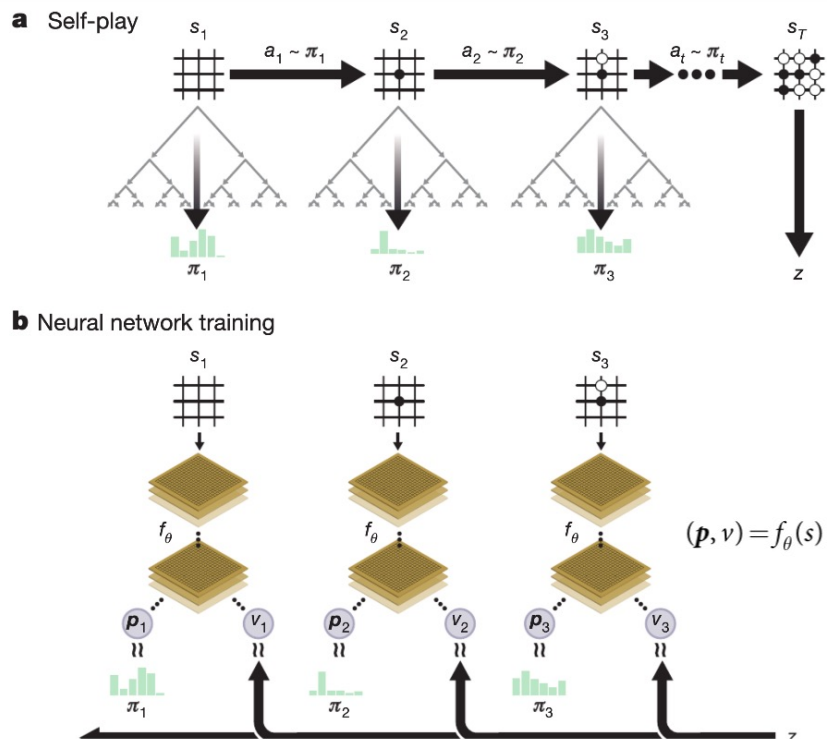
Neural Network:  
-1

# Policy Network (Breadth)

- Focus on moves with higher win rate!



