TEAM ROCKET

Proof of Learning - Definitions and Practice

IE506 Course Project
22B0003 - Anuttar Jain
22B0045 - Ananya Chavadhal

Presentation Outline

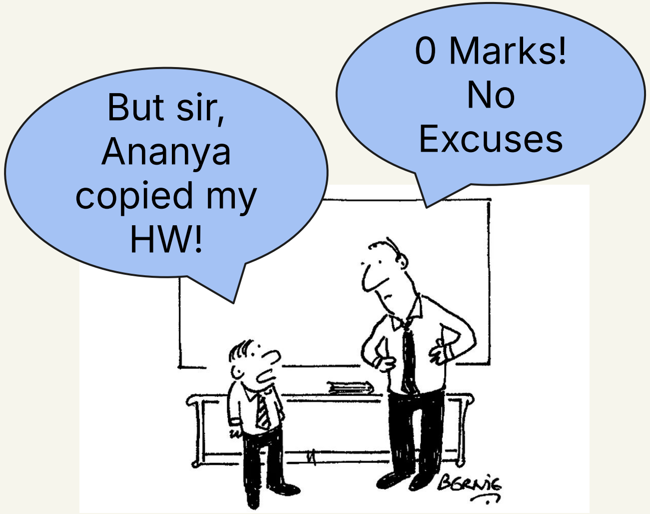
Problems Statement

The paper aims to address mainly the following 2 problems:

Model Ownership Verification: There is currently no way for an entity to prove that a released machine learning model was genuinely obtained through a legitimate training process.

Byzantine-Resilient Distributed Training: In distributed training settings with untrusted workers, malicious participants could sabotage the process by returning incorrect model updates without Detection.

Thus a mechanism must be developed to verify the authenticity of the the model training and ensure that the required computational work was actually done



Algorithm 1 - PoL Generation

- PoL := (**W, I, H, A**)
- **During Training –**
  - Save weights (Wt) at every k steps
  - Save data points (It) corresponding to every step
  - Sign the data points and save the signature (Ht)
  - Save the corresponding hyperparameter values (At) at every step
- **During model release –**
  - Encrypt the PoL with verifier's (V) public key [R := enc(P(f_{WT}), $K_{V,\ pub}$)]
  - Sign the encrypted proof with your private key
  - Publish/timestamp the signature to a public ledger (prevents replay attacks)

Algorithm 1 PoL Creation

Require: Dataset D, Training metadata M
Require: $\mathcal{V}$'s public key $K_{\mathcal{V}}^{pub}$
Require: E, S, k ▷ Number of epochs, steps per epoch, checkpointing interval
Optional: W_0, ζ ▷ Initialization weight and strategy

$\mathbb{W} \leftarrow \{\}, \mathbb{I} \leftarrow \{\}, \mathbb{H} \leftarrow \{\}, \mathbb{M} \leftarrow \{\}$
if $W_0 = \emptyset$ **then**
    $M_0 \leftarrow \zeta$
    $W_0 \leftarrow \texttt{init}(\zeta)$
for $e \leftarrow 0, \ldots, E-1$ **do** ▷ Training epochs
    $I \leftarrow \texttt{getBatches}(D, S)$
    for $s \leftarrow 0, \ldots, S-1$ **do** ▷ steps per epoch
        $t = e \cdot S + s$
        $W_{t+1} \leftarrow \texttt{update}(W_t, D[I_s], M_t)$
        $\mathbb{I}.\texttt{append}(I_t)$
        $\mathbb{H}.\texttt{append}(\mathrm{h}(D[I_t]))$
        $\mathbb{M}.\texttt{append}(M_t)$
        if $t \bmod k = 0$ **then**
            $\mathbb{W}.\texttt{append}(W_t)$
        else
            $\mathbb{W}.\texttt{append}(\textbf{nil})$
$\mathbb{A} \leftarrow \{\mathbb{M}\}$
$\mathcal{R} \leftarrow \mathrm{enc}((\mathbb{W}, \mathbb{I}, \mathbb{H}, \mathbb{A}), K_{\mathcal{V}}^{pub})$
return $\mathcal{R}, \mathrm{h}\left(\mathcal{R}, K_{\mathcal{T}}^{priv}\right)$

Algorithm 2 - PoL Verification

- **PoL Extraction**
    - Decrypt the PoL using V's private key to get PoL
- **Signature Verification (verifier require dataset)**
    - Public Dataset: Immediate verification
    - Private Dataset: Lazy Verification

