TEAM ROCKET

Proof of Learning - Definitions and Practice

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

Presentation Outline

- Background
- Problems Addressed
- Motivation
- Past Works
- Problem Setup
- PoL Generation
- PoL Verification
- Practical Considerations
- Experimental Setup
- Computation considerations
- Results
- Datasets
- Published Code
- Bibliography

Background

Proof of learning?

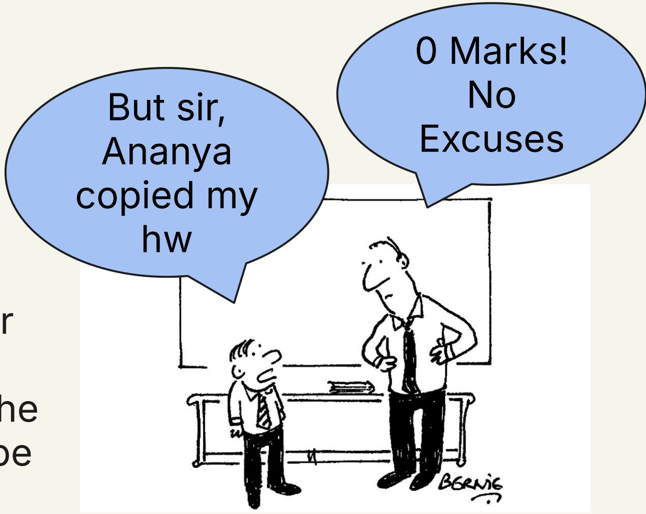
- Cryptographic framework that allows prover to convince a verifier that they have correctly trained an ML model from a given dataset
- Does so without revealing the model itself

Basic Cryptography

- Prover encrypts with verifier's public key and signs with their own private key
- Only the verifier can decrypt with their private key while at the same time ensuring that the signature of the prover cannot be tampered with

Byzantine workers

- malicious or faulty nodes in a distributed machine learning system that behave unpredictably
- They can send incorrect or manipulated updates to disrupt the



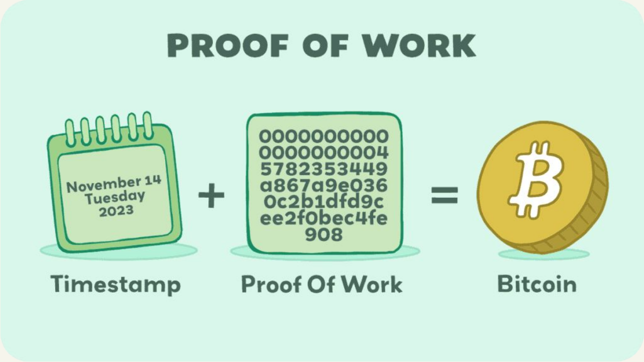
Background

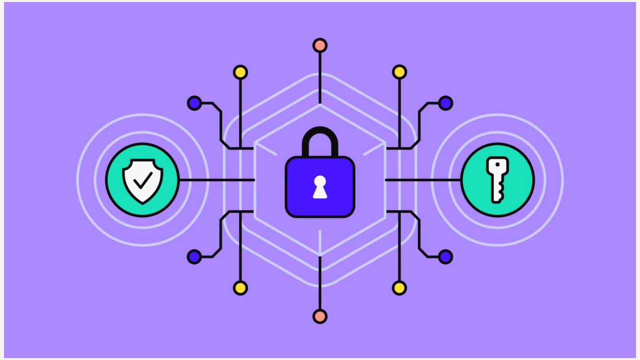
Stochastic Gradient Descent

- The gradient is calculated for each training example (or a small subset of training examples) rather than the entire dataset
- Useful for larger datasets where computing gradient for all points will be computationally very expensive
- Noisy updates can help the model escape local minima or saddle points, potentially leading to better solutions in non-convex optimization problems (common in DL).

Proof of work in Bitcoin

- Ensures that miners must solve a computationally difficult problem before they can add a new block to the blockchain
- Must find a valid nonce, once found they broadcast it to all nodes, verified by other nodes





Problems Addressed

Model Ownership Verification: When multiple parties claim ownership of the same model, there's currently no way to verify who actually performed the training computation.