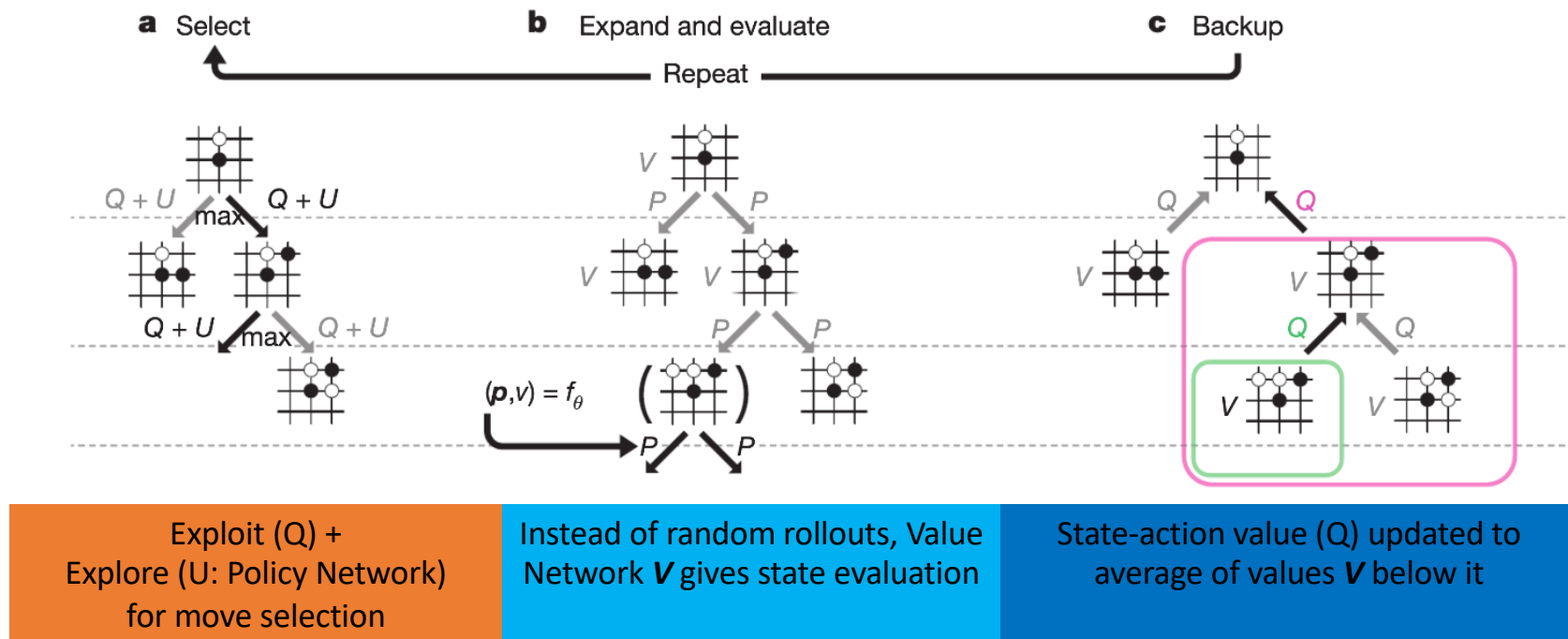
# Learning via Self Play



- Generate data using self-play games
- Train so that policy network  $\mathbf{p}$  mimics the improved policy distribution  $\boldsymbol{\pi}$  generated by MCTS
- Train so that value network output  $\mathbf{v}$  matches the game outcome  $\mathbf{z}$  for all time steps
- $c$ : L2 regularization constant (set at  $10^{-4}$ )

$$l = (z - v)^2 - \boldsymbol{\pi}^{\top} \log \mathbf{p} + c \|\boldsymbol{\theta}\|^2$$

# MCTS in AlphaZero



# HyperTree Proof Search

AlphaZero adapted to Theorem Proving

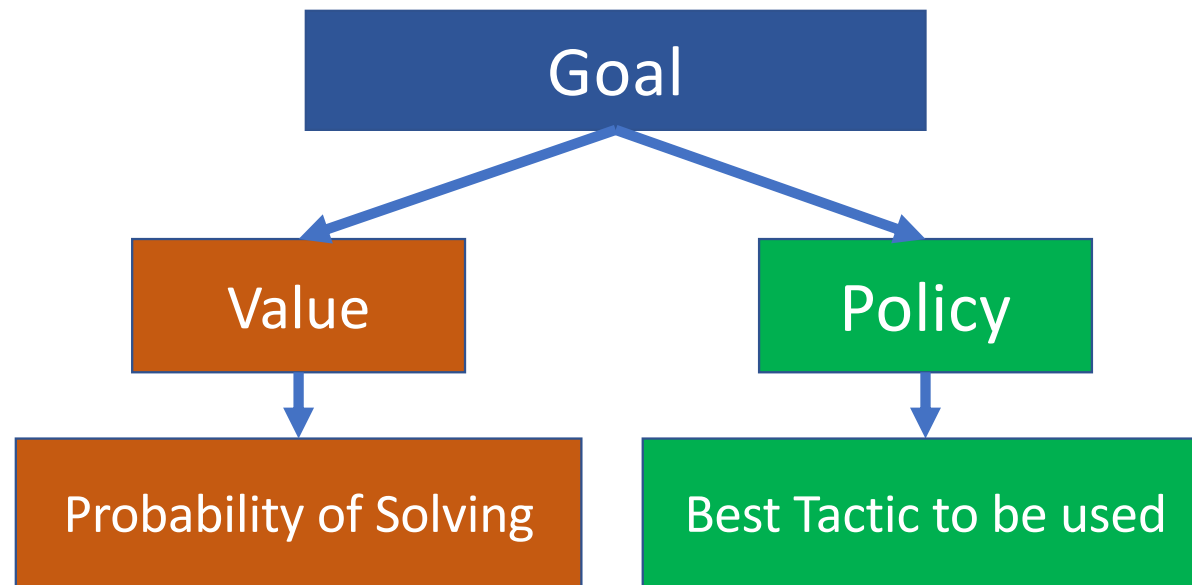


# Challenges to apply AlphaZero-like methods

- Infinite Action (Tactic) Space
  - Uses a Large Language Model to select tactics
- MCTS is not just from start node to leaf, but needs to keep track of all subgoals
  - Need to do some form of multiplicative likelihood of solving a particular goal via the likelihood of solving its subgoals
- Need to prevent cycles whereby tactics lead back to the same subgoal
  - Implement cycle checks and use virtual counts when expanding so that the virtual counts can be removed when there is a cycle

