LIME: Learning Inductive Bias for Primitives of Mathematical Reasoning

Yuhuai Wu, Markus Rabe, Wenda Li, Jimmy Ba, Roger Grosse, Christian Szegedy

University of Toronto, Vector Institute, Google Research

Contributions

- Providing the first method to design inductive biases in the form of datasets for mathematical reasoning.
- Demonstrating significant improvements in the reasoning performance of transformer models on four large mathematical reasoning benchmarks with negligible extra computation cost.
- By showing how pretraining brings benefits other than learning content knowledge, disentangling the study of its working mechanism.

Peirce's Reasoning Primitives

Inspired by the logician Charles Peirce, we consider the following three reasoning primitives: deduction, abduction, and induction.

- **Deduction**: the ability to deduce new truths from premise and inference rules.
- **Induction**: the ability to induce general inference rules from known facts.
- **Abduction**: the ability to explain the relationship between the evidences and inference rules.

A simplistic view with Rule, Case and Result

Reasoning Primitives	Inference Map
Deduction	Rule, Case → Result
Abduction	Rule, Result → Case
Induction	Case, Result → Rule

Deduction

Rule: All the beans in this bag are white.
Case: These beans are from this bag.
Result:These beans are white.

Abduction

Rule: All the beans in this bag are white.
Result:These beans are white.
Case: These beans are from this bag.

Induction

Case: These beans are from this bag.
Result:These beans are white.
Rule: All the beans in this bag are white.

LIME: Synthetic Tasks For Reasoning Primitives

We design three sequence to sequence synthetic tasks inspired by the three reasoning primitives. **The idea is to pretrain transformer networks on these synthetic tasks for learning inductive biases of reasoning.**

- `Deduct`: **Source:** Rule string and Case dictionary.
  Target: Result string.
- `Abduct`: **Source:** Rule string and Result string.
  Target: Case dictionary.
- `Induct`: **Source:** Case dictionary and Result string.
  Target: Rule string.

In the following, we describe one simple way to generate those three elements, though we acknowledge that there are many other possible approaches.

Rule, Case and Result

We require two types of symbols: 1. *math symbols*, 2. *rule symbols*. In general, these symbols can take any forms (e.g., integer representations). We now construct Rule, Case, and Result in order:

1. **Rule** is a randomly sampled string that consists of i) rule symbols and ii) math symbols.
2. **Case** is a dictionary that represents substitutions. For each rule symbol used in the Rule string, we sample a random string of random length that consists of math symbols. This forms a dictionary, whose keys are all rule symbols, and the values are the corresponding sampled string.
3. **Result** is the outcome of the substitution. For each rule symbol in the Rule string, we replace it with the corresponding value stored in the Case dictionary. This gives rise to the Result string.

An example of Rule, Case and Result

Rule: `A + B = B + A`

Case: `{A: "xy$%2", B: "8djwh4"}`

Result: `xy$%2 + 8djwh4 = 8djwh4 + xy$%2`

See more variants of the tasks in the paper.

Experimental Protocol

Benchmarks We have selected four tasks to cover various different styles of interactive theorem provers: The HOL-Light (skip-tree) corpus was created from very high-level tactic-based proofs, but it is less interpretable than IsarStep's declarative style corpus. Lean corpus is based on dependent type theory, and is one of the most popular theorem provers. We also evaluate the next proof-step prediction task on the `set.mm` library of MetaMath, which consists of very granular, basic proof steps.

LIME Pretraining We generate datasets of our synthetic tasks for pretraining: `Deduct`, `Abduct`, `Induct`, `Mix`. We use 44 math symbols and 24 rule symbols. The length of the Rule string is sampled from 5 to 20, the length of the string for each substitution (the values of Case dictionary) is sampled from 2 to 8. We used word-level tokenization for all the tasks. We pretrained the model for 20K updates. We used the Adam optimizer with learning rate $3 \cdot 10^{-4}$. We used a dropout rate of 0.1 and label smoothing with a coefficient 0.1.

Fine-tuning For all the downstream tasks in this section, when loading the pretrained models for fine-tuning, we do not load in the vocabulary embeddings nor the output layer weights. We set the maximum number of tokens in a batch to 4096, and accumulated four batches of gradients for one parameter update. We trained the model for 200K updates. We used the Adam optimizer, and we searched over the learning rates $\{3 \cdot 10^{-4}, 7 \cdot 10^{-4}\}$, and warmup steps $\{4000, 8000\}$. We used a dropout rate of 0.1 and label smoothing with a coefficient 0.1.