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

Motivation

- **Shortcomings of ML models**
  - Intellectual Property Protection
  - High Training Costs
  - Distributed Training Security
  - Model Authenticity
  - Preventing Shortcuts
  - Establishing Provenance

- Enough work on watermarking and inference but **no work** on verifying who did the training

Past Work: Stealing Machine Learning Models via Prediction APIs

- The paper investigates how adversaries can "steal" machine learning models that are deployed as services (**ML-as-a-service**), even when they only have black-box access
- The attacks were successfully tested against real-world ML services including BigML and Amazon Machine Learning

Paper talks about:

- Simple equation-solving model extraction attacks
- A new path-finding algorithm for extracting decision trees
- Model extraction attacks against models that output only class labels

Past Work: Stealing Machine Learning Models via Prediction APIs

Simple equation-solving model extraction attacks

- Works for logistic regression and DNNs
- attacker can collect samples (input, class probabilities), treat them as equations, and solve for the model parameters, effectively extracting the model.

A new path-finding algorithm for extracting decision trees

- By modifying input values, the attacker finds decision predicates.
- APIs handling such inputs help reconstruct the tree structure using functions that return leaf or node identifiers.

Model extraction attacks against models that output only class labels

- Use line searches to find points near the decision boundary
- From these samples, it reconstructs the weight vector and bias.
- Uses retraining to achieve low training errors

Drawing from this, our paper aims to curb the vulnerabilities discussed in this paper

Past Work : Adversarial Machine Learning

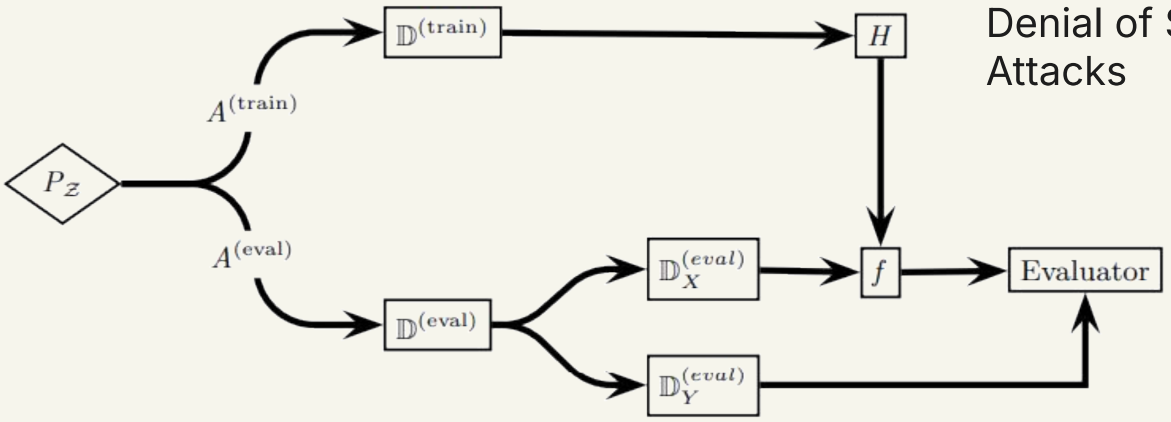
Players–

- Defender: Finds a learning model **H** that is immune to attacks
- Attacker : Adversary trying to disrupt the learning process by transforming Training $\mathbf{A}^{(\mathrm{Train})}$ & Evaluation $\mathbf{A}^{(\mathrm{Eval})}$ Datasets

Model as a Game!