# How AlphaZero is applied for Theorem Proving

- Value Network: How likely a particular goal/subgoal can be solved
- Policy Network: What tactic to use next for a goal/subgoal



# Hypergraph

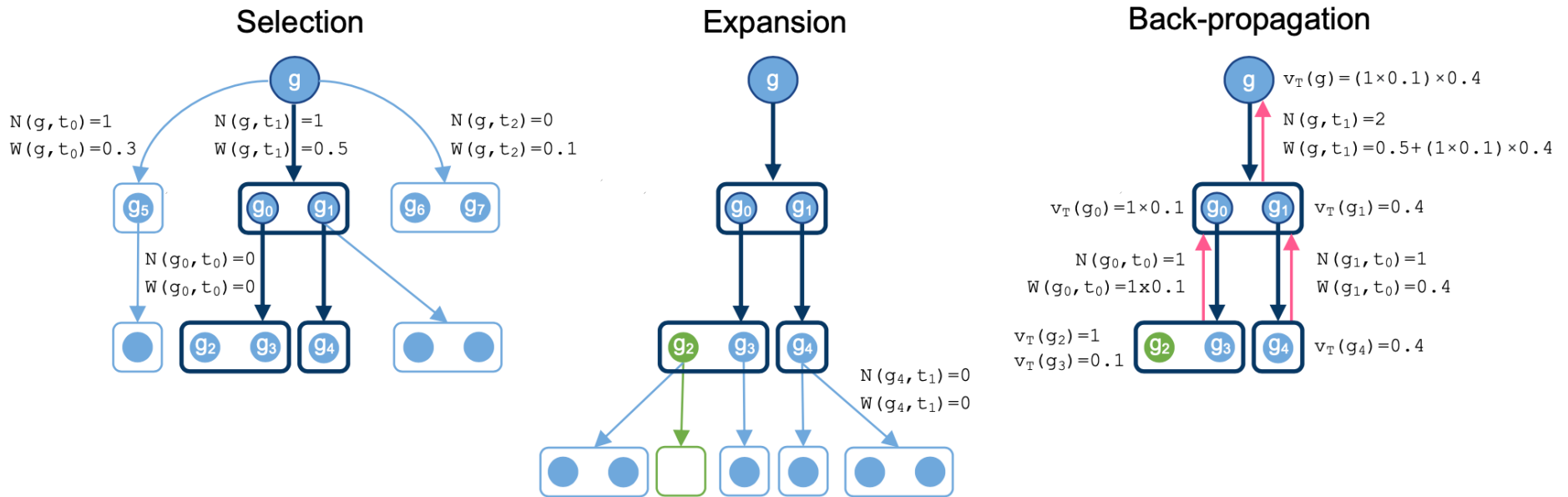
- Start with a hypergraph with:
  - Theorem to be proved as root node
  - Tactics as edges
  - Subgoals as nodes
- Recursively expands leaf nodes
  - Apply tactic to the leaf node
  - Results in new subgoal(s)
- Proof is a hypertree rooted in initial theorem whose leaf nodes are all empty sets (solved)!!

# HyperTree Proof Search Process

Selection

$$\vdash \cos(x) + e^x < 1 + 2e^x$$

# HyperTree Proof Search



**Figure 5: HyperTree Proof Search.** We aim at finding a proof of the root theorem  $g$  with HTPS. Proving either  $\{g_5\}$ ,  $\{g_0, g_1\}$ , or  $\{g_6, g_7\}$  would lead to a proof of  $g$  by tactic  $t_0, t_1$ , or  $t_2$ . The figure represents the three steps of HTPS that are repeated until a proof is found. Guided by the search policy, we select a hypertree whose leaves are unexpanded nodes. The selected nodes are then expanded, adding new tactics and nodes to the hypergraph. Finally, during back-propagation we evaluate the node values of the hypertree, starting from the leaves back to the root, and update the visit counts and total action values.

