Enhancing and Evaluating Neural Network Extraction Through Floating Point Timing Side Channels

by

Gaurav Vipat

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2023 by the
Graduate Supervisory Committee:

Yan Shoshitaishvili, Chair
Adam Doupé
Siddharth Srivastava

ARIZONA STATE UNIVERSITY

December 2023

ABSTRACT

The rise in popularity of applications and services that charge for access to proprietary trained models has led to increased interest in the robustness of these models and the security of the environments in which inference is conducted. State-of-the-art attacks extract models and generate adversarial examples by inferring relationships between a model's input and output. Popular variants of these attacks have been shown to be deterred by countermeasures that poison predicted class distributions and mask class boundary gradients. Neural networks are also vulnerable to timing side-channel attacks. This work builds on top of Subneural, an attack framework that uses floating point timing side channels to extract neural structures. Novel applications of addition timing side channels are introduced, allowing the signs and arrangements of leaked parameters to be discerned more efficiently. Addition timing is also used to leak network biases, making the framework applicable to a wider range of targets. The enhanced framework is shown to be effective against models protected by prediction poisoning and gradient masking adversarial countermeasures and to be competitive with adaptive black box adversarial attacks against stateful defenses. Mitigations necessary to protect against floating-point timing side-channel attacks are also presented.

ACKNOWLEDGMENTS

This work builds on top of ASU PhD student Zachary Wimer's unpublished dissertation on leaking neural networks in blind black box environments through floating-point multiplication timing side channels.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Motivation

Neural networks have seen unprecedented adoption across computer science in recent years. Their ability to learn and solve complex problems in a time and resource-efficient manner has led to their use in everything from image classification and object detection to security-critical applications like facial detection and network intrusion monitoring and detection. Due to the complexity of the tasks at hand, training models that can perform them well is often computationally expensive. Additionally, the data used for training may be privately owned and confidential. This creates a challenge for developers who want customers to be able to use their models to solve their problems but are unwilling to provide the exact parameters or internal structure they use. To address this issue, models are often deployed with only a black box query interface exposed. The two most common techniques are network deployments and deployments onto trusted hardware. Network deployments store and perform inference with proprietary models on secure remote servers [9, 13]. Users query these models over the network through an application programming interface. This preserves the secrecy of proprietary models while still allowing for easy usage. In environments where a network connection is not available, trusted hardware devices can be leveraged to provide the same guarantees [10]. Models are encrypted at rest and are securely loaded onto trusted hardware on demand. This provides the same effect, keeping model parameters and internal structure private. A similar query interface is exposed for users. These model deployment paradigms have facilitated the monetization of trained models. Services where users pay for access to a proprietary model on a per-query basis have become extremely popular. Paid applications that bundle trained models for classification and detection tasks have also become prevalent.

The rise in popularity of applications and services that charge for access to proprietary trained models and the widespread adoption of neural techniques in security-critical applications has led to increased interest in the robustness of these models and the security of the environments in which inference is conducted. Neural network extraction attacks have been shown to be effective in replicating the functionality and, in some cases, the exact parameters and arrangements of proprietary models [5, 11, 14]. This poses a threat to pay-per-query services as network extraction constitutes theft of intellectual property and can be done cost-effectively. Additionally, query-based attacks have been shown to be capable of generating adversarial examples by making perturbations to in-distribution inputs, resulting in both general misclassifications and specific target classifications [12]. These attacks pose severe threats to security-critical applications of neural networks and can be applied in settings where no prior information about the target model is available and only a query interface is exposed. State-of-the-art black box adversarial attacks rely on extrapolating gradients between a model's input and output. Popular implementations of these attacks have been shown to be deterred by defensive countermeasures that poison predictions by manipulating the distribution of output classes and defenses that limit the propagation of gradients by smoothing a model's classification boundaries.

A model extraction attack that leaks the exact parameters and internal structure of a network while being resilient to adversarial countermeasures would reveal limitations in standard adversarial defenses and expose models that were previously thought to be robust to adversarial threats. Leaking exact parameters and structures would have the added benefit of reducing the black box adversarial example generation problem defined by a query interface to a white box one. The high-precision leaks and additional structural information could be leveraged to guide the creation of adversarial examples, even when the target models have been hardened against iterative black box adversarial attacks. Additionally, if the attack could be masked to evade detection from stateful adversarial attack defenses, it would be applicable to an even wider range of protected targets, further demonstrating the gap in modern adversarial defenses.

Potential Contributions

This work explains in detail the inner workings of Subneural, a framework that applies floating-point timing side-channel attacks to leak exact neural network parameters and the internal structure of their hidden layers. It assumes a black box assumption where the target model is only accessible through a query interface and no prior information about its architecture or structure is available. Critically, the attack is not dependent on the output of individual queries, making the framework applicable in a blind setting where query responses are not available or in black box scenarios against models that are protected by prediction poisoning defenses whose output distributions cannot be trusted. Novel sign assignment and arrangement leakage algorithms that exploit addition timing side channels are introduced, expediting the process of leaking the signs and arrangements of discovered parameters. An addition timing based bias extraction technique is also presented, making the framework applicable to a wider range of targets while preserving its independence from model outputs. The enhanced framework is then evaluated on targets protected by accuracy preserving and full prediction poisoning model extraction defenses. It is also shown to be effective in leaking models protected by gradient masking adversarial countermeasures including defensive distillation. Additionally, modifications to the framework that allow it to be applied to targets protected by query similarity score based stateful adversarial defenses are demonstrated. Finally, countermeasures that mitigate the threat posed by floating-point timing side-channel attacks are presented.

Organization

The following chapters are organized as follows. Chapter 2 summarizes the landscape of adversarial attacks on neural networks, introduces the floating-point timing side channels utilized by Subneural and similar frameworks, explains Subneural's attack stages and limitations, and compares Subneural with an existing floating-point timing side channel-based network extraction framework. Chapter 3 details the application of the addition timing side channel to more versatile and efficient leakage of parameter signs, arrangements, and biases as applied to a single neuron, a single-layered perceptron, and a support vector machine. Modifications necessary to apply the attack to Multi-Layer Perceptrons are also discussed. Chapter 4 provides results on the accuracy of leaked parameters. Chapter 5 provides an overview of prediction poisoning adversarial defenses and shows how Subneural is still effective against models protected by this countermeasure. Neural network distillation is then introduced and its application to defending neural networks from gradient-based adversarial attacks is demonstrated. Subneural is again shown to be effective in leaking models protected by this defense. Implementations of stateful adversarial attack countermeasures are also evaluated against Subneural. Modifications to Subneural that have been shown effective in masking gradient-based attacks to defeat these defenses are also presented. Chapter 6 proposes mitigations for the threats posed by Subneural and other attacks that utilize floating-point timing side-channel attack vectors. Finally, Chapters 7 and 8 summarize the contributions of this thesis and discuss potential avenues for future work.

CHAPTER 2

RELATED WORK

Adversarial Attacks

The explosion in popularity of pay-per-query applications and services [13] has created an increased interest in attacks that leak paywalled models in a cost-effective manner. A number of neural network extraction attacks have been shown to be effective in achieving this goal. Extraction attacks fall into two broad categories, functionality approximation and exact model extraction.

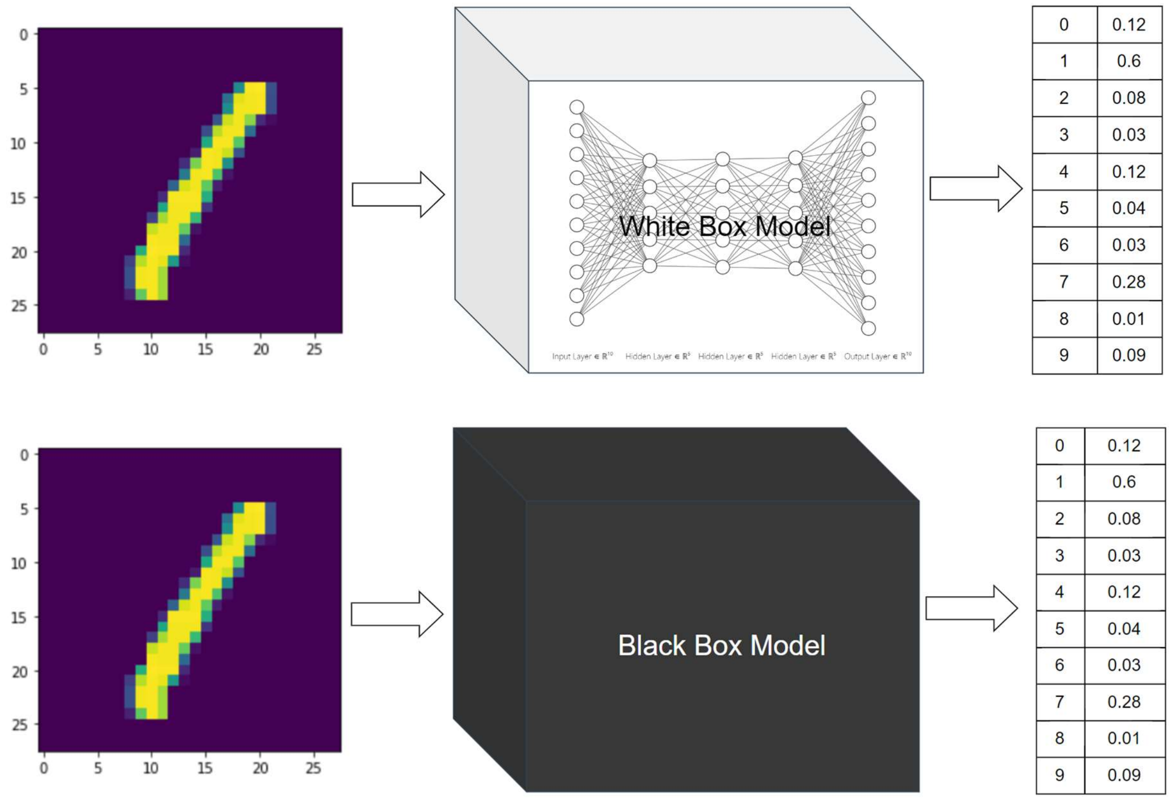
Attacks that attempt to approximate the functionality of a protected model utilize query interfaces to generate a dataset of predictions made by the hidden model [11]. A new neural network is then created by the adversary with an architecture and structure that is determined by prior knowledge about the problem domain or chosen arbitrarily. This replica network is then trained on the input-output pair dataset generated by the protected model, yielding a model that has approximately the same classification boundaries as its target. This technique is feasible because it employs the same principles as knowledge distillation, a technique used to distill the knowledge learned by over-parameterized or ensemble models into smaller more efficient ones. Rather than learning from the one hot encodings provided in training datasets, distilled models can learn from the output probability distributions of the trained teacher model. This allows relationships between classes to be learned in concert with the correct labels, making the training process more sample efficient. Distilled models can often achieve similar classification accuracies while being trained on a fraction of the original datasets. Attacks like Kockoffnets exploit this benefit and have been shown to be effective in replicating the functionality of paywalled models in a bounded number of queries, making the attack cost-effective and feasible against models with high per-query costs.