Taxonomy of Attacks

- **Causative** : Manipulate model by injecting malicious training data
- **Integrity** : Giving false negatives during evaluation without being noticeable
- **Availability** : Denial of Service (DoS) Attacks



Case-study: SpamBayes

Methodology of Filter

- Content-based statistical spam filter that classifies using token count
- Evaluates a spam score of the mail by considering the spam score of its tokens and running statistical tests for conclusive remarks
- The score is compared against 2 thresholds to classify the mail as spam, ham (not spam) or unsure

Attack Strategy

- Causative availability attack (Denial of Service)
- Adversary spams the receiver mails with a large set of tokens (attack's dictionary) that they believe are contained in 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 Strategy

- Uses RONI technique (Reject on Negative Impact)
- The model maintains of list of tokens that are considered harmful and disregards them for spam score useage.
- To find such tokens, it trains twice: once using a base training dataset and other including the token and calculates misclassification rate in both cases.
- If the addition of the token increases misclassification rate above a threshold, then that token is rejected for use.

Problem Setup

- Models are parameterized functions fw that map from input space X to output space Y
- W represents the model parameters (weights).
- **Stochastic gradient descent** over T steps (E epochs); in each step:
    - Mini batch sampling
    - Gradient computation of loss function
    - Updating parameters according to: $\boxed{W_i = W_{i-1} - \eta \nabla L_{i-1}}$
- SGD introduces **secret information** unique to training process due to randomness
- Proof-of-Learning (**PoL**) protocol between:
    - A prover T who trains the model and generates a certificate P(T→fWT)
    - A verifier V who analyzes this certificate
    - An adversary A who might try to claim ownership without performing training

Problem Setup

- Four types of spoofing scenarios are considered:
  - Retraining-based spoofing: Creating the exact same PoL
  - Stochastic spoofing: Creating a valid but different PoL
  - Structurally correct spoofing: Creating an invalid PoL that passes verification
  - Distillation-based spoofing: Creating a valid PoL for a modified model

- W: model-specific information from training; I: information about data points used; H: signatures of training data points; A: auxiliary info about hyperparams and architecture

- Verification cost should be less than training cost:
- Adversary's cost should be at least as high as honest training:

$$E[CV] < E[CT]$$
$$E[CT] \leq E[CA]$$

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 for PoL Creation

- **init()** : sets initial weights –
  - Transfer Learning
  - Sampled from a distribution

- **getBatches()** : randomly distributes dataset indices into batches forming T sets of indices

- **update() :** updates $W_t \rightarrow W_{(t+1)}$ using gradient descent update rule

---

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 $\mathcal{R}, \mathrm{h}\left(\mathcal{R}, K_{\mathcal{T}}^{priv}\right)$

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 $\forall$ t multiple of k, $\forall$ epochs ⇒ #VERIFIED

Algorithm for PoL Creation

Please Read Me :(

Verification Success Rate (VSR):

- Defines probability that verifier accepts a PoL
- Depends on probability of calculated updates landing within ε-ball of purported weights
- Models as product of independent probabilities due to Markovian nature of gradient descent

$$\Pr[\textsc{Verify}[\mathbb{W},\phi]=1] = \prod_{e=1}^{E}\prod_{q=1}^{Q}\Pr[Tr_{e,q,k}\ \wedge\ dist_{e,q+k}\le\delta\mid\phi]$$
$$=\prod_{e=1}^{E}\prod_{q=1}^{Q}\Pr[dist_{e,q+k}\le\delta\mid\phi]\cdot\Pr[Tr_{e,q,k}\mid\phi]$$

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**
      $mag.\mathtt{append}(d_1(\mathbb{W}_t-\mathbb{W}_{t-k}))$
    $e_t=\lfloor\frac{t}{S}\rfloor$ ▷ 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

Practical Considerations

- Private Dataset Handling: When datasets are private, the trainer publishes data signatures rather than the data itself, enabling "lazy verification" where actual data is only revealed when verification is needed.
- Data Transfer Requirements: For lazy verification, the expected amount of data needed is calculated as D[1-(1-Qk/S)^E], where Q is verifications per epoch, k is update steps, S is dataset size, and E is epochs.
- Chain of Trust: To prevent claims of "lucky initialization" with stolen models, proofs require reference to previous proofs (P0) that verify initial weights, creating a verification chain with combined success rates.
- Initialization Verification: For models starting from random initialization, statistical methods (Kolmogorov-Smirnov test) verify if weights match claimed initialization distributions, with layer-by-layer testing.
- Constraint on Initialization Strategies: Both trainers and adversaries must use publicly known initialization strategies to prevent adversarial manipulation.

