Synthetic Proof Term Data Augmentation for Theorem Proving with Language Models

Joseph Palermo[1], Johnny Ye[1], and Jesse Michael Han[2]

[1] Cash App Labs
[2] University of Pittsburgh*

Introduction

Imitation learning for the task of formal theorem proving is bottlenecked by the limited size of existing libraries of formalized mathematics (e.g. mathlib [1]). Prior work utilizing language models for theorem proving indicates that training data limitations are causing performance saturation [2, 3]. We propose using samples from trained language models in conjunction with the Lean kernel [4] to generate novel training examples. Other prior work has demonstrated the utility of synthetic data for learning theorem proving [5, 6, 7, 8, 9]. We train language models to generate Lean proof terms unconditioned by any theorem statement, then sample from these models to generate collections of proof term candidates. Using the Lean kernel to identify type-correct proof term candidates and infer corresponding types, we create novel training examples for proof term language modeling. Augmenting existing datasets with synthetic examples is shown to improve the performance of proof term language modeling on a held-out test set.

Bootstrap Datasets

In order to use trained language models to generate new candidate training examples, we first create a "bootstrap" dataset from which a model can be trained. Our examples for unconditioned proof term language modelling (which we call unconditioned examples) have the form: PROOF <proof> EOT. They are unconditioned in the sense that they are not conditioned on a theorem statement. We call the dataset comprising these examples: **unconditioned bootstrap**. This dataset is used to train the model that is sampled to produce synthetic examples. It is also used to train a baseline model and to provide examples for a held-out test set.

By contrast, our examples for theorem-conditioned proof term language modelling (which we call conditioned examples) have the form: THEOREM <theorem> PROOF <proof> EOT. We call the dataset comprising these examples: **conditioned bootstrap**. This dataset is used only to train a baseline model and to provide examples for a held-out test set.

As a data augmentation strategy, we perform tree traversal on each expression in mathlib to identify unique sub-expressions which are also valid proofs. We convert these filtered sub-expressions into pretty-printed text format and filter for length less than 2048 characters. We parse and type-check these proofs to ensure that pretty-printing has not rendered them invalid and to obtain the corresponding theorem statement.

*work for this project was completed while at OpenAI

Splitting the Bootstrap Datasets

We split our bootstrap datasets into train, validation and test sets, firstly by splitting on mathlib declaration names. We also apply a further filter to reduce the maximum similarity of training examples to validation and test examples. Using TF-IDF [10] embeddings of our examples we remove any validation or test example (x_{test}) for which:

$$\frac{levenstein\_distance(x_{test}, \arg\max_{x_{train}} cosine\_similarity(x_{test}, x_{train}))}{length(x_{test})} < 0.15 \quad (1)$$

The threshold value of 0.15 was determined after we found empirically that it enabled us to overfit our training data. Our initial split of declaration names produced 207,194 train examples, 55,470 validation examples and 54,964 test examples. After the additional filtering step, 11,145 validation examples and 10,233 test examples remain.

Creating Synthetic Examples

To generate proof candidates, we sample language models trained on the unconditioned bootstrap dataset. We parse and type-check proof candidates, and if type-check is successful we serialize the corresponding type. This process produces synthetic training examples of both the conditioned and unconditioned variety. We also generate an additional example from each proof candidate regardless of whether or not it has passed type-checking: NON_TYPE_CHECKED_PROOF <non_type_checked_proof> EOT. These examples are useful in assessing the effect type-check filtering has on data quality.

Bootstrap Training Sets vs. Augmented Training Sets

For these experiments, we train language models using Fairseq [11]. We utilize Fairseq's implementation of GPT-2 [12] with approximately 2 billion parameters (the so-called "big" size). We also utilize Fairseq's implementation of the "gpt2" byte pair encoder. We set max-tokens to 1536, use SGD with a fixed learning rate of 0.01, set early-stopping patience to 100 epochs, and set dropout to 0.1.

After training a model on the unconditioned bootstrap dataset, we use the trained model to sample 20 million proof candidates using beam search. We set the beam search temperature to 1.3 and beam width to 5. From the set of candidates, 1.57 % or 352,469 unique proofs passed type-check.

We create augmented datasets by randomly sampling synthetic examples without replacement and adding them to the bootstrap datasets. Samples are added until the augmented dataset in question is 100% larger than the corresponding bootstrap dataset as measured by the number of examples in the conditioned case and by the number of characters in the unconditioned case. We weight the additional unconditioned examples by counting characters because the synthetic unconditioned examples can include both type-checked and non-type-checked proofs, and the average length of non-type-checked proofs tends to be longer (162 characters vs 275 characters on average).

We create 4 distinct augmented datasets by utilizing different combinations of synthetic examples:

- **conditioned augmented**: conditioned bootstrap +100% synthetic conditioned (weighted by # of additional examples)
- **unconditioned augmented (non-type-checked)**: unconditioned bootstrap +100% synthetic unconditioned non-type-checked (weighted by # of additional characters)
- **unconditioned augmented (50/50 type-correct & non-type-checked)**: unconditioned bootstrap +50% synthetic unconditioned non-type-checked and +50% synthetic unconditioned type-correct (weighted by # of additional characters)
- **unconditioned augmented (fully type-correct)**: unconditioned bootstrap +100% synthetic unconditioned type-correct (weighted by # of additional characters)

We use each of the 2 bootstrap and the 4 augmented datasets to train language models. Then we evaluate each of these 6 models on our held-out bootstrap test sets, matching models trained on conditioned or unconditioned examples to the conditioned or unconditioned test sets respectively. When evaluating models trained on conditioned examples we prompt the models with theorem statements.