Attacks that extract exact model parameters and attributes have also been demonstrated. These attacks build on top of functionality approximation attacks by generating a training dataset of protected model predictions and then training multiple candidate models that approximate the functionality of the target, each with different architectures and internal structures [14]. Once the candidate models are generated, a second meta-network is trained to map approximate black box models to their attributes. The predicted attributes include model architecture features, the optimization technique used for training, and information about the original training dataset. This technique has been shown to be efficient in leaking the exact architecture and parameters of protected models in black box environments with limited query budgets.

The widespread adoption of deep learning techniques in security-critical applications has created an explosion of research in adversarial attacks against trained models. A lot of research has been done into adversarial machine learning techniques that generate adversarial examples with the aim of yielding either a general misclassification or a specific alternate classification. The threat posed by these attacks is severe as they can be applied against neural network based facial detection and network intrusion detection and monitoring systems to perturb malformed or malicious inputs, manipulating their inferred classification and evading detection.

Figure 2.1

Illustration of White and Black Box Adversarial Attack Scenarios



0	0.12
1	0.6
2	0.08
3	0.03
4	0.12
5	0.04
6	0.03
7	0.28
8	0.01
9	0.09

0	0.12
1	0.6
2	0.08
3	0.03
4	0.12
5	0.04
6	0.03
7	0.28
8	0.01
9	0.09

Most adversarial attacks fall into one of two categories, white box attacks and black box attacks. White box attacks have prior knowledge about the architecture and structure of the networks that they are attacking. They then use this information to calculate gradients of individual layers with respect to the model output and to inform their adversarial search algorithm, eliminating the need to search the space of possible models and allowing for exact classification boundaries to be calculated and manipulated [15]. However, these attacks have limited applicability as this information is unavailable in most real-world attack scenarios. Pay-per-query inference services restrict access to model parameters and structure, exposing only a black box interface. Applications that are deployed with trained models encrypt them at rest and load them onto trusted hardware for inference, creating a similar black box query environment. Due to these constraints, white box attacks are not applicable in these environments.

Black box attacks have no prior information regarding the architecture of a target neural network nor the structure of its hidden layers. They instead only have access to the inputs and outputs of a model through a query interface. These attacks extract information about the hidden layers of a model by extrapolating relationships between the inputs to a model and the individual class confidence values provided in the model's output probability distributions [16]. Samples from within a target model's training distribution can be modified by predetermined filters or perturbed iteratively. Changes in the output probability distributions are then observed and correlated with the perturbations that caused them. Further perturbations are made in the direction that results in the desired output probability distribution. Most attacks aim to add the minimal perturbations necessary to result in a sample's misclassification, but they can also be used to perturb an example into a specific desired output class. Black box attacks are applicable to a much broader range of real-world targets as most applications and services that provide access to machine learning models as a service do so through a query interface. They have been shown to be effective in finding adversarial examples through query interfaces with limited query budgets.

Figure 2.2

IEEE 754 Floating Point Encoding Format

$$n = Sign * (1 + Fraction) * 2^{Exponent+bias}$$

Floating Point Timing Side Channels

Modern computers utilize the IEEE 754 standard format to encode floating point values in binary [17]. The standard breaks up floating point values into binary encoded signs, fractions, and exponents such that the value n is equal to the sign times one plus the fraction times two to the sum of the exponent and bias as shown in Figure 2.2. This allows arbitrary floating-point values to be stored and represented in binary and facilitates mathematical operations on these binary-encoded values. Revisions to the IEEE 754 format have added provisions for different encoding sizes including 16-bit, 32-bit, and 64-bit, each with a single sign bit and with the remaining bits split between the exponent and mantissa. One of the features of this encoding format is that it has a provision for storing numbers that are smaller than the smallest representable floating-point number. Normal encoded floating-point numbers are required by the standard to have fractions that have no leading zeros. If a value were to have a fraction with a leading zero, its correct representation would shift the fraction to the left and shift the exponent by one. This ensures that the first bit of encoded fractions for all numbers between the minimum and maximum floating-point values for an encoding size is a one. However, in the special case of numbers that are smaller than the minimum representable floating-point value and, as such, would have exponents that are smaller than the smallest representable exponent, fractions with leading zeros are allowed. Floating point encoded numbers that have leading zeros in their fraction and the minimum all zero exponent are called denormalized or subnormal numbers. They make use of the otherwise wasted space in the encoding format to gradually underflow small numbers to zero rather than abruptly truncating values when they pass below the threshold of the minimum normally encoded number. This enables accurate floating-point operations on values that are near this threshold and provides a general boost in the accuracy of operations on larger numbers.

Table 2.1

Floating Point Multiplication Timing Characteristics

Operand	Operand	Result	Speed
Normal	Normal	Normal	Fast
Normal	Normal	Subnormal	Slow
Subnormal	Normal	Normal	Slow
Subnormal	Normal	Subnormal	Slow
Subnormal	Subnormal	Zero	Fast

Several floating-point timing side channels in x86 CPUs have been discovered over the last decade. They all stem from an unintended side effect of supporting subnormal IEEE 754 values. It has been observed that floating-point multiplication experiences performance degradation on the scale of up to two orders of magnitude when certain operands or results are values that have subnormal floating-point encodings. Standard floating-point multiplication of two normal numbers takes 4 clock cycles on a modern CPU [3]. The same computation takes over 200 clock cycles when one of the operands is subnormal. Table 2.1 shows a number of sample floating point multiplication configurations and their performance characteristics. The key observation is that multiplication operations experience severe performance degradation when either operand is subnormal and if the result is subnormal.

Table 2.2

Floating Point Addition Timing Characteristics

Operand	Operand	Result	Speed
Normal	Normal	Normal	Fast
Normal	Normal	Subnormal	Fast
Small Normal	Small Normal	Normal	Fast
Small Normal	Small Normal	Subnormal	Slow
Subnormal	Normal	Normal	Fast
Subnormal	Normal	Subnormal	Fast
Subnormal	Subnormal	Normal	Fast

Operand	Operand	Result	Speed
Subnormal	Subnormal	Subnormal	Fast

Floating point addition also experiences similar performance degradation, but the conditions under which degradation occurs are much more restrictive. As shown in Table 2.2, addition operations require both operands to be small normal floating-point numbers that are strictly less than a fixed threshold for each floating-point encoding format. The maximum threshold value that operands must be less than has been empirically determined and corroborated [3]. If the summation of two small normal numbers is subnormal, floating point addition experiences the same severe performance degradation as multiplication operations. For these constraints to be satisfied, one of the operands must be negative. The limit on the size of the operands is due to the truncation of addition operations when the result underflows below the smallest representable subnormal floating-point numbers. The two operands need to be small enough that the magnitude of the difference between them is subnormal. If the operands get too large, the magnitude of their differences combined with the error associated with floating point subtraction operations results in an underflow and is rounded to zero. The threshold value is dependent on the precision of the encoding standard as the exact value at which this underflow begins to occur is dependent on the size of the fraction and exponent. Exact addition timing threshold values are defined in Table 2.3.