      Queries prover → gives dataset to verifier → verifies with published signature
- **Initial Weights Verification**
    - Transfer Learning → Verifies PoL of prior model
    - Distribution sampling → Using statistical tests (KS Test)
- **Verification of Weight update per checkpoint interval (k)**
    - If distance(computed(W_k), W_k) > δ : FAIL
    - Focuses verification on Q largest model updates (sorted from mag list)
    - Calibration of δ based on hardware, architecture, dataset, & learning parameters
    - distance(computed(W_k), W_k) < δ ∀ t multiple of k, ∀ epochs

      ⇒ #VERIFIED

Algorithm 2 Verifying a PoL

function $\textsc{Verify}(\mathcal{R}, \mathcal{R}^0, K_{\mathcal{V}}^{priv}, f, D, Q, \delta)$ ▷ encrypted PoLs, $\mathcal{V}$'s private key, model, dataset, query budget, slack parameter
    $\mathbb{W}, \mathbb{I}, \mathbb{H}, \mathbb{M} \leftarrow \mathrm{dec}(\mathcal{R}, K_{\mathcal{V}}^{priv})$
    if $\mathcal{R}^0 = \emptyset$ **then**
        if $\textsc{VerifyInitialization}(\mathbb{W}_0) =$ FAIL **then**
            return FAIL
    else if $\textsc{VerifyInitProof}(\mathcal{R}^0) =$ FAIL **then**
        return FAIL
    $e \leftarrow 0$ ▷ Epoch counter
    $mag \leftarrow \{\}$ ▷ List of model update magnitudes
    for $t \leftarrow 0, \ldots, T-1$ **do** ▷ training step
        if $t \bmod k = 0 \wedge t \neq 0$ **then**
            $mag.\mathrm{append}(d_1(\mathbb{W}_t - \mathbb{W}_{t-k}))$
        $e_t = \lfloor \frac{t}{S} \rfloor$ ▷ Recovering the epoch number
        if $e_t = e + 1$ **then**
            ▷ New epoch started. Verify the last epoch
            $idx \leftarrow \mathtt{sortedIndices}(mag, \downarrow)$
            ▷ get indices for decreasing order of magnitude
            if $\textsc{VerifyEpoch}(idx) =$ FAIL **then**
                return FAIL
            $e \leftarrow e_t, mag \leftarrow \{\}$
    return Success
    function $\textsc{VerifyEpoch}(idx)$
        for $q \leftarrow 1, \ldots, Q$ **do**
            $t = idx[q-1]$ ▷ index of q'th largest update
            $H_t \leftarrow \mathbb{H}_t, I_t \leftarrow \mathbb{I}_t$
            $\textsc{VerifyDataSignature}(H_t, D[I_t])$
            $W'_t \leftarrow \mathbb{W}_t$
            for $i \leftarrow 0, \ldots, k-1$ **do**
                $I_{t+i} \leftarrow \mathbb{I}_{t+i}, M_{t+i} \leftarrow \mathbb{M}_{t+i}$
                $W'_{t+i+1} \leftarrow \mathtt{update}(W'_{t+i}, D[I_{t+i}], M_{t+i})$
            $W_{t+k} \leftarrow \mathbb{W}_{t+k}$
            if $d_2(W'_{t+k}, W_{t+k}) > \delta$ **then** ▷ Dist. func. d_2
                return FAIL
        return Success

Work done before Stage 1 Review

Past Work: *Stealing Machine Learning Models via Prediction APIs*

- Studies how adversaries can steal ML models from MLaaS followed by testing on popular real world services
- Can be done in the following ways:
    - Simple equation-solving model extraction attacks - works for regression and DNNs
    - A new path-finding algorithm for extracting decision trees by modifying input values and finding decision predicates
    - Model extraction attacks against models that output only class labels by finding points near decision boundary then reconstructing and retraining

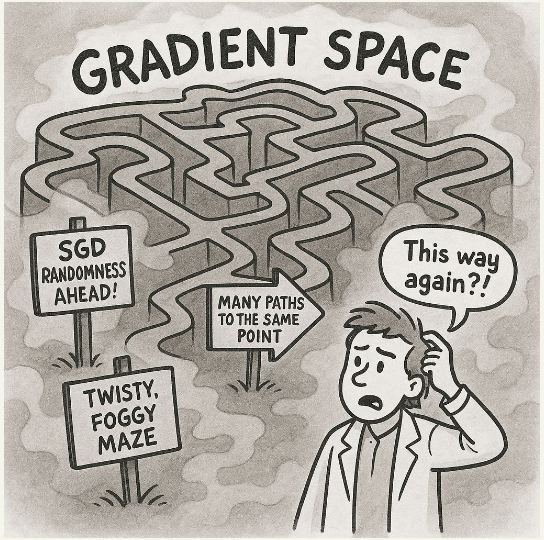
Past Work: *Adversarial Machine Learning*

