ZK-friendly ML model explorations - Milestone 2

saeyoon17

Feburary 2024

This article outlines the progress of milestone 2 for acceleration program by Ethereum PSE team. Article first restates the goals made for the report, concluding with the progress made and future works.

1 Goals

The goal of milestone 2 is to first refactor code for better usability (so that users have slight more flexibility in proving their model), and provide deeper analysis on zk-transformed neural networks.

- Functionality: Refactor circom proof generation code so that proof generation pipeline can handle multi-layer perceptron with flexible number of hidden layers.
- Analysis: Check proof generation / verification time complexity for various model size.
- Analysis: Try out different weight scaling factors and see how they affect time complexity / performance.

2 Progress

The progress made are as follows.

2.1 Refactor circom proof generation

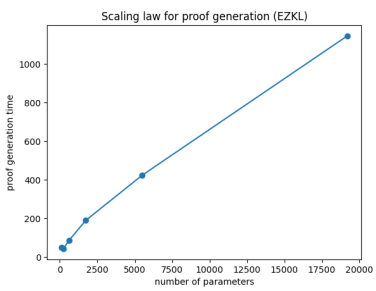
For circom proof generation, the user needs to appropriately scale input and weights so that zk-converted circom circuit resembles the original model architecture. For example, different scaling factors needs to be adopted to different layers of the neural network. To increase the usability, we refactored circom proof generation code so that it can deal with flexible number of hidden layers for *multi-layer perceptron (MLP)*. We do this by considering the fact the input scales upwards by scaling factor for each layer.

2.2 Plot scaling law for proof generation / verification

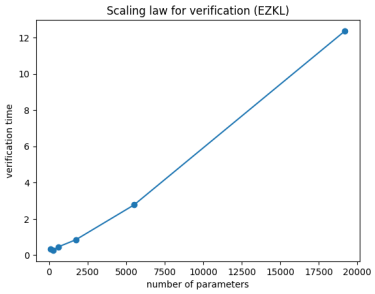
Scaling law is an important concept adopted in various domains to effectively interpolate/extrapolate the expected cost / performance. In this work, we wanted to see how proof generation and verification cost scales with increased number of parameters.

We increase the model size in two approaches; hidden dimension and number of layers. We first make plot by increasing the size of hidden dimension. We then make another plot by increasing the number of layers. By comparing two different plots, we hoped to see if specific model architecture (and not number of parameters) had impact on time complexity.

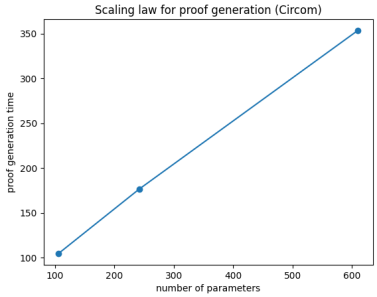
We first present results for increasing hidden dimension while keeping the number of linear layers to 3. We use hidden dimension size of $\{4, 8, 16, 32, 64, 128\}$. For Circom, we only use dimension up to 16 due to circuit constraint size.



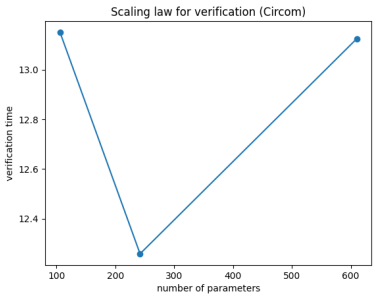
(a) EZKL proof scaling law



(b) EZKL verification scaling law



(c) Circom proof scaling law



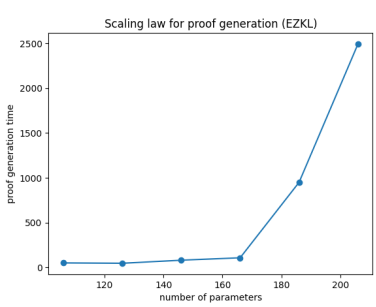
(d) Circom verification scaling law

Figure 1: Scaling law for proof generation / verification

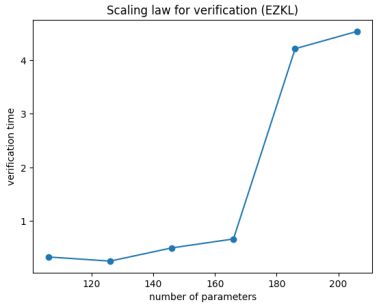
Figure 1 shows the result. It clearly shows that the verification cost scales linearly with the increase of the number of trainable parameters. This aligns with the complexity with Groth16 and Hyperplonk. For EZKL, the verification complexity also scales linearly as shown in Figure 1(b). Circom verification on the other hand, does not scale linearly. Yet the time gap is small, and we note that this is one time computation due to time constraints. In the future we plan

to further strengthen the result by running analysis multiple times.