Table 2.3

Floating Point Addition Timing Thresholds

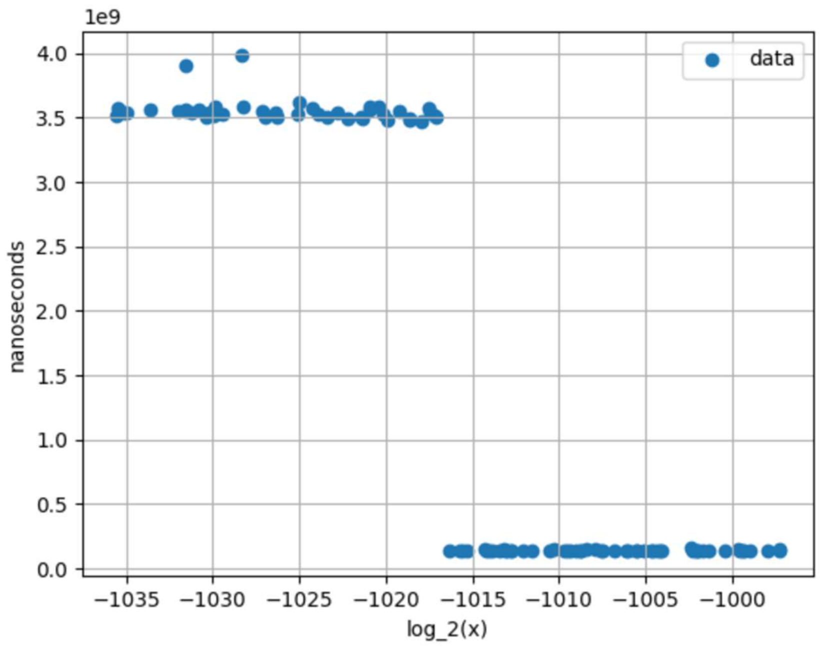
Encoding Precision	Maximum Threshold	Minimum Threshold
Single	6e-33	1e-28
Double	1e-292	2e-308

These floating-point timing side channels can be applied to leak hidden values from floating-point multiplication operations in scenarios in which the secret parameters are multiplied by values input by a user. So long as the inputs to the multiplication operations can be manipulated freely, changes in the running times of these multiplication and or addition operations can be used to leak the values of hidden parameters and make conclusions about the relationships between multiple user-controlled inputs. Consider an example in which a single floating-point value is taken as input and is multiplied by a single value and the output is returned. Assuming the hidden value is a normal floating-point number, we can use the timing characteristics to split the input space into two. For all normal floating-point numbers whose product with the hidden value is also a normal floating-point value, the time it takes for the multiplication operation to complete will be fast. For all normal values whose product with the hidden value is subnormal, the time the multiplication operation takes to complete will be up to two orders of magnitude slower. This partitions the input space into values that are too big for their product with the hidden value to be subnormal, and values that are small enough such that their product with the hidden value is subnormal.

The timing characteristics of one such multiplication operation are shown in Figure 2.3. This property can be exploited to binary search the input space for the input value at which the time the multiplication operation takes changes from fast to slow. The threshold value is known to be the largest value whose product with the hidden value is normal. The smallest normal value can be determined using the numeric limits library for each floating-point encoding precision and since it can be concluded that the product of the input and the hidden value is this largest normal number, the hidden value can be solved for by dividing the determined largest normal number by the input value found in the binary search. This yields an exact leak for the hidden value within the margin of error of floating-point multiplication and division operations for the target encoding format.

Figure 2.3

Timing of Floating-Point Multiplication with a Hidden Parameter



This technique was initially developed and shown to be effective in leaking hidden parameters in web browsers [1] where filters that perform floating-point multiplication of web page pixel values with attacker-controlled filter values can be repeatedly applied to protected web elements by a malicious process to leak the exact values of the hidden pixels.

Side Channel Amplification and Simulation

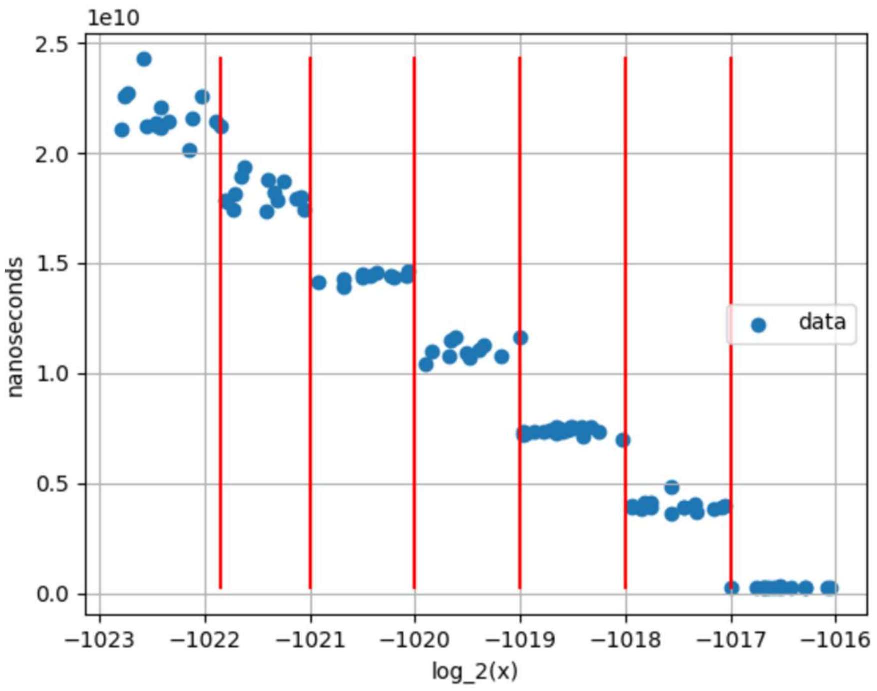
The reality of applying timing side-channel attacks to real-world targets is that the time an operation takes to execute may vary significantly under seemingly identical circumstances. This is due to the fact that in real-world attack scenarios, there may be a number of additional processes running on a target host whose execution is interleaved with the execution of the attack process. This makes the time it takes for the same operation to be executed vary uncontrollably between executions. Suppose a fast floating point multiplication operation is expected to take on average one nanosecond and a slow multiplication takes on average ten. If the variation in execution time due to external factors is more than nine nanoseconds, fast multiplications may be incorrectly registered as slow and vice versa. To address this problem, real word timing side-channel attacks use a technique called amplification in which the same operation is executed with the same inputs hundreds or thousands of times. This, as the name suggests, amplifies the timing side channel, creating a much larger gap between fast and slow executions and allowing for more consistent separation samples. Amplifying a single floating point multiplication operation by a thousand times changes the fast and slow timing scenarios from one and ten nanoseconds to one thousand and ten thousand nanoseconds, greatly increasing the resilience of timing side-channel detection to external factors. The key drawback of amplifying timing side channels is that it makes attacks take significantly longer. Each input will need to be run thousands of times before a clear decision can be made about its performance characteristics. Neural networks can have thousands of parameters per layer, making developing and testing timing side-channel attacks on real-world targets with amplification impractical. For this reason, Subneural uses a simulated timing framework in the form of a floating-point operation library that returns simulated exact computation times for each operation given its inputs. Model parameters and structure are stored as structs and inference is conducted using simulated timing operations. This greatly expedites the search process as each input configuration only needs to be tried once. A timer class is used to decouple the attack from the underlying timing implementation so attacks developed using the simulated timing framework could be easily adapted to use amplified timing side channels on real-world targets by simply replacing the timer class with one that is appropriate for a given scenario and target.

Leaking Neural Structures Through Timing Side Channels

Floating-point timing side channels can also be applied to neural network extraction. Most neural inference techniques can be reduced to the multiplication and accumulation of floating-point vectors and matrixes. Attack frameworks that manipulate inputs and make deductions based on changes in the time it takes for a model to return an output have been shown to be effective in leaking exact hidden parameter values [3]. Subneural is one such framework developed by ASU Ph.D. student Zachary Wimer that extracts neural networks without dependence on the outputs of queries. The attack is split into four stages, leaking weight magnitudes, leaking neuron biases, assigning weight signs, and discerning weight arrangements. The first step is to leak the values of network parameters. In environments in which the input to a model can be freely manipulated, Subneural applies the binary search property previously defined on a per-element basis. It isolates inputs by setting all the other elements in the input vector to zero, then searches for changes in performance characteristics in the input space of the isolated input. This ensures that any changes detected in inference times can be attributed to the input element being modified. Applying this technique to the evaluation of a single neuron yields the weight corresponding to the isolated input times the provided input plus the product of the remaining weights and zero, leaving just the product of the target input and its corresponding weight as the input to the neuron's activation function. Under the standard assumption that all model parameters are normal floating-point values, it can be concluded that the product of our target input and weight will display a degradation in performance when its result is subnormal. This knowledge can be used, as previously shown, to binary search for the threshold value at which the execution time of the multiplication operation changes. Slow and fast baseline execution times can be determined by using the maximum subnormal value and one respectively, and inputs between them can be searched accordingly. The search will terminate at the boundary at which the computation time spikes which is known to be where the product of the first weight and the first input are equal to the largest subnormal value. The input value at this threshold and the minimum normal floating-point value are both known so the value of the hidden parameter can be solved for. At this point, the sign of the leaked parameter has yet to be determined as this computation would display performance degradation for both the discovered input and its negation. Signs are assigned to leaked weights in a later stage. The technique described here can then be applied to each of the elements in the input vector, allowing the magnitudes of all a model's parameters to be leaked.