Case Study: Spam-Bayes

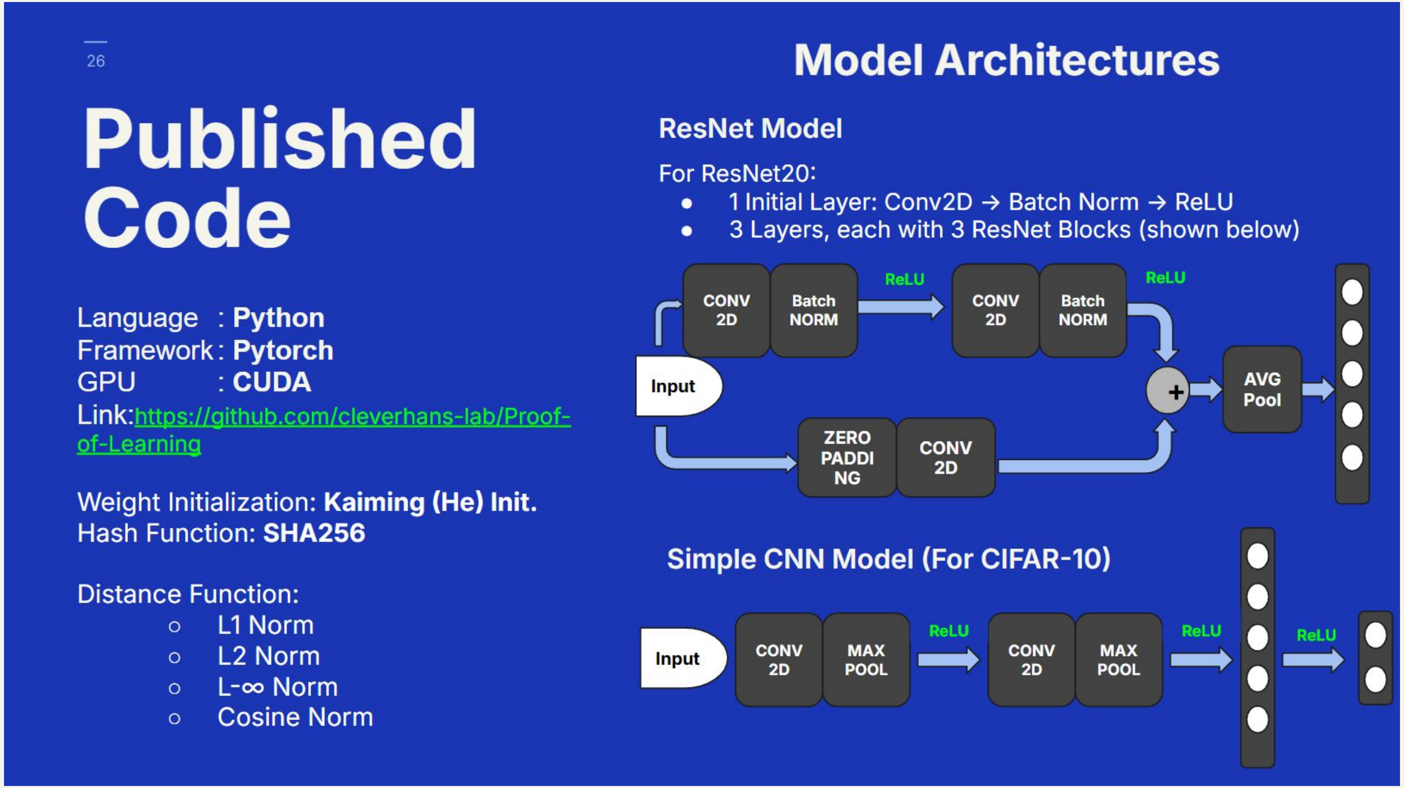
- **Attack:** Causative availability attack (Denial of Service)
- Adversary spams the receiver mails with a large set of tokens (attack's dictionary) that they believe are part of the legitimate mails.
- After some training, the spam score of all those tokens will increase. This will cause the legitimate mails to get marked as spam (false negative) and potentially spam mails to get marked as ham (false positive).
- **Defence:** using RONI technique (Reject on Negative Impact)
- The model maintains of list of tokens that are considered harmful and disregards them for spam score useage.
- If the addition of the token increases misclassification rate above a threshold, then that token is rejected for use.

