Reprogramming of neural networks: A new and improved machine learning technique

by

Eliska Kloberdanz

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Jin Tian, Major Professor
Kris De Brabanter
Zhengdao Wang

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2020

DEDICATION

I would like to dedicate this thesis to my husband Kyle, who encouraged me to pursue computer science and supported me throughout my studies at Iowa State University.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Jin Tian for his guidance, patience and support throughout this research and the writing of this thesis. Dr. Tian introduced me to the fascinating topic of artificial intelligence and his teaching encouraged to study AI and machine learning. I would also like to thank my committee members Dr. Kris De Brabanter and Dr. Zhengdao Wang whose courses enabled me to conduct this research.

ABSTRACT

Neural networks can be repurposed via reprogramming to perform new tasks, which are different from the tasks they were originally trained for. We introduce new and improved reprogramming technique that, compared to prior works, achieves better accuracy, scalability, and can be successfully applied to more complex tasks. While prior literature focuses on potential malicious uses of reprogramming, we argue that reprogramming can be viewed as an efficient training method. Our reprogramming method allows for re-using existing pre-trained models and easily reprogramming them to perform new tasks. This technique requires a lot less effort and hyperparameter tuning compared training new models from scratch. Therefore, we believe that our improved and scalable reprogramming method has potential to become a new method for creating machine learning models.

CHAPTER 1. OVERVIEW

In this paper we introduce a new and improved reprogramming technique that allows for creating new accurate neural net models. Compared to prior works on reprogramming, our methodology is more sophisticated and can be applied to complex tasks. Also, while prior studies present reprogramming as an adversarial attack, we view it as a new machine learning technique.

1.1 Introduction

Reprogramming of neural networks was introduced in Elsayed et al. (2019) as a new form of adversarial attack that allows to "...reprogram the target model to perform a task chosen by the attacker - without the attacker needing to specify or compute desired output for each test-time input" Elsayed et al. (2019). We successfully replicated their results; however, when we applied their method to more complex tasks it did not perform well. In this paper we present new and improved reprogramming methodology, which we successfully applied to complex classification tasks. The reprogramming technique in Elsayed et al. (2019) relies on hardcoding an arbitrary mapping between labels of the original and new tasks, and learning an adversarial program P. The program P is applied as a universal additive contribution to all examples from the new task domain regardless of their label to create $X' = X + P$. Our improved technique is instead based on transforming input X to conform to the dimensions that the target model accepts: $X' = f(X)$, and learning a new prediction layer. In particular, our reprogramming method reuses model hyperparameters of the target original model for layers 1 through $L-1$, where L is the total number of layers. We then create a new L^{th} layer with number of neurons equal to number of new task labels and learn its parameters, which can be formulated as $\hat{\theta} = argmin_{\theta}(-logP(y|h))$, where h are the values of hidden units from the penultimate layer of the target model. Using our methodology, we successfully reprogrammed several ImageNet models to perform classification on Caltech 101

and reduced Caltech 256 datasets. Moreover, we demonstrate that creating a new model using our reprogramming technique yields accurate results while requiring a lot less effort in terms of training time and tuning compared to training a new model from scratch. Therefore, we argue that reprogramming does not need to be adversarial, instead we view it as a new machine learning technique.

1.2 Contributions

Our main contributions are two-fold:

- We demonstrate the shortcomings of the methodology for reprogramming introduced in prior literature with various experiments to show that it does not scale to complex tasks. We hope that this finding will be helpful to other researchers.
- We develop a new methodology for reprogramming that addresses the limitations of the prior methodology. Our technique, compared to prior works, yields more accurate results, is scalable, and can be applied to complex tasks. We also show that our reprogramming method can be used to efficiently create new models. Creating new models using our reprogramming methodology requires significantly less effort in terms of training time and hyperparameter tuning compared to training models from scratch.

CHAPTER 2. REVIEW OF LITERATURE

Adversarial reprogramming is a new research area of adversarial machine learning, which studies security vulnerabilities of machine learning models. It is a new type of attack first introduced in Elsayed et al. (2019), which involves *reprogramming* an existing machine learning model to perform a different task. The adversarial reprogramming literature is very limited - to our best knowledge there are only three works that studied adversarial reprogramming: Elsayed et al. (2019), Neekhara et al. (2019), and Wang et al. (2019). Elsayed et al. (2019) successfully conducted white-box adversarial reprogramming on neural net models for image classification. Neekhara et al. (2019) demonstrated that reprogramming also works on text classification neural networks by performing a white-box attack and also a black-box attack. And Wang et al. (2019) claim to have developed a defense against reprogramming attacks.