Figure 2.4

Timing of Floating-Point Multiplication with a Hidden Layer



In the case of a single-layer perceptron with multiple neurons, a binary search of the entire input space can be employed to discern the number and magnitudes of hidden parameters. Since each input element has a weighted term in each of the neurons in the layer, slow and fast baselines can be determined using the same values as before. Because there is now one multiplication operation for each neuron, the slow baseline will have one potential degradation for each of the neurons in the layer. This causes the timing characteristics of the space between the baselines to become a step function with an increase in execution time each time the value of the input becomes small enough for its product in another neuron to become subnormal. This necessitates a recursive binary search between each of the detected performance degradations, resulting in a vector of leaked parameters in increasing order of magnitude that corresponds to the isolated input. In the case of support vector machines, a linear or polynomial kernel can be applied to the summation without affecting Subneural's ability to leak parameters. Gaussian and other exponential kernels, however, make this attack infeasible as they restrict the output space that can be searched. The hidden parameters of a multi-layer perceptron can also be leaked via the same technique augmented by a backtracking search.

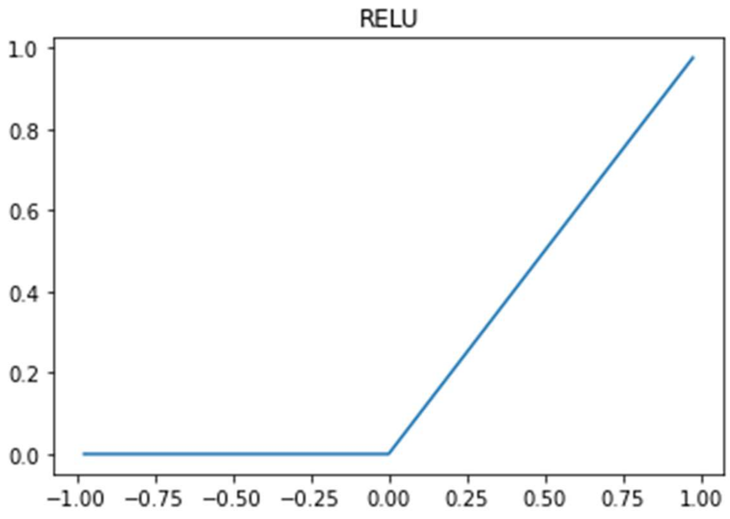
In models in which a bias node is present, the bias must be determined before the signs as the bias value will initialize each accumulation and overpower any small values that follow. There are multiple ways in which the bias can be leaked. If the type of activation function used by a model can be identified or assumed to be tanh or another hyperbolic function, Subneural can take advantage of another timing side channel that affects this family of functions. Tanh experiences similar performance degradation when applied to subnormal inputs. This property can be used to leak the bias in a summation by setting the product of the first input and weight to be our guess for the bias and setting the product of the second input and weight to be a subnormal number. Should the bias guess match the actual bias, those two terms will zero out, allowing the third subnormal term to remain. This will result in a subnormal input to the Tanh activation function which will exhibit detectable performance degradation. This allows the bias term to be brute force searched for utilizing a fixed stride length and to be leaked with the margin of floating-point error.

A backtracking assumption stack is used to test candidate sign assignments and parameter arrangements. Since there is no way to discern this information using only the multiplication timing side channel, an exponential brute force search must be employed. If the performance characteristics of an input are inconsistent with the current assumptions, individual signs and arrangements are iteratively rolled back until a satisfying assignment and configuration is found. This step is also computationally intensive and as such limits the applicability of the attack against networks with large numbers of neurons and inputs.

Subneural has been shown to be effective in blind black box attack scenarios against single neuron targets with and without biases, single layer perceptrons with and without biases, and support vector machines with kernels that don’t restrict the ability to search the output space with and without biases. It has also been demonstrated to be applicable in theory to shallow multi-layer perceptrons.

Figure 2.5

Rectified Linear Unit Activation

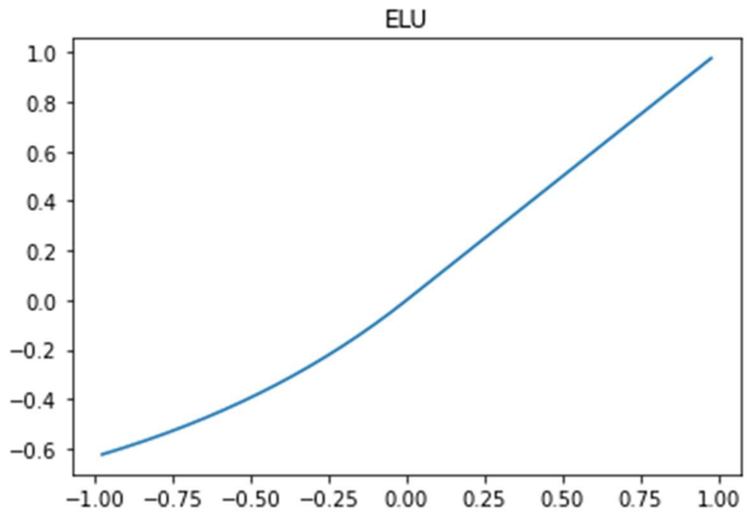


Competing Frameworks

Recent work has shown successful applications of floating-point multiplication and addition timing side-channel attacks to neural network leakage [3]. This attack is reliant on a model's output, making them ineffective against prediction poisoning defenses in black box attack scenarios and making them ineffective in blind scenarios. It also assumes that all of the neurons in the network use rectified linear units (RELU) as their activation functions. RELU is a function that takes a linear input and returns the same linear output with negative values truncated to zero as shown in Figure 2.5. This means that if a neuron's summation was negative and an additional negative input was added to it, the output of the model would not change as RELU would rectify it to zero as it had done before. If the summation was previously positive, adding either a positive or negative value to it would affect the RELU output and in turn the output of the model. The attack exploits this property to determine the signs of leaked parameters and leverages exact output values to determine the signs and magnitudes of biases.

Figure 2.6

Exponential Linear Unit Activation



The assumption that all network layers utilize RELU activations also limits the applicability of these attacks. While RELU is one of the most popular activation functions, it is uncommon for all the layers of a network to use the same activation, RELU or not. Typically, classification models use a SoftMax activation for their final output layer to normalize the output probability distribution. Additionally, RELU alternatives such as the exponential linear unit (ELU), shown in Figure 2.6, have become increasingly popular. ELU allows restricted negative values to propagate through neurons, breaking the assumptions made by this attack and rendering it useless against targets that utilize ELU activations. The attack also makes the assumption that it is possible to detect which layer a slow multiplication occurred in. While this is theoretically possible, it also severely limits the attack’s applicability.

A number of cache timing and power side-channel based neural network extraction attacks have also been demonstrated [2] but these attacks require intrusive access to the hardware being used for inference and or additional attack processes to be run in concert with the inference process. These limitations restrict their applicability to trusted hardware implementations of model query interfaces and render them useless against models deployed using the more popular paradigm of making queries over the network to remote servers that conduct inference. Subneural is already more versatile than these types of attacks and the improvements made in this work further increase its efficiency and applicability.

CHAPTER 3

METHODOLOGY

Overview

This section is laid out as follows. Improvements to the sign assignment component of Subneural that leverage the addition timing side channel mentioned above are first demonstrated against a single neuron. The addition timing side channel is also applied to bias extraction, eliminating the framework's dependence on activation function specific timing side channels. The side channel is then applied to matching relative parameter signs and discerning arrangements of single-layer perceptrons and nonrestrictive support vector machines without bias nodes. Finally, modifications to the framework are proposed that would enable its application to single-layer architectures with biases and shallow neural networks.

Figure 3.1

Single Neuron Computation Visualization

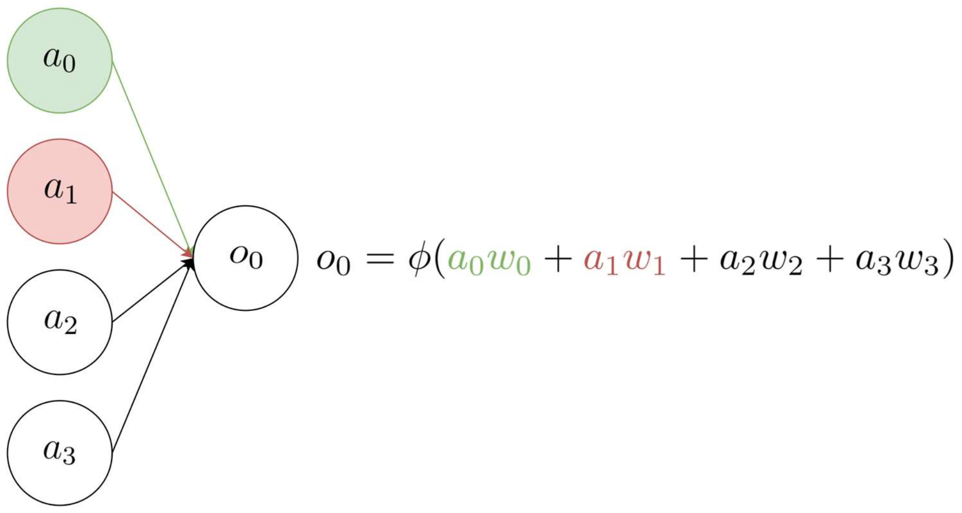


Table 3.1

Addition Timing Neuron Evaluations

Possible Evaluations	Speed
$\phi_0(1 + 3e^{-38} + -2e^{-38} + 1)$	Fast
$\phi_0(3e^{-38} + 1 + 1 + -2e^{-38})$	Fast
$\phi_0(3e^{-38} + 2e^{-38} + 1 + 1)$	Fast
$\phi_0(3e^{-38} + -2e^{-38} + 1 + 1)$	Slow