# Overall Procedure

---

**Algorithm 1** Finding an expandable subtree

---

**Input:** A hypergraph  $H$  and its  $root$

**Output:** A partial proof tree with unexpanded leaves

*:start*

T: hypertree( $root$ )

to\_explore: list = [ $root$ ]

**while** to\_explore **do**

$g = \text{to\_explore.pop}()$

**if**  $g$  is internal **then**

**if**  $\text{expandable}(g) \neq \emptyset$  **then**

$tactic = \arg \max_t \pi|_{\text{expandable}(g)}(g, t)$

**else**

            continue { expandable nodes are in a sibling branch }

**end if**

**if** tactic leads to cycle **then**

            kill tactic

            remove virtual counts for elements of T

            goto *start*

**end if**

$VC(g, tactic) += 1$

        T.add( $g$ , tactic, children( $g$ , tactic))

        to\_explore += [children( $g$ , tactic)]

**end if**


**end while**

---

- Tries to recursively expand the tree until it is fully solved
- Ends when all subgoals solved or max number of simulations reached
- Virtual counts is something like a “try” block in programming
  - Allows for elimination of entire counts of an unwanted tactic

# Selection

Given the visit count  $N$ , the total counts  $C$ , value estimates  $Q$ , the model prior  $P_\theta$  and an exploration constant  $c$ . The policy used in Alpha-Zero is PUCT:

$$\text{PUCT}(g) = \arg \max_{t \in \mathcal{A}} \left[ \boxed{Q(g, t)} + c \cdot \boxed{P_\theta(t|g) \cdot \frac{\sqrt{\sum N(g, \cdot)}}{1 + C(g, t)}} \right]$$


The diagram shows the PUCT formula with two terms inside brackets. The first term,  $Q(g, t)$ , is highlighted in a yellow box. A blue arrow points from this box to a yellow box labeled "Exploit (Value Network)". The second term,  $c \cdot P_\theta(t|g) \cdot \frac{\sqrt{\sum N(g, \cdot)}}{1 + C(g, t)}$ , is highlighted in a green box. A blue arrow points from this box to a green box labeled "Explore (Policy Network)".

$$Q(g, t) = \frac{W(g, t)}{N(g, t)}$$

- $C(.) = N(.)$  (consolidated counts) +  $VC(.)$  (virtual counts)
- $Q$  is the averaged value across number of times node has been expanded

# Expansion

- The policy model produces  $B$  tactics for an unexpanded node  $g$ 
  - $B$  uniformly distributed in  $[8, 16, 32, 48]$
- These tactics are evaluated in the proving environments to generate the resultant sub-goals
- Only valid tactics are added to hypergraph
- Value of node  $v_T(g)$ :
  - If no tactics are valid, the node is marked as invalid (value 0)
  - If a tactic solves  $g$ , the node is marked as solved (value 1)
  - Otherwise value is estimated by value network



# Backpropagation

---

**Algorithm 2** Back-propagation of total action value  $W$ 

---

**Input:** Partial proof-tree  $T$  and value estimates  $c_\theta(g)$  of its leaves.

to\_backup = []

**for**  $g$  in leaves of  $T$  **do**

$v_T(g) = c_\theta(g)$

    to\_backup.append(parent $_T(g)$ )

**end for**

**while** to\_backup **do**

$g = \text{to\_backup.pop}()$

    to\_update =  $\prod_{c \in \text{children}_T(g)} v_T(c)$

$W(g, t) += \text{to\_update}$

$N(g, t) += 1$

$VC(g, t) -= 1$

$v(g, t) = \text{to\_update}$

$g.\text{is\_prop} = \text{true}$

**if** all  $c.\text{is\_prop}$  for  $c$  in siblings $_T(g)$  **then**

        to\_backup.append(parent $_T(g)$ )

**end if**

**end while**

---

- Backup the expanded tree from the leaves to the root:
  - Updates  $W$  to accumulated likelihood of solving node
  - Updates  $N$  by 1 for each child expanded
- Removes the added virtual count if there is no cycle formed after applying this tactic
  - Becomes consolidated into the actual count

# Backpropagation

- If a branch is solved (value of current backpropagation step is 1), it will be marked as solved and ignored for subsequent expansion steps
- If all branches are solved, then theorem is solved!!
- Note value of node is given as the multiplicative value of all its sub-goal nodes after applying a tactic. Value is interpreted as probability of solving:

$$v_T(g) = \prod_{c \in \text{children}(g,t)} v_T(c)$$

# Model Training

# Training

- Policy and value networks are both the same Transformer model!
  - **Policy Network:** Takes as input a tokenized goal and generates tactics. It is trained with a standard seq2seq objective, where we minimize the cross-entropy loss of predicted tactic tokens conditioned on an input goal
  - **Value Network:** Takes as input a tokenized goal and restrict valid output tokens to either PROVABLE or UNPROVABLE. Similar to cross-entropy loss for classification