2.1 Adversarial Machine Learning

Machine learning models, especially deep neural networks, have shown impressive performance in several application domains. However, it has been shown that machine learning models are susceptible to adversarial input examples that can easily fool them causing up to 100% error rate Szegedy et al. (2014) and, thereby, render them useless. Machine learning is used for very important applications such as search algorithms, automated trading, data analytics, driveless cars, malware detection, etc. Therefore, an attack on those models could result in serious repercussions. For example, an attack aimed at machine learning models used for automated electronic trading could cause a market crash with real economic consequences. An attacker targeting computer vision models processing traffic signs in driveless cars systems could cause a car crash by fooling a model to misclassify a stop sign as a yield sign. These findings regarding security vulnerabilities of machine

learning models gave rise to adversarial machine learning - a study of effective machine learning techniques against an adversarial opponent.

Barreno et al. (2006) and Barreno et al. (2010) are one of the first works to discuss security in machine learning. Biggio et al. (2012) explored poisoning attacks against support vector machines, which involve injecting perturbed training data to "poison" the model during training with the goal of reducing testing accuracy. Biggio et al. (2013) then studied gradient-based evasion attacks, white-box attacks conducted during the testing phase by manipulating the gradients of test inputs. However, it was Szegedy et al. (2014), who showed that even state-of-the-art neural networks are vulnerable to adversarial examples, that sparked the growth of adversarial machine learning research.

2.1.1 The Rise of Adversarial Machine Learning

Traditionally, it has been assumed that the environment during training and evaluation of machine learning models is benign. Until the year 2014 the focus of machine learning research has been on accuracy. However, after Szegedy et al. (2014) have shown that neural nets with human level accuracy can have 100% error rate on adversarial examples, robustness of machine learning models and protecting against adversarial attacks became an active research area. Goodfellow et al. (2018) defines adversarial examples, a frequently used term in adversarial machine learning literature, as inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake.

2.1.1.1 Adversarial Attacks on Machine Learning Models

Attacks can be either untargeted or targeted depending on the goal of the attacker – an attacker may design adversarial examples that cause the model to output any incorrect label, or a specific incorrect label. Formally, given a target model that takes x as input and outputs y, and a perturbation η, an adversarial example for an untargeted attack can be formulated as:

$$x' = x + \eta, f(x) = y, x \in X \ s.t. \ f(x') \neq y \tag{2.1}$$

Adversarial example for a targeted attack can be expressed as:

$$x' = x + \eta, f(x) = y, x \in X \ s.t. \ f(x') = y' \tag{2.2}$$

One important factor that determines the attack methods is the amount of knowledge that an attacker has about the target model. In a white-box scenario the attacker has access to the model architecture and parameters, whereas in a black-box scenario this information in not available and the attacker can only query the target model for labels.

There are several white-box scenario techniques for crafting adversarial examples. Some of the most common ones are: Fast Gradient Sign Method (FGMS) Goodfellow et al. (2014), Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm Szegedy et al. (2014), Jacobian-based Saliency Map Attack (JSMA) Papernot et al. (2015) , and Carlini & Wagner (C&W) attack developed by Carlini and Wagner (2016). The FGMS method is one of the most popular technique for its low computational cost. It fixes the size of perturbation and maximizes loss as follows:

$$\begin{aligned} &\eta = argmax_\eta J(x + \eta, y) \ s.t. \ ||\eta||_\infty \leq \epsilon \\ &\eta = argmax_\eta J(x, y) + \eta^T \nabla_x J(x, y) \ s.t. \ ||\eta||_\infty \leq \epsilon \\ &\eta = \epsilon \times sgn(\nabla_x J(x, y)) \end{aligned} \tag{2.3}$$

where η is the perturbation, ϵ is the perturbation magnitude parameter, $\nabla_x J(x, y)$ is the loss function gradient, and sgn is the sign function.

There are three types of methods that can be deployed to conduct a black-box attack: (1) training a substitute model and leveraging white-box techniques for crafting adversarial examples, (2) estimating gradients using zeroth order approximation, or (3) GenAttack, a gradient-free optimization technique that uses genetic algorithms for synthesising adversarial examples.

2.1.1.2 Causes of Adversarial Examples

Adversarial examples are not created by adding random noise to legitimate inputs, instead they are carefully computed perturbations that possess the property of transferability. Adversarial examples crafted for a particular model *transfer* to other models, which means that the same adversarial examples can be applied to different models to cause mistakes Szegedy et al. (2014). In particular, adversarial examples generated against a neural network can fool other neural networks with the same architecture, but trained on different datasets Szegedy et al. (2014). It has also been shown that adversarial examples can also fool neural networks with different architectures and even models trained with different machine learning algorithms Papernot et al. (2016). Therefore, adversarial examples must exploit some systematic issue or property of neural nets.