Linearizing Sign Assignments

In the case of a single layer with one neuron without a bias in its summation, as shown in Figure 3.1, the addition timing side channel can be applied to discern the signs of the leaked weights. Relationships between input elements are evaluated in a pairwise fashion. To determine which pair of inputs corresponds to the first two elements in a neuron's summation, each candidate pair has one value set to a previously defined encoding specific small positive normal number and the other set to a small negative normal number such that the sum of these two values is a subnormal number. The remaining input elements are all set to one as illustrated in Table 3.1. If the elements in the candidate pair correspond to the first two elements in the neuron's summation and the two weights have the same sign, the accumulation of the products of the weight and input vectors will first result in a subnormal number before the remaining ones are added on. As previously shown, the summation of two small normal numbers with a subnormal result will exhibit performance degradation. If any of the ones corresponding to the remaining input elements precede or split the elements corresponding to the candidate pair, the small normal numbers will be summed with a one first which, due to their exceedingly magnitudes, will be rounded to one. This means the two input elements that correspond to the first two elements in a neuron’s summation can be determined through the timing side channel. Additionally, because the summation will only result in a subnormal value when the products of the inputs and their weights have opposite signs, a sign assignment relative to the assumed sign of the first weight can be determined by testing candidate pairs with same and opposite sign variants of the target values. The order of the first two elements also cannot be determined so the first input element is assumed to be the first input in the summation. To determine the third input in a summation, the assumed first input is set to zero and the assumed second input is used to test candidate pairs for the third input. The remaining elements are again set to one. Another performance degradation will be detected when the input corresponding to the third weight in the neuron’s summation is tested with the addition timing targets because the effective summation will be zero plus the two target values plus a sequence of ones. Once the first three inputs and signs are identified, each of the remaining inputs can be iteratively sign matched by setting all of the previously discovered inputs to zero, setting the last discovered input and test input pair to the two addition timing target values, and setting the remaining inputs to one.

Figure 3.2

Activation Dependent Bias Leakage

$$f(\vec{a}) = tanh(b + b' + \sum_{i=1}^{n} w_i * 0)$$

$$f(\vec{a}) = tanh(b + b' + (b - b')' + ... + \sum_{i=m}^{n} w_i * 0)$$

Figure 3.3

Activation Independent bias Leakage

$$f(\vec{a}) = \phi(b + b' + 3e^{-38} + -2e^{-38} + \sum_{i=3}^{n} w_i * 0)$$

$$f(\vec{a}) = \phi(b + b' + (b - b') + ... + 3e^{-38} + -2e^{-38} + \sum_{i=m}^{n} w_i * 0)$$

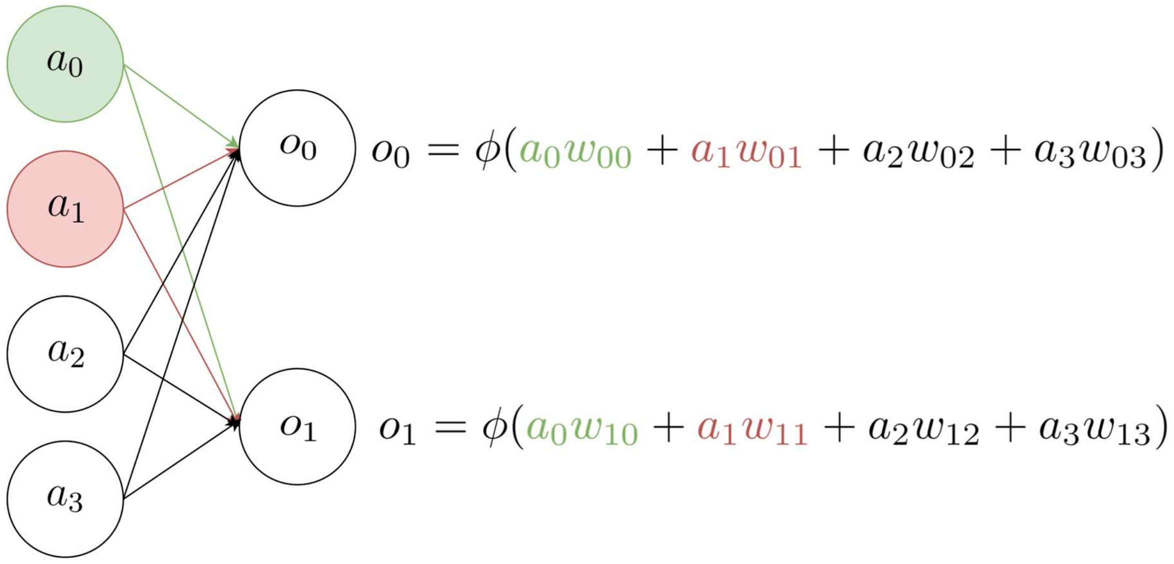
Improving Bias Leakage

Subneural employs a brute force search parameterized by a stride length to search over possible bias values to find inputs that have a subnormal summation when added to the bias. The subnormal results can be detected through timing side channels in neuron activation function. Hyperbolic activation functions like tanh experience performance degradation when evaluated on subnormal inputs. A visualization of this technique is shown in Figure 3.2. While this approach is effective in leaking biases in blind black box environments, many popular network activation functions are not susceptible to any timing side channel. Notably, the standard Rectified Linear Unit (RELU) activation function is not, rendering Subneural useless against the majority of neural networks. This technique, however, can be adapted to instead search for a similar degradation in the performance of floating-point addition operations, making it applicable regardless of the activation function used. If the product of the first term and its corresponding weight is set to the bias guess and the products of the remaining two terms are set to the two small normal targets whose sum is subnormal, the resulting summation will be slow if the bias term is zeroed out by the first product and the following two products are allowed to sum to a subnormal value as shown in Figure 3.3. If the bias guess is incorrect, the first term will dominate, and the following values will be rounded off. This allows Subneural to be used to leak the biases of networks that use activation functions that are not susceptible to a subnormal timing side channel without relying on the influence of activation-specific properties on the model's output.

A limitation of this approach is setting target values for the product of an input and its weight is prone to floating point error accumulation. The summation of the bias and the product of the first input and weight is the only aspect of the algorithm in which a multiplied input must sum with another value to be exactly zero. Due to floating point error, this is extremely unlikely and further effort is required. The simplest solution is to approximate the error using additional floating-point operations. The error in the summation of the bias and the first input's target can be mitigated by the second input and so on until the error becomes subnormal at which point, we can apply one of the two aforementioned techniques. The error introduced by these floating-point multiplication and addition operations necessitates using the largest possible floating-point values for which there is an addition timing side channel as the final two targets. This allows for a larger margin of error in the zeroing out of the bias term while still allowing for slow additions. A potential alternative solution would be to use higher precision floating point arithmetic operations that allow the product of an input and a weight to be accurately set to a target value and then use the corresponding lower precision values to achieve those targets, but this would be difficult to implement and is left as future work.

Figure 3.4

Single Layer Computation Visualization



a_0
a_1
a_2
a_3
o_0
o_1
$o_0 = \phi(a_0 w_{00} + a_1 w_{01} + a_2 w_{02} + a_3 w_{03})$
$o_1 = \phi(a_0 w_{10} + a_1 w_{11} + a_2 w_{12} + a_3 w_{13})$

Table 3.2

Floating Point Addition Timing Thresholds

Neuron 0	Neuron 1	Speed
$\phi_0(6e^{-33} + -5.7e^{-33} + 0 + 0)$	$\phi_1(6.2e^{-33} + -5.9e^{-33} + 0 + 0)$	Fast
$\phi_0(-6e^{-33} + -5.7e^{-33} + 0 + 0)$	$\phi_1(6.2e^{-33} + 5.9e^{-33} + 0 + 0)$	Fast
$\phi_0(6e^{-33} + 5.9e^{-33} + 0 + 0)$	$\phi_1(-6.2e^{-33} + -5.8e^{-33} + 0 + 0)$	Fast
$\phi_0(6e^{-33} + -5.9e^{-33} + 0 + 0)$	$\phi_1(6.2e^{-33} + -5.8e^{-33} + 0 + 0)$	Slow

Expediting Arrangement Discernment

Addition timing side channels can also be applied to discerning leaked parameter arrangements. Consider the single-layer perceptron with two neurons, no biases, and an arbitrary activation function shown in Figure 3.2. By isolating individual input elements, Subneural's multiplication timing guided recursive binary search can be used to leak one weight for each neuron in the network. The leaked parameters are discovered in an order that is dependent on the way in which the binary search finds performance degradations. As a result, it is not known which leaked parameters for a given input correspond to which neurons. Interactions between the leaked parameters of different inputs can be used to determine which parameters belong to the same neuron, effectively reconstructing an identical network. Employing a similar pairwise query strategy, leaked parameters associated with a given input are sorted in increasing order. The target values can then be set to where its product with the corresponding maximum magnitude weight is set to the smallest value for which an addition timing side channel is present. A second input is then configured such that the product with its corresponding minimum magnitude weight is set to be the largest value that when added with the maximum value from the first input exhibits performance degradation. An example where the maximum value for which an addition timing side channel exists and the minimum value that triggers it are 6e-38 and 5.9e-38 is shown in Table 3.2. This configuration ensures that a forward pass through the model will only exhibit performance degradation if the largest weight from the first input and the smallest weight from the second input are in the same neuron’s summation. From here the values for each input can be iteratively increased until a performance degradation is detected, at which point it can be inferred that the current largest magnitude weight on the first input belongs to the same neuron as the current smallest magnitude weight of the second input. As leaked parameters are matched, the search space and number of combinations that need to be tested gradually decreases, making this approach much faster than a brute-force search over all possible arrangements.