Architecture All experiments used the transformer base model, i.e. 512 hidden size, 2048 filter size, 8 attention heads, 6 layers for both the encoder and decoder.

Results on Four Benchmarks

Table: Test top-8 Accuracy on Skip-Tree HOList (%).

Model	Equation completion	Hard type inference	Missing assumptions	Easy type inference
No pretrain	46.3	95.0	41.8	95.9
LIME `Deduct`	50.3	94.8	**47.9**	97.0
LIME `Abduct`	48.4	94.8	46.1	96.3
LIME `Induct`	44.8	94.9	42.6	96.4
LIME `Mix`	**51.7**	**95.6**	46.1	**97.6**

(a) Test top-1, top-10 (%) accuracy on the IsarStep task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain	20.4	33.1
HAT	22.8	35.2
LIME `Deduct`	24.7	37.7
LIME `Abduct`	26.7	**41.0**
LIME `Induct`	23.9	38.8
LIME `Mix`	**26.9**	40.4

(b) Test top-1, top-10 (%) accuracy on the LeanStep unseen lemma prediction task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain	15.8	27.4
LIME `Deduct`	25.8	38.0
LIME `Abduct`	26.0	38.6
LIME `Induct`	25.0	38.2
LIME `Mix`	**29.8**	**41.8**

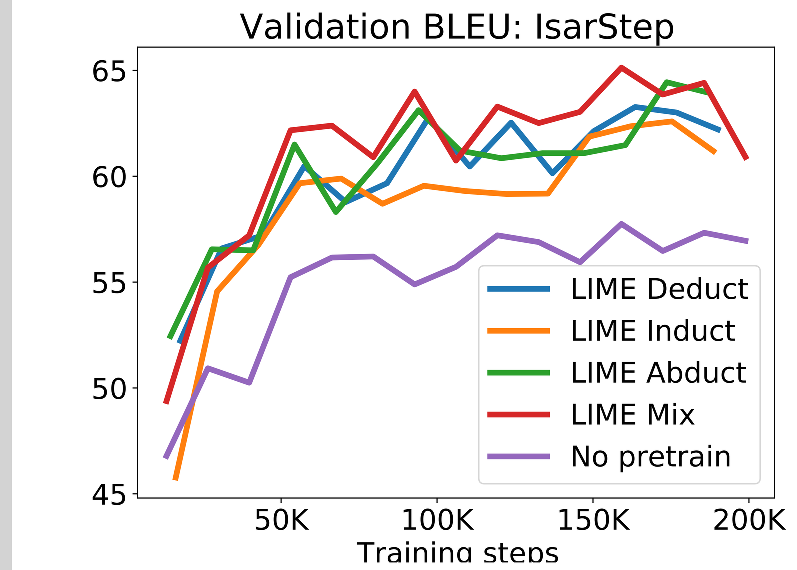

Figure: Validation BLEU along training on the IsarStep task.

Table: Test top-1, top-10 (%) accuracy on the MetaMathStep task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain	67.7	76.5
LIME `Deduct`	68.8	77.4
LIME `Abduct`	68.8	76.1
LIME `Induct`	**69.9**	**78.0**
LIME `Mix`	69.1	77.9

We observed a huge gain with LIME pretraining across all four benchmarks.

Ablation Studies

(a) Comparisons to other pretraining tasks on IsarStep task.

Model	Top-1	Top-10
No pretrain	20.4	33.1
LIME `Mix`	26.9	40.4
Pretrain on MetaMathStep	23.1	35.7
Pretrain on WMT En-De	17.2	30.3

(b) Comparing LIME's benefits on LSTMs on the IsarStep Task

Model	Top-1	Top-10
LSTM	5.5	11.3
LSTM + LIME `Abduct`	6.9	14.3
LSTM + attention	12.3	22.7
LSTM + attention + LIME `Abduct`	13.4	26.3
Transformer	20.4	33.1
Transformer + LIME `Abduct`	26.7	41.0

Pretraining on Formal Reasoning and Natural Language Tasks We observe that pretraining on other tasks does not provide as much improvement as provided by pretraining on LIME tasks.

Does LIME help LSTMs? We observe that the benefits of LIME for LSTM was shown less than transformer. We hypothesize this is due to transformer's malleable self-attention architecture which allows it to learn inductive biases during pretraining time.