Paper Understanding

- Experimental Setup:
  - ResNet-20 and ResNet-50 models on CIFAR-10 and CIFAR-100 datasets
  - 200 Epochs; 128 batch size
  - verification relies on checking small intervals where the reproduction error remains below a reference threshold owing to entropy growth
- Stochastic Gradient Descent is used for optimisation
- Randomness introduced by SGD - unique to every training
- Cannot be reproduced by inverting gradient as
  - Multiple ways to arrive at same initialisation
  - Computationally expensive, back training is at least as expensive as forward training
  - Invalid initialisations found usually - fails Kolmogorov Smirnov test
- Directed weight minimisation using regularization term does not work as it requires knowledge of initial weights, leading to a circular problem



Code and More Datasets



Other Datasets that could work:
- MNIST
- OrganMNIST

Other Models:
- Simple_Conv

Link to MNIST:
Click here

Link to OrganMNIST:
Click here

Code link : https://github.com/cleverhans-lab/Proof-of-Learning

Inputs received after Stage 1 presentation

- Since code was already available we were asked to try a different random optimisation algorithm and other datasets and see how they worked

- Work to be done on something that hasn't been touched by the paper yet - like PoL concatenation attacks

Addressing feedback

- We decided to implement ADAM optimiser in the PoL creation algorithm instead of SGD and see how things change

- For the datasets we also tried doing our analysis on MNIST and MedMNIST

- We also wanted to explore genetic optimisation algorithms such as CMA-ES but were advised against it due to time and the uncertainty of obtaining useful results

- We also experimented on possible ways to mitigate PoL Concatenation attacks (discussed further)

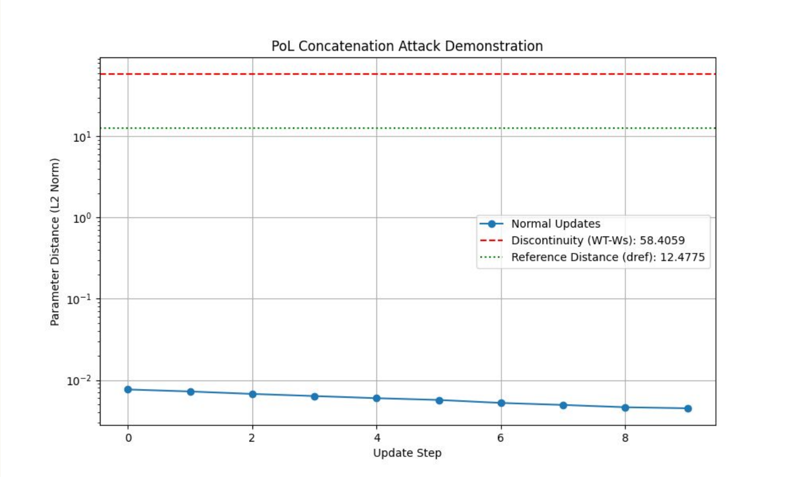
PoL Concatenation Attack - Premise

How can this be done?

- Use a stolen model fWt whose PoL is not available
- Make a valid PoL from this by fine-tuning fWt to obtain f
- Train another model from a random initialization f′ that gives a valid PoL
- Concatenate these 2 PoLs

Does this pass?

- No, update at the concatenation point is very large
- The top Q verification process does not pass

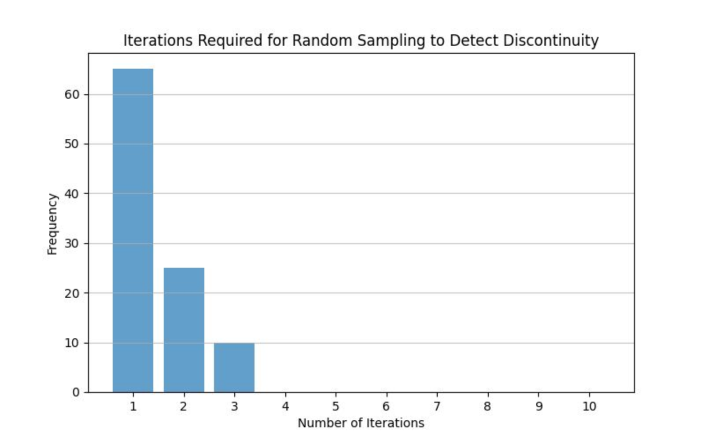


Discontinuity coming from the point of joining is much much larger than updates in a normal training setting.
Using Simple_Conv on MNIST dataset

PoL Concatenation Attack - Exploring

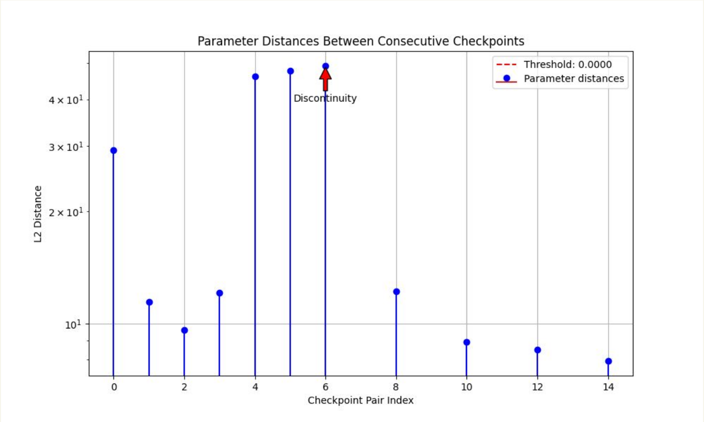
What if we use random sampling instead of Top Q?