The same technique can be applied to support vector machines with linear and polynomial kernels. These kernels do not restrict the output space that can be searched through them and as such are suitable for sign, bias, and arrangement searches using small normal and subnormal addition timing target values. Support vector machines with Gaussian or radial bias function kernels can not be leaked through these attacks. It can also be augmented by Subneural’s backtracking assumption stack to discern the arrangement of weights in multi-layer perceptrons. The assumption stack is necessary because observed performance degradations may be observed due to summations in neurons from multiple candidate layers. This problem could be addressed by using layer-accurate timing side channels [2] but dependence on such high-precision measurements would affect the applicability of the attacks.

Towards Multi-Bias Leakage

Introducing network biases to single-layer and support vector machine targets complicates the extraction process significantly. Because biases are the first components of neuron summations, they must be leaked first as initializing accumulations with large bias values erases the effects of all succeeding small normal and subnormal floating-point operations. Because it may take multiple inputs per bias to zero out its value, having to search for multiple biases simultaneously without knowing which weights correspond to which neurons is extremely difficult. A large number of sign combinations must be tried for every bias guess, and each guess must be formulated for each of the possible weight arrangements. These additional search dimensions compound with the already difficult task of brute forcing an arbitrary floating-point value to make attacking single-layer structures with biases exceedingly difficult. Additionally, a backtracking assumption stack would be necessary for even the addition timing based attack as performance degradations caused by accidental slow addition operations could result in false bias detections.

CHAPTER 4

RESULTS

The modified version of Subneural was applied to leaking the weights, signs, arrangements, and biases of a single neuron, and to leaking the weights, signs, and arrangements of a single layer perceptron and a support vector machine with a non-restrictive kernel. In accordance with the limitations of the timing side channels, relative sign assignments to an assumed positive first sign were accurately discovered for each neuron across these experiments. Accurate arrangements were also discerned in the single layer perceptron and support vector experiments. Additionally, the modified bias leakage algorithm was demonstrated against a single neuron target. Table 4.1 shows the average leaked parameter accuracy across the single layer perceptron experiments and the leaked bias accuracy from the single neuron experiments. The error in the leaked parameter values can be attributed to floating-point multiplication error. This results in leaked parameters being twice as accurate for double-precision values as they are for single-precision ones. The error in the bias arises from the error in the sequence of floating-point addition operations used to leak them.

Table 4.1

Leaked Parameter and Bias Accuracies

Encoding Precision	Parameter Accuracy	Bias Accuracy
Single	+/- 1.57e-08	+/- 2.48e-08
Double	+/- 1.80e-17	+/- 2.72e-17

CHAPTER 5

DEFENSES

Defeating Prediction Poisoning Defenses

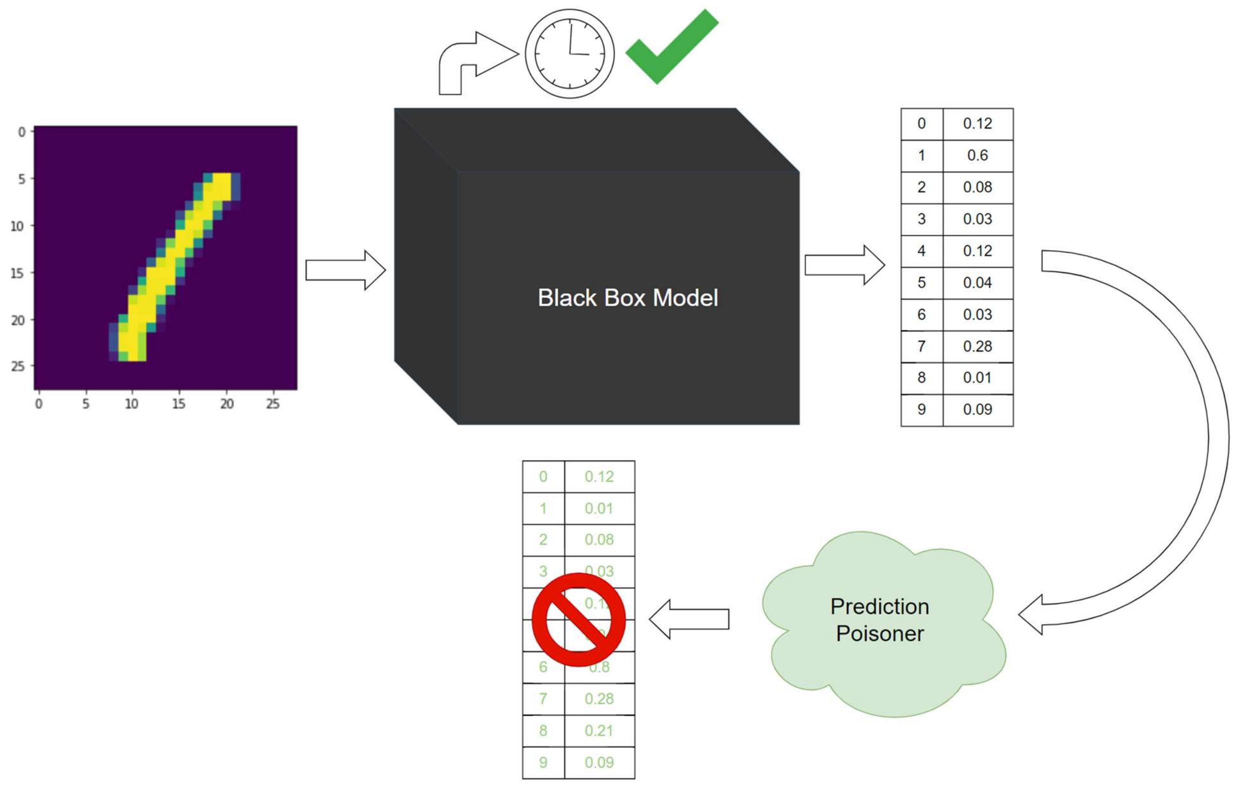
Model extraction attacks have been shown to be extremely effective against black box targets. The most prevalent attacks attempt to train a model that approximates the functionality of a target network through only a model query interface. Models are evaluated on a curriculum of sample inputs and the corresponding output probability distributions are used to generate a distilled training data set. Attackers can then train a student network with an arbitrary or domain knowledge informed internal structure on the dataset generated by the target teacher network. Because the teacher outputs are probability distributions over the class space, the student can learn relations between classes while it is learning the correct labels for each sample, greatly expediting the learning process when compared with training a model on a one-hot encoded dataset like that which the target network would have been trained on. Approximate functionality extraction attacks have been applied to a wide range of target architectures in black box attack scenarios and have been shown to be effective in leaking models in environments where queries are filtered or limited. Additionally, adversarial examples generated for the functionally similar extracted models using white box techniques have been shown to be transferrable to target networks that have been hardened against adversarial attacks, further increasing the threat posed by this attack. There have also been a number of attacks that replicate exact model parameters and internal structure through black box query interfaces. These attacks use prior knowledge about likely neural network architectures to generate multiple candidate networks from the distilled target training data set. They then train metamodels on a dataset of publicly available and generated classifiers [14]. The classifiers learn to predict attributes like the internal structure of a neural network from a model's output probability distributions over an input dataset. Inputs are iteratively refined to better separate the space of possible models. These techniques have been shown to be effective in determining approximate internal structures of their target models in a query-efficient manner, making the leaked white box models much better suited for generating transferable adversarial examples than models that were arbitrarily constructed and trained solely for functionality approximation.

In response to functionality approximating and exact internal structure extraction attacks, a number of prediction poisoning defenses have been developed [8]. These countermeasures perturb the output probability distributions of a protected model with the aim of interfering with an adversary's ability to train a student model on its outputs. Since black box model extraction attacks attempt to train models using the output probability distributions from the outputs of the trained models, poisoning predicted probability distributions causes them to learn incorrect or contradictory relationships between output classes, decreasing their accuracy by up to fifty percent and significantly increasing the number of queries required to extract a model. Variants that use out-of-distribution detectors to identify adversarial queries to poison and that poison all queries indiscriminately have been shown effective against adversarial extraction attacks. Most use cases for black box neural networks only rely on the top one or most likely class prediction. This means that the probability distribution over the remaining output classes can be freely manipulated while still preserving the overall accuracy of the model. In use cases where the top k class probabilities are required, similar accuracy-preserving poisoning can be applied. Poisoning strategies like adding random noise to the output distribution or maximizing the angular deviation of gradients between the poisoned and original predictions have been used to substantially reduce the accuracy of leaked models. Adapted black box model extraction attacks have been shown to be effective against accuracy-preserving prediction poisoning techniques, albeit with a significantly higher query budget. In response to these attacks, countermeasures that detect adversarial queries and either refuse to answer them or provide blatantly incorrect outputs have been developed. While these techniques are not applicable in security-critical environments, they further reduce the accuracy and increase the number of queries required to extract black box models.