Kolmogorov–Smirnov test

- Compares empirical and theoretical cumulative distribution functions to determine if weights match claim.
- A p-value below the significance level indicates invalid initialization
- Assumes that layer initialisations are independent, Bonferroni's normalisation required if they are not independent

Stationary Markov Process

- Markov process i.e. future state depends only on its current state, not past states.
- It is also stationary, assuming fixed randomness
- This property enables in-place model updates in ML frameworks like PyTorch and TensorFlow.
- Model update is:

$$\tilde{W}_{t+1} = \tilde{W}_t - \eta \nabla_{\tilde{W}_t} \hat{\mathcal{L}}_t + z_t,$$

Entropy Growth

- uncertainty or variance in the gradient descent paths taken during training.
- entropy increases linearly with training steps.
- Because entropy is defined logarithmically in probability, this translates to an exponential increase in possible training sequences
- the presence of small random variations (e.g., hardware noise, cuDNN library behavior) makes exact model reproduction impossible

EXPERIMENTAL SETUP

- ResNet-20 and ResNet-50 were trained on CIFAR-10 and CIFAR-100 respectively
- CIFAR-10 only has 10 classes whereas CIFAR-100 has 100 classe
- Each of the two datasets is composed of 50,000 training images and 10,000 testing images, each of size 32 32 3
- Both models are trained for 200 epochs with batch size being 128

EVALUATION METRICS

- entropy growth in training makes exact reproduction of a model’s training sequence impossible
- verification relies on checking small intervals where the reproduction error remains below a reference threshold.

DETERMINISTIC APPROACHES

- Using deterministic operations in PyTorch reduces training randomness and improves reproducibility but still results in significant errors, incurs high computational costs, and can lower model accuracy.

CHECKPOINTING INTERVAL

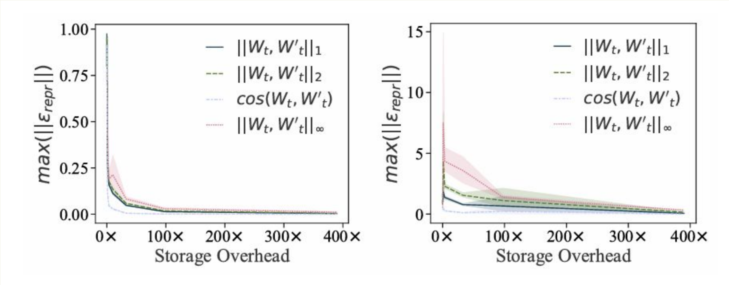
- The study finds that saving checkpoints at every step (k=1) is unnecessary, as using k=S maintains accuracy while reducing storage needs
- To save storage costs save checkpoints in float16 rather than float32

VARYING LEARNING RATE

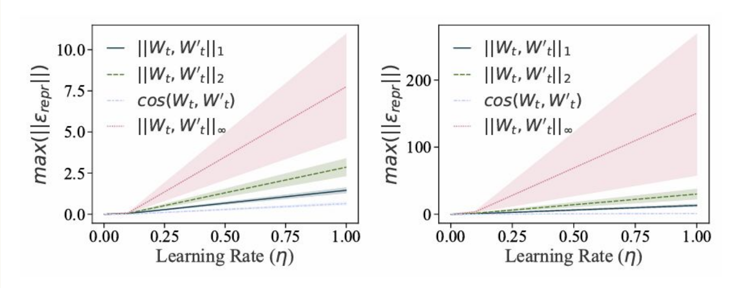
- Learning rate and reproduction error are correlated
- when learning rate is too large, the training process is unstable so a tiny difference may lead to distinct parameters after a few steps.