- Top Q may be computationally expensive for more epochs, random sampling could be cheaper
- Random sampling only succeeds with probability 1/S (S is number of steps)
- Need to keep iterating - inefficient!



Top Q to the top!

- Top Q detects with 100% probability
- May be expensive but is very reliable
- Flags discontinuity on the first go itself



PoL Concatenation Attack - Smart Adversary

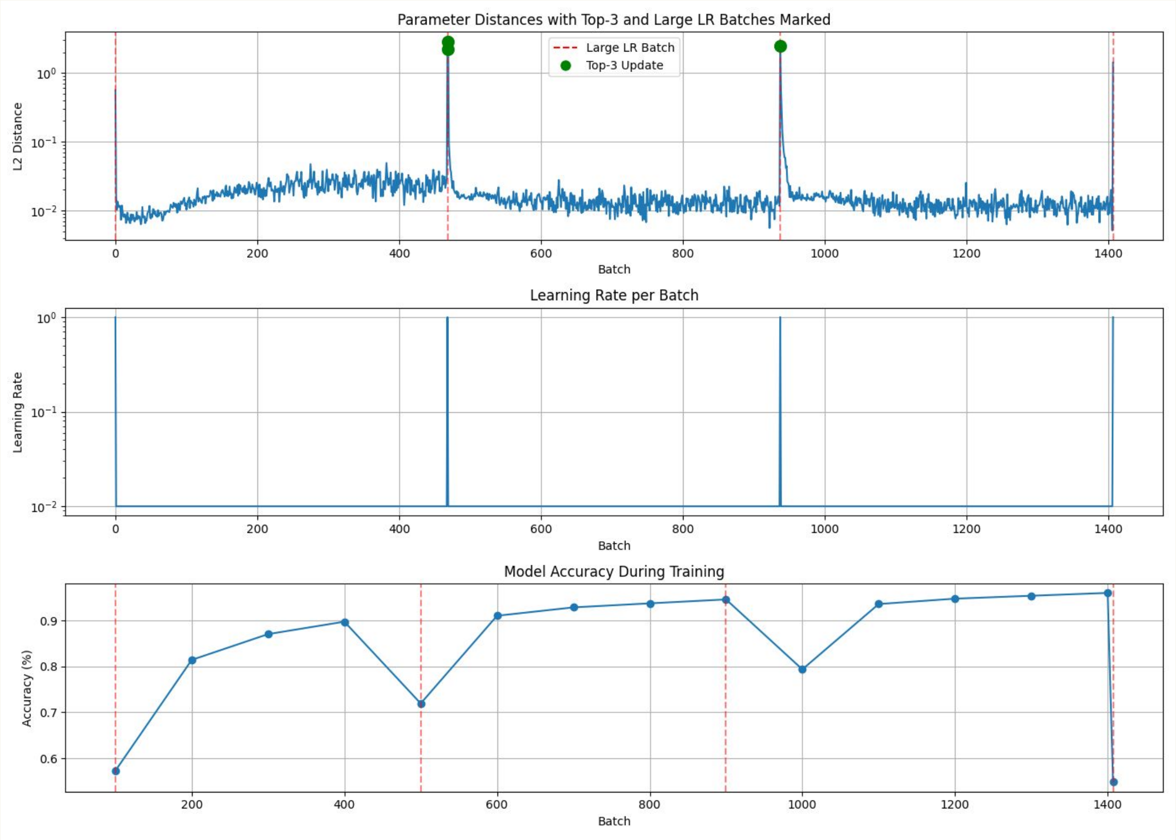
What if the adversary knows the value of Q?

- In this case the adversary can generate Q+1 very large updates, with Q of them justified and only 1 of them spoofed (for eg during concatenation)
- These large updates can easily be justified by putting in very large learning rates at Q points
- The algorithm can detect the top Q updates, deem them legitimate and say that the PoL is valid

Possible ways in which this could be tackled

- Keep increasing Q/keep Q random but big
- Basic top-Q+random sampling
- Monitor performance at each step during training