Subneural is immune to all prediction poisoning defenses because it does not rely on the outputs of its target models at all. As such, it can be applied against targets with any degree of prediction poisoning. It is also applicable against targets that use a separate module for out-of-distribution detection. So long as the input is fed through the target neural network, the time it takes for the input to be processed can be used to leak its parameters and weights as depicted in Figure 2.5. It can also be applied against models that are trained on adversarial examples or have a dedicated output class for out-of-distribution queries for the same reason. However, the applicability of white or black box adversarial example generation techniques and the transferability of the generated examples to targets that are hardened via these methods is limited. In these scenarios, the attack can only be used to steal proprietary parameters and intellectual property.

Figure 5.1

Using Timing Side Channels to Defeat Prediction Poisoning



Defeating Gradient Masking Defenses

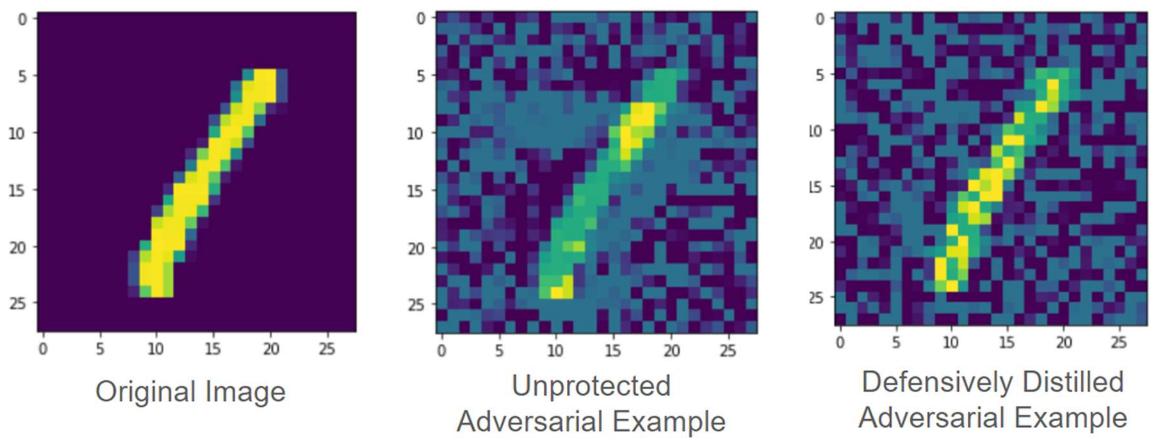
The widespread adoption of neural networks for security-critical applications has created an explosion in the development of adversarial attacks that discreetly perturb samples to manipulate their classification. The most prevalent and broadly applicable black box attacks iteratively perturb samples from within the training distribution of a model and observe the impact of these perturbations on the output probability distributions. The changes are then correlated with the perturbations by extrapolating gradients between the inputs and outputs. Further perturbations are made in the directions that decrease the overall confidence of a model or perturb a sample into being classified as a specific target class [12, 16]. Most attacks attempt to minimize the size of the perturbations necessary to achieve their goals and can find these perturbations in a sample efficient manner. Iterative gradient-based adversarial example generation attacks have been shown to be effective in black box attack scenarios and to pose a severe threat to the robustness of applications of neural networks to security-critical applications like facial recognition and network intrusion and detection.

In response to gradient-based black box adversarial attacks, a number of gradient masking defenses have been proposed to increase model robustness [4, 6]. The most effective of these is defensive distillation. Neural network distillation is a technique used in training neural networks to transfer knowledge from a teacher network to a student network. It was initially proposed to combat the run time complexity of running inference with parameterized or ensemble model architectures. These large and complex models require a lot of memory and compute to make predictions and as such are not applicable in real-time or resource-constrained environments. Distillation is a technique that, as the name suggests, distills the knowledge learned by a larger teacher model into a smaller student model. The teacher model is used to generate a new dataset that maps inputs from the original training dataset to the output probability distributions of the teacher network. The student network is then trained on this dataset which now instead of capturing only one hot encodings of the output class for each sample, captures the learned probability distribution over the output class space. This allows the student network to learn the relationships between output classes in addition to the correct classes for each sample, simplifying and expediting the learning process over the sample distribution. Student models often have significantly fewer parameters and much simpler architectures, reducing their time and space complexities and making them applicable in a wider range of environments.

Defensive distillation applies distillation to smooth out the classification boundaries of a trained model, making it harder for iterative perturbation based adversarial attacks to discern which direction to perturb samples in. This is achieved by training a teacher model on the original dataset for a problem and then training a structurally identical model on the generated dataset of output probability distributions of the teacher model. This provides the same benefits of standard knowledge distillation by allowing the model to learn extracted relationships between output classes. Because the knowledge from the teacher neural network is fed back through a structurally identical network, the student network also learns smoother classification boundaries. The effect of applying this technique is that the student model is less sensitive to small changes in the input [6], resulting in small perturbations of a sample input yielding the same or similar output probability distributions. It has been shown that iterative black box adversarial attacks are significantly less effective against networks protected by defensive distillation, in some cases eliminating the threat entirely.

Figure 5.2

Generated Adversarial Examples



Subneural and other timing side channel attack-based neural network extraction frameworks are inherently immune to these defenses as they either do not rely on the output of a model at all or do not attempt to extrapolate gradients between inputs and their corresponding output probability distributions. At most, timing side-channel frameworks look for changes in output that are unaffected by gradient-based defenses. To demonstrate the effectiveness of attacks like Subneural, we trained several MNIST classifiers ranging from single-layer perceptrons with no bias nodes to shallow fully connected neural networks. These classifiers were then used to generate distilled training datasets of output probability distributions and train identical student networks with a distillation temperature of 10. We then used off-the-shelf adversarial example generation techniques [16, 19] to generate examples for both the unprotected teacher models and the defensively distilled models. Figure 2.6 shows a sample image of a one from the MNIST dataset and the outputs from running a black box attack that applies projected gradient descent to gradients estimated by natural evolution strategies [16] with a target output class of seven on the original and protected models. The attack is able to find the necessary pixels that need to be perturbed to result in the target classification as a seven for the original model but is unable to find an adversarial example for the protected one. The parameters of the defensively distilled model are then loaded into the simulated timing framework used by Subneural and the attack is applied to leak exact parameters and arrangements. White box adversarial attacks can then be used on the leaked model to generate better adversarial examples that have been shown to transfer to the defensively distilled ones [18].

Defeating Stateful Defenses

State-of-the-art techniques in protecting neural networks from query-based adversarial example generation and extraction attacks have moved towards stateful defense systems. This shift was motivated by the inability to curtail these attacks through defenses that poison prediction distributions or modify model weight distributions and classification boundaries to make it harder for adversaries to extrapolate gradients. Stateful approaches to securing neural network query interfaces leverage out-of-distribution detection and query similarity measurement techniques to detect malicious queries. These techniques leverage the fact that iterative adversarial example generation attacks make a lot of similar queries while extrapolating output gradients. If an individual query or query sequence is determined to be malicious, its execution can be halted and the IP address from which those queries originated can be banned. Popular stateful defense implementations [7] maintain a fixed sized buffer of the last k queries received. Each time a new query is made, its similarity to each of the queries stored in the buffer is calculated. If the average similarity score over the n most similar queries drops below a threshold value, the query sequence is flagged as malicious. Query similarity scores can be calculated in a number of ways but the most common is the absolute distance between query tensors. These stateful adversarial defense systems have been shown to be effective in detecting iterative gradient-based attacks.

Table 5.1

Adversarial Attack Average Query Distances

Attack Type	K	K Average Distance
Projected Gradient Descent	50	0.04
Subneural	100	0.01
Subneural (Interleaved)	50	1.4

Floating point timing side channel based adversarial attacks are not inherently immune to these countermeasures as they also make a lot of very similar queries. Like iterative black box adversarial attacks, however, they are able to circumvent stateful defenses by employing a query strategy [20]. Adaptive black box attacks that learn the stateful detection scheme and use it to generate queries that avoid triggering have been demonstrated against similarity score based stateful defenses. A similar adaptive querying technique could be applied to timing side channel based attacks to mask similar queries with benign ones. The binary search employed by Subneural provides the added benefit of being an anytime algorithm that iteratively narrows down the possible range of model parameter values. The search could be cut off at any point and approximate parameter values could be used to bootstrap a model learning approach. Subneural can also be configured to mask its query process by interleaving searches for different inputs, increasing the average similarity score of the attack while still converging to exact parameter values as shown in Table 4.1. It also had the added benefit of yielding a high-resolution model that can be queried and tested independently of a protected model when compared with iterative attacks that only generate individual adversarial examples. White box adversarial attacks can then be used to generate adversarial examples in an unrestricted environment that transfer to the protected model.

CHAPTER 6

MITIGATIONS

Disabling Subnormal Numbers

There are multiple ways to approach mitigating the threats posed by floating-point timing side-channel attacks. The simplest approach is to disable the use of subnormal floating-point values. X86 CPUs have denormals are zero and flush to zero flags that can be used to treat all denormal numbers as zeros and to flush underflows from floating point operations to zero. If these flags are set, the floating-point operations that would otherwise result in subnormal values will instead result in zero, eliminating all of the timing side channels surrounding subnormal numbers. However, disabling the use of subnormal numbers will result in reduced accuracy of floating-point operations. This would effectively reduce the classification accuracy of models on samples that are in close proximity to class boundaries. For most applications of neural networks, the reduced accuracy would be negligible. This would primarily be a concern in security-critical applications where samples are frequently on or near class boundaries and occasional misclassifications can not be masked.