There are several hypothesis attempting to explain the existence of adversarial examples and the phenomenon of adversarial vulnerability of machine learning models. According to Goodfellow et al. (2014) the reason why neural nets can be exploited with adversarial examples is excessive linearity of neural net models. However, it is not only neural nets that are susceptible to attacks, so further aspects may play a role. Gilmer et al. (2018), Fawzi et al. (2018), Mahloujifar et al. (2019), and Shafahi et al. (2019) argue that it is the high dimensionality of the input space that prevents a classifier from learning a robust model that would be resilient to adversarial examples. On the other hand, Schmidt et al. (2018) argued that adversarial examples arise due to insufficient information about the true data distribution. Tanay and Griffin (2016) propose that overfitting causes adversarial examples and suggest to utilize regularization as a form of defense. Another possible explanation, proposed by Fawzi et al. (2016) and Ford et al. (2019), pertains to noise in data and maintains that a classifier's robustness to noise determines the extent of its adversarial vulnerability. Shamir et al. (2019) suggests that the piecewise-linear geometric structure of decision boundaries leads to adversarial perturbations. Bubeck et al. (2019) and Nakkiran (2019) argue that computational constraints or model complexity cause the model to learn non-robust features, which means that the resulting model may be accurate on benign test data, but highly vulnerable to adversarial inputs. Finally, Ilyas et al. (2019) claim that adversarial examples can be attributed

to the presence of non-robust features. They define non-robust features as "highly predictive, yet brittle" features, which are well-generalizing on benign data, but vulnerable to adversarial perturbations. They show that robust training (training on data containing only robust features) results in classifiers resilient to adversarial attacks. However, training on non-robust features can help to increase the model accuracy on unmodified non-adversarial data. Therefore, they present accuracy and robustness as almost a trade-off and claim that "robustness can be at odds with accuracy". Hence, the reasons for the existence and common widespread presence of adversarial examples is still an active research issue that not fully understood.

2.2 Adversarial Reprogramming

Adversarial reprogramming is a new type of attack introduced in Elsayed et al. (2019), which reprograms a target model to perform a different task chosen by the attacker. There are two primary differences between prior adversarial attacks and reprogramming in terms of their goals and methodology. First, the goal of reprogramming is to reprogram a target model to perform a different task, while the goal of adversarial examples is to degrade performance of the model. Second, reprogramming is achieved by finding a single adversarial perturbation, an adversarial program that can be added to all inputs without the need for crafting many different adversarial examples and computing their outputs.

Reprogramming does not require modifications to the target network architecture or parameters. Instead, only two reprogramming functions must be learned to map the inputs and outputs between the two domains of the new and original task. The motivation of reprogramming is to reprogram an existing model in a computationally efficient way to perform a new task. Computational efficiency is key – if the desired results were achievable using a computationally inexpensive classifier created from scratch specifically for that task, this would defeat the purpose of reprogramming Neekhara et al. (2019). It has been shown that adversarial reprogramming is significantly less effective on randomly initialized untrained networks (Elsayed et al. (2019) and Neekhara et al. (2019), which is evidence that reprogramming works.

2.2.1 Motivations for Reprogramming

There is a concern that reprogramming could be used for malicious purposes such as theft of computational resources through attacks on cloud-hosted machine learning models, which could be maliciously re-purposed by an attacker to, for example, create spam accounts. Such an attack should be a great concern to cloud providers offering machine learning APIs such as Google or Amazon. Nevertheless, despite the fact that the current literature on adversarial reprogramming focused on the security concerns created by adversarial reprogramming, we would like to argue that reprogramming does not need to be adversarial. Reprogramming has a great potential for developing high quality models at a significantly reduced computational cost.

2.2.2 Reprogramming Methods

Adversarial reprogramming and its methods were first introduced in Elsayed et al. (2019) and later used by Neekhara et al. (2019). Elsayed et al. (2019) assume a white-box scenario and their reprogramming method is based on crafting an adversarial program, which they formulate as an additive contribution to network input. The adversarial program can be viewed as a "universal" adversarial perturbation that can be applied to all inputs, whose parameters are learned through backpropagation. The goal is to repurpose an existing trained target model with inputs x and outputs f(x) as seen in Figure 2.1 for a new task with inputs x' and outputs g(x') as illustrated in Figure 2.2.