PoL Concatenation Attack - Smart Adversary




```
Using Q=5:
  - Caught 3 out of 4 large learning rate updates
  - Smallest distance in top-5: 0.586858
  - FAILURE: 1 large updates escaped detection
Using Q=10:
  - Caught 4 out of 4 large learning rate updates
  - Smallest distance in top-10: 0.093597
  - SUCCESS: All large learning rate updates detected with Q=10
```

```
2. Testing top-Q + random sampling:
Using Q=3 + 5 random samples:
  - Caught 2 out of 4 large learning rate updates
  - FAILURE: 2 large updates escaped detection
Using Q=3 + 10 random samples:
  - Caught 2 out of 4 large learning rate updates
  - FAILURE: 2 large updates escaped detection
Using Q=3 + 20 random samples:
  - Caught 2 out of 4 large learning rate updates
  - FAILURE: 2 large updates escaped detection
```

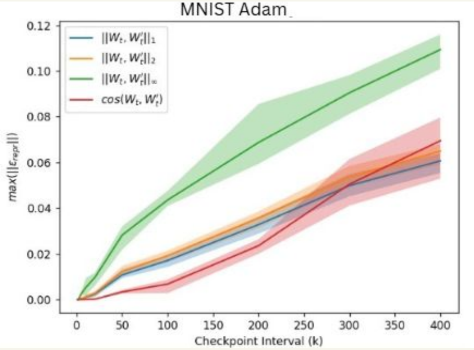
```
3. Testing performance-based detection:
Largest performance drops:
  - Step 1407.0: Drop of 0.41% (Distance to closest large LR: 1.0 steps)
  - Step 500.0: Drop of 0.18% (Distance to closest large LR: 32.0 steps)
  - Step 1000.0: Drop of 0.15% (Distance to closest large LR: 63.0 steps)

Performance-based detection results:
  - Top 3 performance drops would detect 3 out of 4 large LR batches
  - PARTIAL SUCCESS: 3/4 detected via performance drops
```