Modifying Stateful Defenses

A practical mitigation for these attacks would be to apply stateful adversarial attack detection techniques to identify malicious queries. The stateful defenses examined in this work could be trained on a dataset of sample query traces from Subneural to learn to detect timing side-channel attack sequences. Floating-point timing side-channel attacks on neural networks require making thousands of queries where all but one of the inputs are zeroed out. All of these queries are well outside of the training distribution for essentially all target models, regardless of domain. If out-of-distribution detection were to be performed by a separate component as a preprocessing step, malicious samples could be detected and handled without ever reaching the protected networks. Refusing to respond to out-of-distribution queries would not only eliminate the threat posed by timing side-channel attacks but would also stifle gradient-based black box adversarial attacks which also make a lot of out-of-distribution queries. A rule-based detection system that determines whether a sufficient number of pixels have nonzero values would also be sufficient for detecting attacks like Subneural.

Neural Network Quantization

Neural network quantization to eight bit signed integers also eliminates the timing side channel due to the way large integers interact with subnormal numbers. Quantization is a technique that decreases the precision of the weights and activations of a neural network to improve the space and computational efficiency of trained models with a large number of parameters or to facilitate real-time inference on embedded devices. The most popular quantized parameter size is 8-bit integers. Quantizing both the weights and the activations to 8-bit integers converts all computations to integer operations and as such eliminates the timing side channel entirely. Weight quantization alone is still susceptible to the subnormal timing side channel in the general case but falls apart with 8-bit unsigned integer weights. Once the magnitude of a weight exceeds 5, multiplication with small normal numbers still experiences the same performance degradation but the result of the multiplication is always positive or negative infinity. This renders the containing summation useless as all other products are dominated by the infinities or result in not a number. Furthermore, the output of applying activation functions like RELU to positive infinity leaves the value unchanged, causing all summations in succeeding layers to also result in infinity, making the output useless. Quantization to 16-bit half-precision floating point values is also popular but x86 CPUs do not support operations on half-precision values. They instead convert them to single precision values at run time, making Subneural equally effective against 16-bit quantized models. Modern GPUs have seen the addition of hardware support for half-precision floating-point values, but they are not susceptible to the subnormal floating-point timing side channels.

Complex Network Architectures

Lastly, convolutional and recurrent neural networks both make it impossible to distinguish where exactly in a model a subnormal computation occurred so adding complex layers or rearchitecting models may be another suitable mitigation. Models that already employ similar techniques do not require additional protections to mitigate the threats posed by timing side-channel based extraction attacks.

CHAPTER 7

CONCLUSION

Summary

This work showed floating-point timing side-channel attacks against neural networks to be effective in leaking common neural structures through query interfaces in blind black box settings. Modifications were made to Subneural to speed up extraction and make it more broadly applicable. Addition timing side channels were leveraged to expedite leaked parameter sign assignment and arrangement discernment. They were also applied to bias leakage to make blind extraction attacks applicable against networks that use a wider range of activation functions. This work also evaluated Subneural against a suite of adversarial attack mitigations. The attack was used to circumvent gradient masking adversarial defenses by facilitating the application of white box adversarial attacks to models extracted through the side channel. Masking techniques used by adaptive black box adversarial example generating attacks against models protected by stateful defenses were shown to be applicable in masking timing side-channel based attacks. Subneural's unique ability to leak models without relying on query outputs was shown to make it resilient to accuracy preserving and complete prediction poisoning adversarial countermeasures. Finally, mitigations that protect against the threats posed by floating-point timing side-channel attacks were proposed and shown to be effective.

Limitations

While these attacks have been shown to be effective against rudimentary neural structures protected by adversarial defenses, there are a number of hurdles in the way of widespread real-world application. As the depth of the target neural network increases, the floating-point error in the computations required to search for weights, biases, and arrangements accumulates. This makes it difficult to set and test target values for leaking biases and arrangements and with enough layers starts to limit the ability to search the input space for values that result in performance degradations in deeper layers. The biggest limitations of blind network leakage attacks are the computational overhead required to amplify the timing side channels and the time it takes to brute force the values of biases. Addressing these challenges is left as future work.

CHAPTER 8

FUTURE WORK

Augmenting Gradient Based Black Box Attacks

Existing black box adversarial extraction attacks on neural networks [5, 14, 16] have been shown to accurately leak network structures and parameters with relatively high degrees of accuracy. While these attacks primarily rely on domain knowledge and brute force search to determine the internal structures of target networks, their model architecture approximations could be used to guide Subneural's search process. Approximate network depth, layer configurations, and weight arrangements could be used to initialize parameter values or a knowledgebase of possible models that could be refined and pruned with timing side-channel based leaks and test cases. Additionally, approximate bias leaks could be used to greatly decrease the brute force search space and running time of Subneural's bias leak stage. At present, bias leakage is the most time-consuming and computationally expensive component of the attack by multiple orders of magnitude. Expediting this step would make Subneural effective against broader and deeper targets, greatly increasing its real-world applicability.

Side Channel Amplification

Subneural's current implementation relies on a simulated timing framework that reports accurate simulated running times on a per-competition basis. This allows each input configuration to be tested with deterministic results, greatly expediting the process of searching for parameters and biases. Applying Subneural in the real world would require significant amplification of the timing side channels it uses for changes in inference times to be accurately separated. Each input configuration will need to be run between 1000 and 100,000 times, depending on the complexity of the target neural structure, to be able to consistently detect performance degradation steps. Amplification is a common technique applied in timing side-channel attacks but due to the large input space Subneural has to search, even small amplifications would significantly increase the running time of the algorithm and may affect its feasibility against broader and deeper targets.

REFERENCES

[1] M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner and H. Shacham, "On Subnormal Floating Point and Abnormal Timing," 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 2015, pp. 623-639, doi: 10.1109/SP.2015.44.

[2] G. Dong, P. Wang, P. Chen, R. Gu and H. Hu, "Floating-Point Multiplication Timing Attack on Deep Neural Network," 2019 IEEE International Conference on Smart Internet of Things (SmartIoT), Tianjin, China, 2019, pp. 155-161, doi: 10.1109/SmartIoT.2019.00032.

[3] C. Gongye, Y. Fei and T. Wahl, "Reverse-Engineering Deep Neural Networks Using Floating-Point Timing Side-Channels," 2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2020, pp. 1-6, doi: 10.1109/DAC18072.2020.9218707.

[4] T. Lee, B. Edwards, I. Molloy and D. Su, "Defending Against Neural Network Model Stealing Attacks Using Deceptive Perturbations," 2019 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 2019, pp. 43-49, doi: 10.1109/SPW.2019.00020.

[5] Duddu, V., Samanta, D., Rao, D. V., & Balas, V. E. (2018). Stealing neural networks via timing side channels. arXiv preprint arXiv:1812.11720.

[6] Papernot, Nicolas & McDaniel, Patrick & Wu, Xi & Jha, Somesh & Swami, Ananthram. (2016). Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. 582-597. 10.1109/SP.2016.41.

[7] Chen, S., Carlini, N., & Wagner, D. (2020, October). Stateful detection of black-box adversarial attacks. In Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence (pp. 30-39).

[8] Orekondy, T., Schiele, B., & Fritz, M. (2019). Prediction poisoning: Towards defenses against dnn model stealing attacks. arXiv preprint arXiv:1906.10908.

[9] E. Chung et al., "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," in IEEE Micro, vol. 38, no. 2, pp. 8-20, Mar./Apr. 2018, doi: 10.1109/MM.2018.022071131.

[10] Tramer, F., & Boneh, D. (2018). Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. arXiv preprint arXiv:1806.03287.

[11] Orekondy, T., Schiele, B., & Fritz, M. (2019). Knockoff nets: Stealing functionality of black-box models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 4954-4963).

[12] Bhambri, S., Muku, S., Tulasi, A., & Buduru, A. B. (2019). A survey of black-box adversarial attacks on computer vision models. arXiv preprint arXiv:1912.01667.

[13] Tu, Z., Li, M., & Lin, J. (2018, June). Pay-per-request deployment of neural network models using serverless architectures. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations (pp. 6-10).

[14] Oh, S. J., Schiele, B., & Fritz, M. (2019). Towards reverse-engineering black-box neural networks. Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, 121-144.

[15] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

[16] Ilyas, A., Engstrom, L., Athalye, A., & Lin, J. (2018, July). Black-box adversarial attacks with limited queries and information. In International conference on machine learning (pp. 2137-2146). PMLR.

[17] Kahan, W. (1996). IEEE standard 754 for binary floating-point arithmetic. Lecture Notes on the Status of IEEE, 754(94720-1776), 11.

[18] Papernot, N., & McDaniel, P. (2017). Extending defensive distillation. arXiv preprint arXiv:1705.05264.

[19] Liu, Y., Mao, S., Mei, X., Yang, T., & Zhao, X. (2019, December). Sensitivity of adversarial perturbation in fast gradient sign method. In 2019 IEEE symposium series on computational intelligence (SSCI) (pp. 433-436). IEEE.

[20] Feng, R., Hooda, A., Mangaokar, N., Fawaz, K., Jha, S., & Prakash, A. (2023). Investigating Stateful Defenses Against Black-Box Adversarial Examples. arXiv preprint arXiv:2303.06280.

APPENDIX A

PERMISSION STATEMENTS

Permission has been obtained from Zachary Wimer, the sole author of the unpublished dissertation that this work builds on top of, for the use of his research and codebase in this master’s thesis.