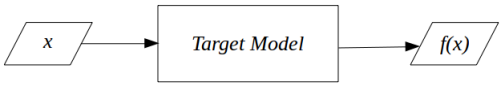

Figure 2.1 Original Setting

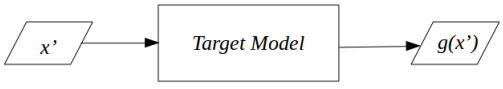

Figure 2.2 New Setting

2.2.2.1 Adversarial Program Crafting

In this section we describe the adversarial program crafting methodology from Elsayed et al. (2019) with additional details and explanations. Let $x \in \mathbb{R}^{n \times n \times h}$ be an input image for the original task, where n is the image width and height, and h is the number of channels. Note that grayscale images have one channel and color images have three channels. Let $f(x)$ be an output of the original task. The new task sample input is defined as $x' \in \mathbb{R}^{k \times k \times h}$, where $k < n$. The output of the new task is represented as $g(x')$. In order to feed the new inputs into the original target model and receive outputs that are of the new task's domain as illustrated in Figure 2.3, two mapping functions are required h_f and h_g. The function h_f maps x' into the domain of x, i.e.: $h_f(x', \theta) = x$. And h_g maps $f(x)$ to $g(x')$, i.e.: $h_g(f(x), \theta) = g(x')$.

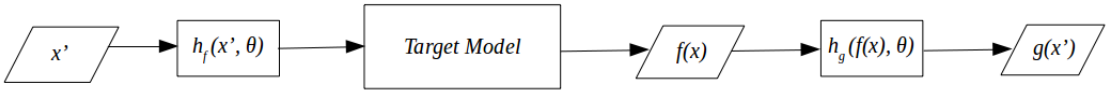

Figure 2.3 Reprogramming Process

The goal is to learn θ such that $h_g(f(h_f(x'))) = g(x')$. Prior to learning θ the label mapping function h_g is defined as a hard-coded one-to-one mapping function that can convert $f(x)$ to $g(x')$. Let P be the adversarial program, which is applied to all image inputs **x'** be defined as:

$$P = \tanh(\theta \odot M) \tag{2.4}$$

where tanh scales the adversarial program to in range of (-1, 1), $\theta \in \mathbb{R}^{n \times n \times h}$ are the parameters of P, and M is a $n \times n \times h$ masking matrix $\in 0, 1$. Let x_{adv} be the image input after conversion of x' into the domain of x:

$$x_{adv} = h_f(x', \theta) \tag{2.5}$$

The adversarial program P is an additive variable applied to every input x' as follows:

$$x_{adv} = \mathbf{x'} + P \tag{2.6}$$

Let y_{adv} be the label of x_{adv}. The goal is:

$$max\ P(h_g(y_{adv})|x_{adv}) \tag{2.7}$$

which can be derived as follows:

$$P(y'|x') = P(g(x')|x') = P(h_g(g(x'))|h_f(x',\theta)) = P(h_g(y_{adv})|h_f(x',\theta)) = P(h_g(y_{adv})|x_{adv})) \tag{2.8}$$

The probability $P(h_g(y_{adv})|x_{adv})$ can be maximized by solving the following optimization problem:

$$\begin{aligned}
\theta^* &= argmax_{\theta \in \mathbb{R}^{n \times n \times h}}(P(h_g(y_{adv})|h_f(x',\theta)) \\
\theta^* &= argmin_\theta(-P(h_g(y_{adv})|h_f(x',\theta)) \\
\theta^* &= argmin_\theta(-logP(h_g(y_{adv})|h_f(x',\theta)) \\
\theta^* &= argmin_\theta(-logP(h_g(y_{adv})|h_f(x',\theta) + \lambda||\theta||_2^2))
\end{aligned} \tag{2.9}$$

where λ is a coefficient that serves as a regularization hyperparameter, which is multiplied by $L2$ norm squared of θ. Therefore, the adversarial program can crafted by solving the optimization problem $argmin_\theta(-logP(h_g(y_{adv})|h_f(x',\theta) + \lambda||\theta||_2^2)$.