# Three Phases of Training

- **Pre-training:** Model pre-trained with a masked seq2seq objective on the LaTeX source code of papers from the mathematical section of arXiv
- **Fine-tuning:** Model fine-tuned with a supervised dataset from the Mathlib library. The training set is composed of 24k theorems and 144k goal-tactic pairs.
- **Online training:** The shortest proof is taken after each successful proof search is used to train model in a supervised manner similar to fine-tuning

# Difference from AlphaZero

- **Not using the MCTS-like search to update the Policy/Value networks!!**
  - Probably because the Value Network is multiplicative and the resultant value can be too low and too varied to train effectively (perhaps log likelihood is better if we want to train using the backpropagation step?)
  - AlphaZero with MCTS-like policy improvements can suffer from bootstrapping errors
- Not doing a search over discrete actions
  - Infinitely many actions
  - Use a large language model instead

# Results

Online training statements	Supervised -	GPT-f	Evariste-1d miniF2F-curriculum	Evariste-7d	Evariste miniF2F-valid
miniF2F-valid	38.5	47.3	46.7	47.5	58.6 <sup>†</sup>
miniF2F-test	35.3	36.6	38.9	40.6	41.0
miniF2F-curriculum	20.8	30.6	33.6 <sup>†</sup>	42.5 <sup>†</sup>	32.1
Train time (A100 days)	50	2000	230	1620	1360

Table 2: **Pass rate on Lean environment using 64 trials (pass@64).** Numbers with a <sup>†</sup> exponent correspond to the cumulative pass-rate since the evaluated statements are part of the online training. Evariste refers to the method described in this paper.

# Solves (some) IMO problems!

```
1 theorem imo_1964_p1_2 (n : ℕ) : ¬7|2n + 1 :=
2 begin
3   rw nat.dvd_iff_mod_eq_zero,
4   rewrite [nat.add_mod, nat.mod_eq_of_lt],
5   obviously,
6   apply nat.strong_induction_on n,
7   induction n,
8   {
9     intros n IH,
10    cases n,
11    norm_num,
12    cases n,
13    norm_num,
14    rw [nat.succ_eq_add_one, pow_succ],
15    rw [nat.succ_eq_add_one, pow_succ],
16    induction n,
17    norm_num,
18    rw [nat.succ_eq_add_one, pow_succ],
19    norm_num [nat.mul_mod, ←mul_assoc],
20    contrapose! IH,
21    refine ⟨n_n, nat.lt_succ_iff.mpr _, IH⟩,
22    exact nat.le_succ_of_le (nat.le_succ _),
23  },
24  exact n_ih,
25 end
```

Figure 13: A proof of the imo\_1964\_p1\_2 problem found by our model. The model shows that for any value of  $n \in \mathbb{N}$ ,  $2^n + 1$  is not divisible by 7, by showing that  $2^n \bmod 7 + 1 \neq 0$ , and  $2^n \bmod 7 + 1 < 7$ . The second part of the proof uses strong induction and the fact that  $2^n \equiv 2^{n+3} \pmod{7}$ . We provide a version of the proof that was automatically cleaned by removing unnecessary tactics and tactic arguments.



# Discussion

- There are infinitely many tactics for theorem proving. AlphaZero works best in a constrained fixed environment. How can we make the tactic space more amenable for an AlphaZero-style network?
  - Should Large Language Models be used like in this paper?
  - Could hierarchical problem solving be used?
  - Could supervised learning help in classifying the type of solution required?
- How do we get the shortest solution? In AlphaTensor there was a negative reward for each extra timestep to incentivize the agent to learn the shortest solution, how can it be done in HyperTree Proof Search?
- Regime of big data: Should we generate more synthetic datasets for training just like for AlphaTensor?

# Discussion

- Should we only do backward search from the theorem to be proved?  
How about a combination of both forward and backward search?
- Do we think in mathematical terms? Is it worthwhile to push for this line of automated theorem proving to solve generic problems?
- What are the mental faculties used to perform normal activities in life? Are we logical in nature?
- What is the benefit/disadvantage of logical thinking in real life?