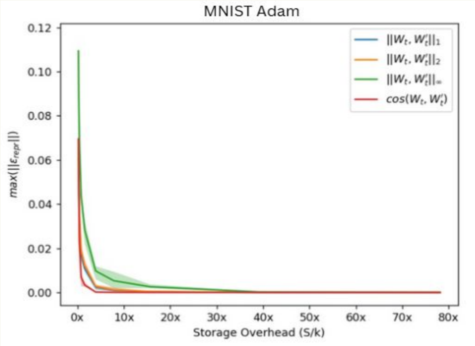
Method 1 > method 3 > method 2

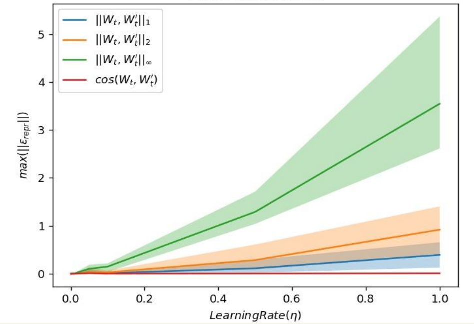
Improvement using ADAM Optimizer

MNIST

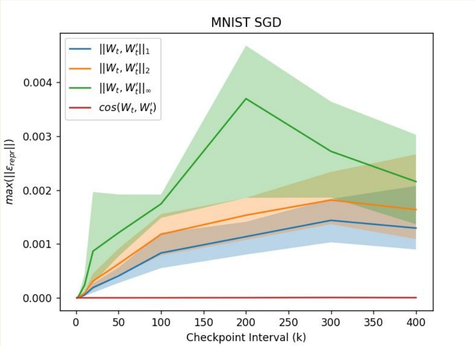
Adam

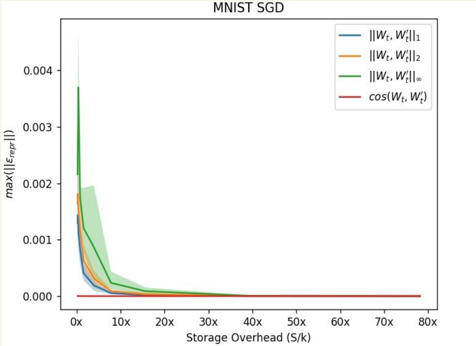


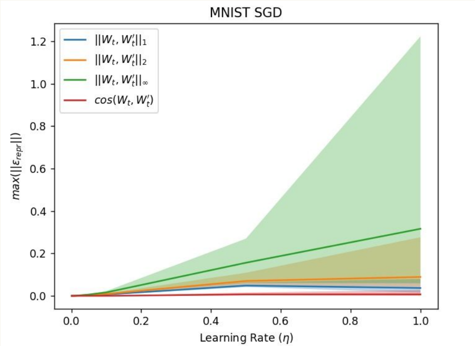




Stochastic Gradient Descent

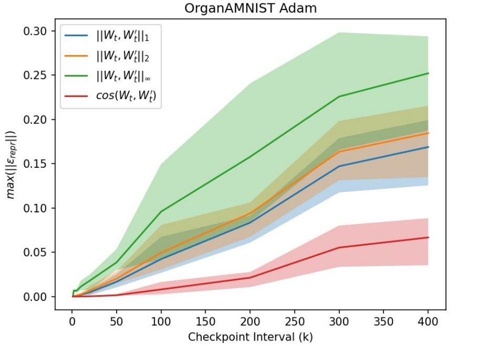


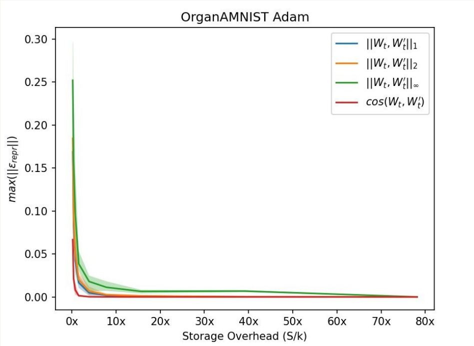


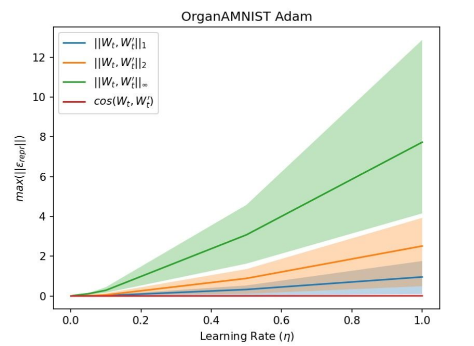


OrganAMNIST (MedMNIST)