2.2.2.2 Reprogramming of Image Classification Neural Networks

Elsayed et al. (2019) conducted reprogramming on image classification pre-trained neural networks. Their idea for reprogramming is based on learning adversarial reprogramming functions for mapping inputs and outputs between different domains. In particular, given a trained target model that performs some original task that takes x as input and produces f(x) as output and a new task that takes x' as input and outputs g(x'), two mapping functions are required:

$$h_f(\cdot;\theta), where\ h_f\ maps\ x'\ into\ the\ domain\ of\ x$$

$$h_g(\cdot;\theta), where\ h_g\ maps\ f(x)\ into\ the\ domain\ of\ g(x')$$

Elsayed et al. (2019) assume a white-box scenario, which allows them to use backpropagation to learn h_f, which converts the new inputs into the domain of the original inputs that the target model was trained for. They do not learn the label mapping function h_g, which converts the outputs of the

target model into the outputs of the new task. They simply hard-code some arbitrary mapping that establishes a correspondence between the original and new labels. Elsayed et al. (2019) conduct experiments with six different pretrained ImageNet models, which they reprogram to perform new tasks - counting number of squares in an image, MNIST handwritten digits classification, and CIFAR-10 image classification. The accuracy achieved on reprogramming pretrained Imagenet models to classify handwritten digits is impressive - the accuracy is 90% and above on both training and testing data. However, the accuracy of the pretrained networks reprogrammed to perform CIFAR-10 image classification is a lot lower - 65% on average for training and testing data.

2.2.2.3 Reprogramming of Text Classification Neural Networks

Neekhara et al. (2019) extends adversarial reprogramming from image to text classification tasks. They propose a context-based vocabulary remapping function, which is learned through backpropagation as in the case of image classifiers. However, due to the fact that text is discrete, they apply Gumbel-Softmax (Jang et al., 2016) to make the optimization problem differentiable. Neekhara et al. (2019) also perform a black-box attack using reinforcement learning with the reprogramming function being the policy network. They conducted experiments using 5 different text datasets and three victim models: LSTM, Bi-LSTM, and CNN. The results achieved seem encouraging in both white-box and black-box setting, which on average yielded a testing accuracy of 80% and 90% respectively.

2.2.2.4 Defenses against Adversarial Reprogramming

Wang et al. (2019) claim to have developed the first effective defense against adversarial reprogramming called Hierarchical Random Switching, a new stochastic defense method which they test on image classification models. Stochastic defenses such as dropout or adding Gaussian perturbations aim to improve adversarial robustness through useful randomness injected during inference with the goal to make adversarial examples crafting difficult. In essence, Hierarchical Random Switching adds several blocks of randomly switching channels derived from a base neural network

model to prevent adversaries from exploiting fixed model structures and parameters. Similarly to Ilyas et al. (2019), Wang et al. (2019) argue that the drawback of current defenses is that the robustness enhancement is at the cost of noticeable performance degradation on legitimate data. With the goal to alleviate this shortcoming Wang et al. (2019) propose a new metric called Defense Efficiency Score that measures the gain in unsuccessful attack attempts at the cost of drop in test accuracy. They claim that Hierarchical Random Switching yields higher Defense Efficiency Score than current stochastic defences and therefore achieves better robustness accuracy trade-off. The defense technique proposed by Wang et al. (2019) seems compelling. However, we would like to argue that we will not know what are the most appropriate defenses against reprogramming or other types of attacks on machine learning models until the causes for adversarial vulnerabilities are fully explained.

CHAPTER 3. EXPERIMENTS WITH PRIOR REPROGRAMMING METHOD

In this section we examine the reprogramming method introduced in Elsayed et al. (2019) in more detail. First, we apply the method in new scenarios to test if this method also works for datasets other than the ones tested in Elsayed et al. (2019). Second, we attempt tweaking this method in various ways to investigate whether the shortcomings of this method discovered in the first set of experiments can be addressed with small changes to the method.

3.1 Experiment 1: Prior Reprogramming Method

We applied the reprogramming method introduced in Elsayed et al. (2019) to various datasets and scenarios that have not been explored and found that this method does not work reliably, is not scaleable, and cannot be applied to more complex tasks as evidenced in Table 3.1. Elsayed et al. (2019) successfully reprogrammed six different models trained on ImageNet to perform three relatively simple tasks: counting number of squares in an image, MNIST handwritten digits classification, and CIFAR 10 image classification. However, when the target model is trained on more simpler datasets, the accuracy of the reprogrammed model on the new tasks is equivalent to random guessing. In addition to that, reprogramming Imagenet models using this technique to perform more complex tasks such as CIFAR 100 or Caltech 101 does not work well. Moreover, these operations are quite computationally expensive and therefore, not practical - using Nvidia GPU GeForce GTX 1080 it took approximately 12 hours to reprogram a Resnet50V2 Imagenet model to perform MNIST. Based on the results there seems to be a relationship between the reprogramming performance and the nature of the new task - as the input dimension and number of categories get larger, reprogramming becomes less accurate and more computationally expensive.