INITIALISATION VERIFICATION

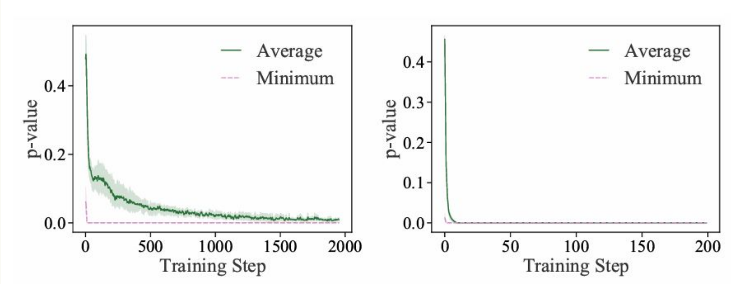
- For both models, the minimum p-value across all network layers drops to 0 rapidly.
- This means that the weight distribution for at least one of the layers is statistically different from the initialization distribution



(a) CIFAR-10 (b) CIFAR-100



(a) CIFAR-10 (b) CIFAR-100



INVERSE GRADIENT METHOD - ISSUES

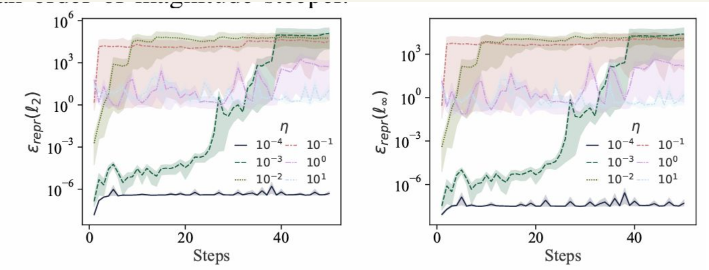
- Attempts to reconstruct the initial weights W0 from the final weights WT by iteratively solving the inverse of the stochastic gradient descent (SGD) step

$$\beta(W_{t-1}) := W_{t-1} - W_t - \eta \nabla_{W_{t-1}} \mathcal{L} = 0$$

- Since different training paths can lead to the same final weights, an adversary trying to spoof training using the inverse gradient method will face high uncertainty
- Reversing a training sequence has at least as much entropy as forward training, making exact reconstruction nearly impossible due to exponentially many training paths as in DNNs (so **retraining based spoofing** is impractical)
- **Stochastic spoofing** won’t work as the adversary still faces a computational cost at least as large as that for T and it is difficult to end in a suitable random initialization.

INVERSE GRADIENT METHOD - ISSUES

- Inverting a training step is at least as computationally expensive as training due to the non-linearity of DNNs and the lack of analytical solutions.
- Increasing learning rates to reduce computational costs leads to high reconstruction errors, making this approach ineffective



(a) ℓ_2 distance (b) ℓ_∞ distance

DIFFICULTY IN FINDING SUITABLE INITIALISATION

- Empirical tests on CIFAR-10 and CIFAR-100 show that inverse gradient methods fail to produce valid initializations, with p-values far below the threshold.
- Even directed approaches, fine-pruning, or sparsification fail to pass verification (KS test)

PoL CONCATENATION

- The difference between WT and Ws is significantly larger than valid gradient updates, making it easy to detect when Q=1. Also, if discontinuity magnitude matches dref, indicating WT and Ws are unrelated.
- If the verifier randomly samples updates instead of selecting the largest, the probability of detecting the discontinuity is 1/S, making verification unreliable
- To counter adversaries exploiting small Q values, potential solutions include increasing Q, random verification, or periodically checking model performance to catch large, unnatural updates.

DIRECTED WEIGHT MINIMIZATION

- An adversary may attempt to retrain toward WT by adding a regularization term that minimizes weight distance but they require knowledge of WT.
- The required information cannot be encoded in synthetic data, as no gradient exists for the regularization term. Tactics like adaptive learning rate tuning to reach W also fail verification.