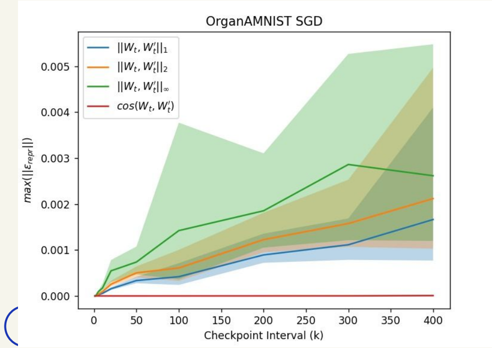
Adam

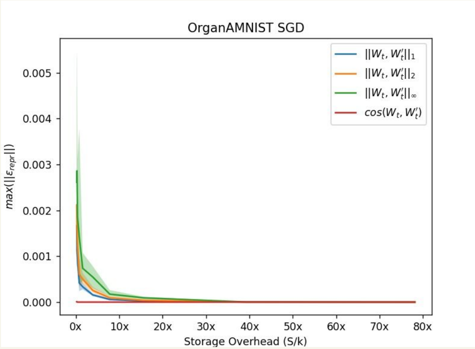


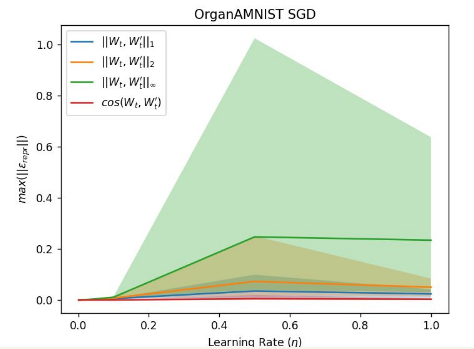




Stochastic Gradient Descent

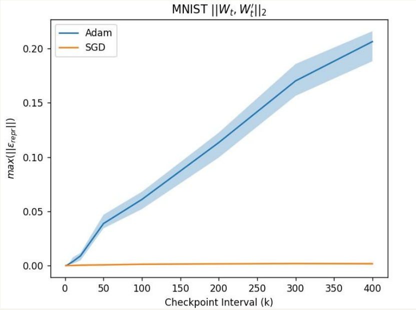


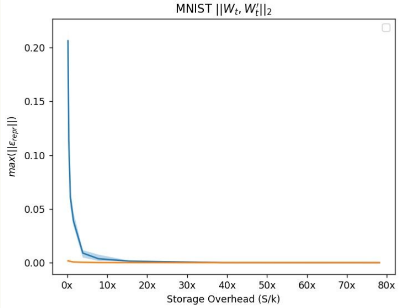


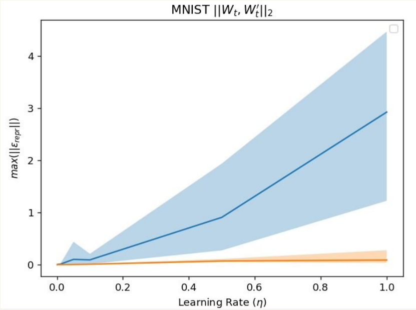


Adam vs SGD Comparison

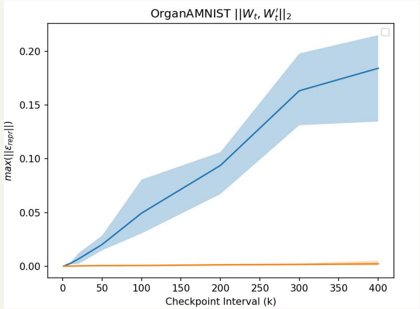
MNIST

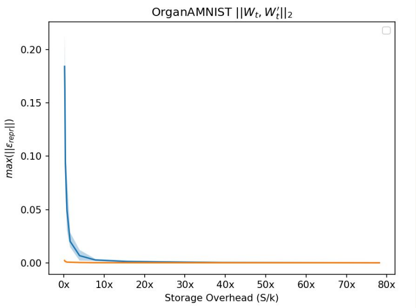


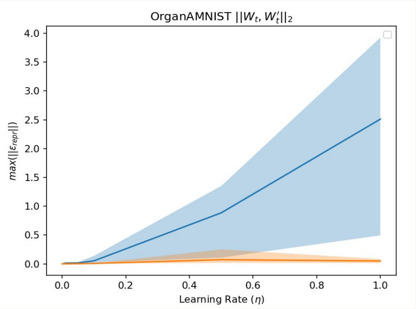




OrganAMNIST







- Adam produces more stochastic error but well within acceptable range
- Adam has larger expected computation time
- Adam computed POL is more difficult to “**spoof**”

Theoretical Validation

- **Markov Nature and Stationarity**
  Like SGD, Adam updates model parameters in a step-wise fashion based solely on the current state (weights and gradients), rendering the optimization process Markovian

- **Linear Entropy Growth**
  Using the same entropy framework as in the original SGD analysis, we define the entropy rate under Adam as

$$H_0(\Theta_T) = \lim_{T\to\infty} H(\tilde{W}_T | \tilde{W}_0, \ldots, \tilde{W}_{T-1}) = H(z_0)$$

Possible future directions

- Try on other types of optimization algorithms for eg. Genetic algorithms
- Work on making verification process even less costly
- Tweak with edge cases of weight initializations in PoL Concatenation attacks

Datasets used

- MNIST: http://yann.lecun.com/exdb/mnist/
- MedMNIST: https://zenodo.org/records/10519652

Contributions

Ananya:

- Understanding the paper
- Slides for theory part for stage 1 review
- Reading past work on model stealing attacks
- Code and experiments for PoL concatenation attacks
- Slides 1-14 in stage 2 review
- Report

Anuttar:

- Understanding the paper
- Slides for code and finding datasets for stage 1
- Reading past work on adversarial ML
- Code and experiments for ADAM, other datasets
- Slides 15-22 in stage 2 review
- Report

References

- Hengrui Jia, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, Nicolas Papernot. Proof of Learning: Definitions and Practice, 42nd IEEE Symposium on Security and Privacy, 2021, https://arxiv.org/abs/2103.05633
- Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
- F. Tramer, F. Zhang, A. Juels, M. K. Reiter, T. Ristenpart. Stealing machine learning models via prediction apis, 25th USENIX Security Symposium (USENIX Security 16), 2016, https://arxiv.org/pdf/1609.02943
- Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, J. D. Tygar. Adversarial machine learning, ACM Conference on Computer and Communications Security, 2011, https://dl.acm.org/doi/pdf/10.1145/2046684.2046692
- Geeksforgeeks. https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/ Accessed on: 15th March 2024
- Geeksforgeeks. https://www.geeksforgeeks.org/blockchain-proof-of-work-pow/ Accessed on: 16th March 2024
- Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, Bingbing Ni. MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification, Scientific Data, 2023, https://arxiv.org/abs/2110.14795
- Jiancheng Yang, Rui Shi, Bingbing Ni. MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis, IEEE 18th International Symposium on Biomedical Imaging (ISBI), 2021, https://ieeexplore.ieee.org/abstract/document/9434062
- Y. Lecun, L. Bottou, Y. Bengio, P. Haffner. Gradient-based learning applied to document recognition, Proceedings of the IEEE, 1998, https://ieeexplore.ieee.org/document/726791

AI Tools Usage

- We used ChatGPT (GPT 4o) and Claude 3.7 in order to understand the paper better by comparing the summarisations it gave with our own understanding of the paper
- Asked it to generate examples that were helpful in understanding the meaning of the text
- Also used it to phrase our points more concisely in the presentation
- Used to understand the hard to decipher lines of code and class definitions formed in the code
- Used to understand the working and implementation of built-in functions of Pytorch and other unknown function
- For stage 2 we took help of AI tools in writing the code and in giving equations for the report.
- Also used AI tools for sanity checks and to come up with better ways to achieve simulations