Table 3.1 Reprogramming Experiments with Elsayed et al. (2019) Methodology

Original Task	**New Task**	**Train Accuracy**	**Test Accuracy**
Imagenet[2]	MNIST	93.69 %	94.36 %
Imagenet	CIFAR 100	1.16 %	1.21 %
Imagenet	Caltech 100	0.40 %	0.30 %
Caltech 100[3]	MNIST	9.87 %	9.61 %
CIFAR 10[4]	MNIST	9.03 %	8.92 %
CIFAR 10	FASHION MNIST	14.32 %	14.25 %
CIFAR 10	MNIST 0s and 1s	53.17 %	53.24 %
MNIST FASHION[5]	MNIST	12.08 %	12.30 %
MNIST FASHION	MNIST 0s and 1s	42.82 %	43.83 %
MNIST[6]	MNIST FASHION	14.66 %	14.94 %

3.2 Experiment 2: Tweaking Prior Reprogramming Method

After concluding that the reprogramming method introduced in Elsayed et al. (2019) does not work reliably on all datasets we attempted tweaking their methodology in different ways, but none of them yielded better results. First, we experimented with learning the label mapping function as opposed to hardcoding it. Second, we tried resizing the new task input image to smaller dimensions to achieve a higher ratio between the original and new input dimensions. In particular, we took a model pre-trained on MNIST FASHION with input dimension 28x28x1 and reprogrammed it to classify resized handwritten 1s and 0s digits with input dimension 4x4x1. Third, we investigated whether changing the area of the image that is being trained makes a difference. The method in Elsayed et al. (2019) trains the adversarial program only on the part of the image, where the new and original image don't overlap. We tried training the entire area; however, this change also did not yield better results. Finally, we attempted changing the operations used to apply the adversarial program to the the new task input image. Specifically, we experimented with multiplying and subtracting the program from the input image as opposed to adding it as in Elsayed et al. (2019).

[2]Pretrained Resnet50V2 model
[3]Train Accuracy: 99.58 %, Test Accuracy: 74.37 %
[4]Train Accuracy: 92.12 %, Test Accuracy: 80.18 %
[5]Train Accuracy: 99.99 %, Test Accuracy: 89.40 %
[6]Train Accuracy: 100.00 %, Test Accuracy: 99.25 %

CHAPTER 4. PROPOSED METHODOLOGY

The shortcomings discovered in the first two sets of experiments with the reprogramming methodology used in prior works motivated us to explore new techniques for reprogramming and lead to proposing a new and improved methodology for reprogramming.

Our proposed methodology assumes access to target model parameters and architecture, which is reprogrammed to perform a new chosen task. The objective is to learn input and output reprogramming functions that allow for re-using a trained model without changing the original model parameters to create new model performing a new task. Prior works focused on crafting an adversarial program that served as additive contribution to inputs and learning the adversarial program parameters while hardcoding an arbitrary mapping between to original and new labels. This can be expressed as:

$$x + P, \tag{4.1}$$

where x is the new task inputs and P is the adversarial program. Our improved technique does not rely on simple constant additive contribution to inputs and arbitrarily hardcoding label mappings, instead it is based on learning two programs/functions - one applied to inputs and one to outputs. The first program applied to the new task inputs x can be viewed as a conversion or input adjustment function that outputs x', an adjusted input that can be passed into the target model, i.e.:

$$x' = f(x) \tag{4.2}$$

The purpose of function f is to resize the input image such that its dimensions match the ones that the target model accepts. The second program applied to the target model outputs is represented as a new dense layer containing a number of neurons equal to number of new task labels and a softmax activation function.

The softmax function takes an input vector of k real numbers and outputs k probabilities that sum up to 1, where k is the total number of labels. This can be expressed as:

$$softmax(\mathbf{z})_i = exp(z_i)/\sum_j exp(z_j), \tag{4.3}$$

where z_i is the unnormalized log probability that $\mathbf{x}$ belongs to class i:

$$z_i = log\tilde{P}(y = i|\mathbf{x}) \tag{4.4}$$

The original target model parameters are not changed, only the last dense layer that outputs probabilities is removed. Therefore, the second program can be expressed as a function g:

$$y = g(NN'(x')), \tag{4.5}$$

where y' is the new task label and NN' is the original model with the last layer removed.

The goal of a neural is to approximate some function f^*, in case of classification mapping function $f^*(x) = y$, which maps input $\mathbf{x}$ to class y. The objective is to find function f that approximates the true function f^*. Therefore the mapping can be expressed as $y = f(\mathbf{x};\theta)$ and the goal is to learn θ which represents the neural net weights and biases that results in best approximation of f^*. A deep neural net is a chain of functions, where each function represents one layer:

$$f(\mathbf{x}) = f^{(L)}(f^{(L-1)}...(f^{(1)}(\mathbf{x}))) \tag{4.6}$$

Training a neural net from scratch requires learning weights and biases for all functions $f^{(1)}$,...$f^{(L)}$. However, our reprogramming method allows us to re-use θ from a previously trained target model for layers $1, ..., L-1$ and learn the parameters of the last layer only, which can be formulated as:

$$y = f^{(L)}(h;\theta), \tag{4.7}$$

where h are the values of hidden units from the penultimate layer.