Training Dataset	Test Loss	Test Ppl.	Test Accuracy
conditioned bootstrap	1.25	2.38	9.72%
conditioned augmented	**1.12**	**2.18**	**16.92**%
unconditioned bootstrap	1.74	3.35	N/A
unconditioned augmented (non-type-checked)	1.72	3.30	N/A
unconditioned augmented (50/50 type-correct & non-...)	1.71	3.28	N/A
unconditioned augmented (fully type-correct)	**1.70**	**3.25**	N/A

Table 1: Test loss, test perplexity, and test accuracy of the models trained on each dataset. Test accuracy measures the % of test examples for which the generated proof matches ground truth.

We find that training on the augmented datasets results in superior metrics on our test sets. In the unconditioned case we also find that better metrics are achieved by using training sets in which a higher percentage of the synthetic data is type-correct. This demonstrates the improvement in data quality afforded by using the Lean kernel as a filter. However, since only a small percentage of synthetic proofs pass type-check, in practice we can likely expect the best possible unconditioned language modelling metrics to be achieved by simply training on all generated examples, as such a dataset would be much larger.

Can Increased Regularization Explain the Performance Boost?

We investigate how much of the improvement in loss associated with training on an augmented dataset is accounted for by an increase in regularization that can be achieved with dropout. To do this we train models on the conditioned bootstrap dataset and the conditioned augmented dataset with successively higher levels of dropout (incrementing by 0.1), until increasing dropout no longer improves the best achieved validation loss.

Training Dataset	Metric	Dropout: 0.1	Dropout: 0.2	Dropout: 0.3	Dropout: 0.4
conditioned bootstrap	Loss	1.25	1.15	1.09	1.09
conditioned augmented	Loss	1.12	1.04	**1.01**	1.01
conditioned bootstrap	Perplexity	2.38	2.21	2.13	2.13
conditioned augmented	Perplexity	2.18	2.06	**2.01**	2.02
conditioned bootstrap	Accuracy	9.72%	11.47%	12.97%	14.2%
conditioned augmented	Accuracy	16.92%	18.29%	**20.82**%	19.37%

Table 2: Test loss, test perplexity, and test accuracy of the models trained on each dataset with varying dropout.

We find that optimal test loss is achieved at a dropout value of 0.3. Notably, even with optimized dropout we observe a significant performance advantage from training on the augmented dataset.

Code. Source code is available at: https://github.com/joepalermo/synthetic-proof-term-data-augmentation

Acknowledgements. The authors would like to thank Jason Rute, Alok Singh, Alex Krizhevsky, Ragavan Thurairatnam, Hashiam Kadhim, Marc Tyndel, Rayhane Mama, and Louay Hazami for helpful discussions.

References

[1] T. mathlib Community, "The lean mathematical library," in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, (New York, NY, USA), p. 367–381, Association for Computing Machinery, 2020.

[2] S. Polu and I. Sutskever, "Generative language modeling for automated theorem proving," *CoRR*, vol. abs/2009.03393, 2020.

[3] J. M. Han, J. Rute, Y. Wu, E. W. Ayers, and S. Polu, "Proof artifact co-training for theorem proving with language models," *CoRR*, vol. abs/2102.06203, 2021.

[4] L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer, "The Lean Theorem Prover (System Description)," in *Automated Deduction - CADE-25* (A. P. Felty and A. Middeldorp, eds.), (Cham), pp. 378–388, Springer International Publishing, 2015.

[5] M. Wang and J. Deng, "Learning to prove theorems by learning to generate theorems," *CoRR*, vol. abs/2002.07019, 2020.

[6] E. Aygün, Z. Ahmed, A. Anand, V. Firoiu, X. Glorot, L. Orseau, D. Precup, and S. Mourad, "Learning to prove from synthetic theorems," *CoRR*, vol. abs/2006.11259, 2020.

[7] Y. Wu, A. Jiang, J. Ba, and R. B. Grosse, "INT: an inequality benchmark for evaluating generalization in theorem proving," *CoRR*, vol. abs/2007.02924, 2020.

[8] Y. Wu, M. N. Rabe, W. Li, J. Ba, R. B. Grosse, and C. Szegedy, "LIME: learning inductive bias for primitives of mathematical reasoning," *CoRR*, vol. abs/2101.06223, 2021.

[9] V. Firoiu, E. Aygün, A. Anand, Z. Ahmed, X. Glorot, L. Orseau, L. Zhang, D. Precup, and S. Mourad, "Training a first-order theorem prover from synthetic data," *CoRR*, vol. abs/2103.03798, 2021.

[10] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, pp. 29–48, Citeseer, 2003.

[11] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A fast, extensible toolkit for sequence modeling," *CoRR*, vol. abs/1904.01038, 2019.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.