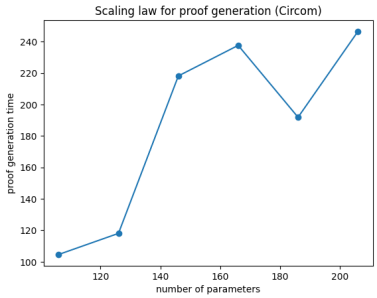
For layer analysis, we fix the hidden dimension to 4 and vary number of layers from 3 to 8. The results are shown in Figure 2.



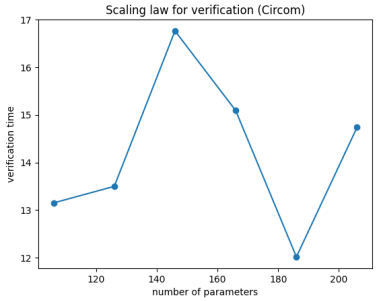
(a) EZKL proof scaling law



(b) EZKL verification scaling law



(c) Circom proof scaling law



(d) Circom verification scaling law

Figure 2: Scaling law for proof generation / verification

The scaling law does not scale linearly in this case. We conjecture that this result is from the use of varying ReLU layers. For EZKL, the computation is done by using lookups whereas it is done by intense operation for Circom. Figure 2(a) shows the proof generation complexity for EZKL. This contains a breakpoint where the tendency changes significantly. For now, we were not able to pinpoint clearly about the cause of such phenomenon. Circom verification also shows similar result to dimension analysis. It did not show any tendency. We conjecture since the proof size of Groth16 is constant, the verification time does not increase to the number of parameters, but rather depend more on local environments.

2.3 Weight scaling analysis

We also analyze different effects of using different scaling factors. EZKL contains its own dynamic scaling strategy, where it contains multiple scaling and rescaling back operations. We use manual scaling for Circom, in which we manually scale

Table 1: Performance degradation

Language	EZKL			
	accuracy	precision	recall	f1
scale=0	40.58	0	0	0
scale=1	40.58	0	0	0
scale=2	40.58	0	0	0
scale=3	64.13	66.17	81.10	72.88
scale=4	63.04	77.19	53.66	63.31
scale=5	65.22	77.42	58.54	66.67
Language	Circom			
	accuracy	precision	recall	f1
scale=0	40.58	0	0	0
scale=1	62.68	87.65	43.29	57.96
scale=2	64.86	74.81	61.59	67.56
scale=3	65.22	76.56	59.76	67.12

weights and input by a scaling factor. Our implementation do not contain rescaling operation. This makes the hidden vectors to scale exponentially as number of layers go up. In this approach, high scaling factor cannot be used. For example, using scaling factor of $1e3$ for MLP with 8 linear layers result in overflow.

A way to alleviate this is to manually add rescaling operations. Yet such paradigm requires the user to input intermediate hidden vectors to the circuit, and also prove that rescaled vectors are within appropriate range. For simplicity, we stick with naive scaling approach and leave this as future work.

To analyze the effect of different weight scaling, we check performance degradation. Since the target task is binary classification, we calculate precision, recall, and f1 score along with accuracy for better analysis. For baseline model, we use model with hidden dimension 4 having 5 linear layers. The performance degradation reports is shown in Table 1.

As the table shows, the tendency shows that increasing the scaling factor reduces the performance degradation except for one case; scale=4 for EZKL. We believe this depends on the dynamic scaling strategy EZKL takes. Since Circom does not have rescaling operations, we were only able to scale up to scale factor of 3. Yet the largest scale used for both tools achieved the accuracy of original neural network.

Through the analysis, we found the necessity of having a general consensus on how users should scale their weights. This should also include when to rescale back. Rescaling does trade off proof generation size and the capability to prove deeper models, therefore a sensitive analysis should be doned to find the sweet spot.

3 Future Works

For milestone 3, we plan to do the followings.

- Further automate code by making scripts that automatically compiles, and generate appropriate keys for Circom proof generation.
- Build train/test pipeline for decision tree using EZKL and possibly by Circom using library built for Circom decision tree proof generation.