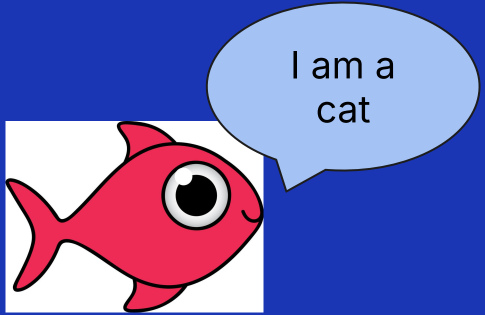
Dataset

CIFAR-10

- Image Classification Dataset
- 32×32 RGB Images
- 60,000 Total Images
- 10 Classes (Plane, Bird, Cat, Frog,...)
- 6,000 Images per class

CIFAR-100

- Image Classification Dataset
- 32×32 RGB Images
- 60,000 Total Images
- 100 Classes
- 20 Superclasses
- 600 images per class
- Each image has 2 labels-
  - Fine : Its class
  - Coarse : Its superclass



Published Code

Language : **Python**
Framework: **Pytorch**
Link:https://github.com/cleverhans-lab/Proof-of-Learning

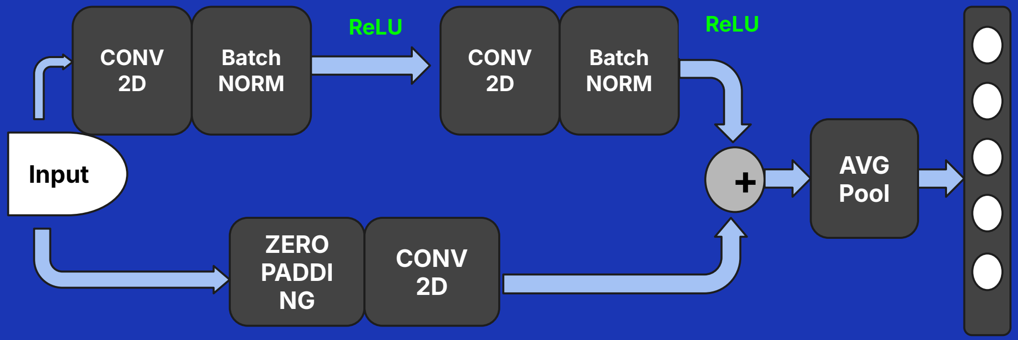
Weight Initialization: **Kaiming (He) Init.**

Model Architectures

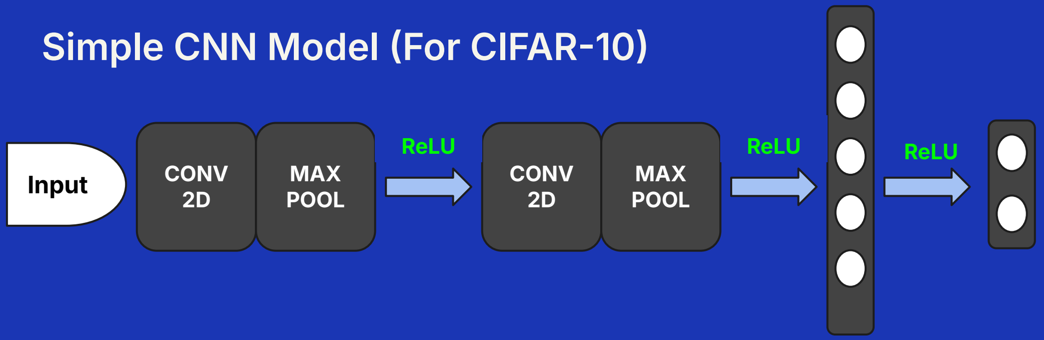
ResNet Model

For ResNet20:

- 1 Initial Layer: Conv2D → Batch Norm → ReLU
- 3 Layers, each with 3 ResNet Blocks (shown below)



Simple CNN Model (For CIFAR-10)



Bibliography

1. Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
2. 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
3. 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
4. Geeksforgeeks. https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/ Accessed on: 15th March 2024
5. Geeksforgeeks. https://www.geeksforgeeks.org/blockchain-proof-of-work-pow/ Accessed on: 16th March 2024

Contribution

Ananya - understanding paper, theory slides, past work on model stealing attacks
Anuttar - understanding paper, code slides, finding new datasets, past work on Adversarial machine learning

AI Tools

- 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