Therefore, our goal is to maximize probability $P(y|f^{(l)}(f^{(l-1)}))$, which can be set up as an optimization problem:

$$\hat{\theta} = argmin_\theta(-logP(y|h)) \tag{4.8}$$

CHAPTER 5. RESULTS OF PROPOSED METHODOLOGY

This section describes the experiments conducted and their results.

5.1 Experiment 3: New and Improved Reprogramming Method

To demonstrate effectiveness and superiority of our reprogramming method, we conducted experiments on eight different architectures trained on ImageNet, which served as the target model and were reprogrammed to perform new tasks: Caltech 101 and Caltech 256 - reduced image classification. As evidenced in Table 5.1 our reprogramming method yields higher accuracy than prior works even on more complex tasks. We also trained models on Caltech 101 and Caltech 256 - reduced datasets from scratch with the same hyperparameters and number of training epochs to compare the effectiveness of training a model utilizing our reprogramming method and training a model from scratch.

Using our methodology we successfully reprogrammed various pre-trained ImageNet models to perform classification on Caltech 101 and Caltech 256 - reduced datasets. It can be seen that the testing accuracy of models trained from scratch is significantly lower than of models trained using our reprogramming technique. It is possible to create models from scratch with much higher accuracy than the one we report; however, this requires increased training time and careful hyperparamenter tuning. The purpose of comparing reprogrammed models and models trained from scratch is to compare the two techniques in terms of the effort spent to create them. In both cases we trained models only for 10 epochs with the same hyperparameters contained in the appendix.

Most models created with reprogramming performed very well - the average testing accuracy on Caltech 101 was 82.07 % with the best testing accuracy of 89.81 % achieved by reprogramming ResNet152V2. This is a significantly better result compared to the model created from scratch using the same hyperparameters, which achieved only 38.95 % testing accuracy on Caltech 101.

The Caltech 256 models created through our reprogramming method also performed significantly better than the model made from scratch train with the same hyperparameters. The average testing accuracy of the models made using reprogramming was 55.95 % with the best one of 69.65 % created through reprogramming ResNet152V2, whereas the testing accuracy of the model made from scratch was only 3.63 %. While an average testing accuracy of 55.95 % is not very high, it should be noted that the training dataset was reduced by 83 % and that the testing accuracy of the model created from scratch using the same hyperparameters was a mere 3.63 %.

Overall, these results demonstrate that: (1) our reprogramming method is superior to the prior method, and (2) that our reprogramming method can be used to quickly create new accurate models without requiring much training time and hyperparameter tuning.

Table 5.1 Reprogramming Experiments with New and Improved Methodology

Original Task	**New Task**			
Imagenet	**Caltech 101**		**Reduced Caltech 256**	
architecture	train	test	train	test
ResNet50V2	98.16%	88.73%	98.39%	66.93%
ResNet101V2	98.48%	89.48%	98.36%	70.04%
ResNet152V2	98.14%	89.81%	97.75%	69.65%
MobileNet	96.94%	83.05%	95.89%	53.96%
NASNetMobile	93.19%	82.73%	92.19%	67.06%
MobileNetV2	93.47%	83.05%	91.58%	60.05%
VGG16	80.79%	71.14%	54.10%	29.96%
VGG19	80.44%	68.56%	55.02%	29.96%
From scratch	85.16%	38.95%	27.47%	3.63%

CHAPTER 6. SUMMARY

The focus of this paper on reprogramming of neural networks, which involves re-purposing a target model to perform a new task without changing its parameters. We introduce new reprogramming methodology that improves upon Elsayed et al. (2019) and demonstrate that this method, compared to prior works, yields more accurate results, is more scalable, and can be applied to complex tasks. The first two sets of experiments focus on exploring reprogramming methodology from prior literature and show that, among other shortcomings, it does not scale well to more complex tasks. The last set of experiments demonstrates the superiority of our new and improved reprogramming method.

BIBLIOGRAPHY

Barreno, M., Nelson, B., Joseph, A. D., and Tygar, J. D. (2010). The security of machine learning. *Machine Learning*, 81:121–148.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. (2006). Can machine learning be secure? In *ASIACCS '06*.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Srndic, N., Laskov, P., Giacinto, G., and Roli, F. (2013). Evasion attacks against machine learning at test time. In *ECML/PKDD*.

Biggio, B., Nelson, B., and Laskov, P. (2012). Poisoning attacks against support vector machines. In *ICML*.

Bubeck, S., Price, E., and Razenshteyn, I. P. (2019). Adversarial examples from computational constraints. In *ICML*.

Carlini, N., and Wagner, D. A. (2016). Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy*, pages 39–57.

Elsayed, G. F., Goodfellow, I. J., and Sohl-Dickstein, J. (2019). Adversarial reprogramming of neural networks. *ICLR*.

Fawzi, A., Fawzi, H., and Fawzi, O. (2018). Adversarial vulnerability for any classifier. In *NeurIPS*.

Fawzi, A., Moosavi-Dezfooli, S.-M., and Frossard, P. (2016). Robustness of classifiers: from adversarial to random noise. In *NIPS*.

Ford, N., Gilmer, J., Carlini, N., and Cubuk, E. D. (2019). Adversarial examples are a natural consequence of test error in noise. *Proceedings of the 36th International Conference on Machine Learning*.

Gilmer, J., Metz, L., Faghri, F., Schoenholz, S. S., Raghu, M., Wattenberg, M., and Goodfellow, I. J. (2018). Adversarial spheres. *ICLR*.

Goodfellow, I. J., McDaniel, P. D., and Papernot, N. (2018). Making machine learning robust against adversarial inputs. *Commun. ACM*, 61:56–66.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *ICLR*.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial examples are not bugs, they are features. In *NeurIPS*.

Mahloujifar, S., Diochnos, D. I., and Mahmoody, M. (2019). The curse of concentration in robust learning: Evasion and poisoning attacks from concentration of measure. In *AAAI*.

Nakkiran, P. (2019). Adversarial robustness may be at odds with simplicity. *ICLR*.

Neekhara, P., Hussain, S., Dubnov, S., and Koushanfar, F. (2019). Adversarial reprogramming of text classification neural networks. In *EMNLP/IJCNLP*.

Papernot, N., McDaniel, P. D., Goodfellow, I. J., Jha, S., Celik, Z. B., and Swami, A. (2016). Practical black-box attacks against machine learning. In *ASIA CCS '17*.

Papernot, N., McDaniel, P. D., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2015). The limitations of deep learning in adversarial settings. *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387.

Schmidt, L., Santurkar, S., Tsipras, D., Talwar, K., and Madry, A. (2018). Adversarially robust generalization requires more data. In *NeurIPS*.

Shafahi, A., Huang, W. R., Studer, C., Feizi, S., and Goldstein, T. (2019). Are adversarial examples inevitable? *ICLR*.

Shamir, A., Safran, I., Ronen, E., and Dunkelman, O. (2019). A simple explanation for the existence of adversarial examples with small hamming distance. *ArXiv*, abs/1901.10861.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014). Intriguing properties of neural networks. *2nd International Conference on Learning Representations*.

Tanay, T., and Griffin, L. D. (2016). A boundary tilting persepective on the phenomenon of adversarial examples. *ArXiv*, abs/1608.07690.

Wang, X., Wang, S., Chen, P.-Y., Wang, Y., Kulis, B., Lin, X., and Chin, S. P. (2019). Protecting neural networks with hierarchical random switching: Towards better robustness-accuracy trade-off for stochastic defenses. *ProceedingsoftheTwenty-EighthInternationalJointConferenceonArtificialIntelligence(IJCAI)*, page 6013–6019.

APPENDIX. SUPPLEMENTAL MATERIAL

Link to code implementation: https://github.com/ekloberdanz/Thesis/tree/master/FinalCode

Table .1 Hyperparameters used for reprogramming and creating models from scratch

Hyperparameter	**Value**
batch size	128
number of epochs	10
optimizer	adam
learning rate	0.001

The datasets used in our experiments along with their descriptions are listed in Table .2.

Table .2 Datasets

Dataset Name	**Number of Categories**	**Input Dimensions**	**Description**
Caltech 256 - reduced[1]	256	224 x 224 x 3	Various images
Caltech 101	101	224 x 224 x 3	Various images
CIFAR 100	100	32 x 32 x 3	Various images
CIFAR 10	10	32 x 32 x 3	Various images
MNIST FASHION	10	28 x 18 x 1	Clothing items
MNIST	10	28 x 18 x 1	Handwritten digits
MNIST 0s and 1s	2	28 x 18 x 1	Handwritten digits

[1]The Caltech 256 dataset was reduced from 30,607 images to only 5,140 images