Adversarial Reprogramming Revisited

Anonymous Author(s)
Affiliation
Address
email

Abstract

Adversarial reprogramming, introduced by Elsayed, Goodfellow, and Sohl-Dickstein, seeks to repurpose a neural network to perform a different task, by manipulating its input without modifying its weights. We prove that two-layer ReLU neural networks with random weights can be adversarially reprogrammed to achieve arbitrarily high accuracy on Bernoulli data models over hypercube vertices, provided the network width is no greater than its input dimension. We also substantially strengthen a recent result of Phuong and Lampert on directional convergence of gradient flow, and obtain as a corollary that training two-layer ReLU neural networks on orthogonally separable datasets can cause their adversarial reprogramming to fail. We support these theoretical results by experiments that demonstrate that, as long as batch normalisation layers are suitably initialised, even untrained networks with random weights are susceptible to adversarial reprogramming. This is in contrast to observations in several recent works that suggested that adversarial reprogramming is not possible for untrained networks to any degree of reliability.

1 Introduction

Elsayed, Goodfellow, and Sohl-Dickstein [2019] proposed *adversarial reprogramming*: given a neural network $\mathcal{N}$ which performs a task $F : X \to Y$ and given an adversarial task $G : \widetilde{X} \to \widetilde{Y}$, repurpose $\mathcal{N}$ to perform G by finding mappings $h_{\text{in}} : \widetilde{X} \to X$ and $h_{\text{out}} : Y \to \widetilde{Y}$ such that $G \approx h_{\text{out}} \circ F \circ h_{\text{in}}$. They focused on a setting where F and G are image classification tasks, X consists of large images, $\widetilde{X}$ consists of small images, and h_{in} and h_{out} are simple and computationally inexpensive: $h_{\text{in}}(\widetilde{x}) = p + \widetilde{x}$ draws the input $\widetilde{x}$ at the centre of the *adversarial program* p, and h_{out} is a hard coded mapping from the class labels Y to the class labels $\widetilde{Y}$. Then the challenge of adversarial reprogramming is to find p such that $h_{\text{out}} \circ F \circ h_{\text{in}}$ approximates well the adversarial task G.

Remarkably, Elsayed et al. [2019] showed that a range of realistic neural networks can be adversarially reprogrammed to perform successfully several tasks. They considered six architectures trained on ImageNet [Russakovsky, Deng, Su, Krause, Satheesh, Ma, Huang, Karpathy, Khosla, Bernstein, Berg, and Fei-Fei, 2015] and found adversarial programs that achieve very good accuracies on a counting task, MNIST [LeCun, Bottou, Bengio, and Haffner, 1998], and CIFAR-10 [Krizhevsky, 2009]. In addition to the basic setting, they investigated limiting the visibility of the adversarial program by restricting its size or scale, or even concealing both the input and the adversarial program by hiding them within a normal image from ImageNet, and obtained good results.

Adversarial reprogramming can be seen as taking the crafting of adversarial examples [Biggio, Corona, Maiorca, Nelson, Šrndić, Laskov, Giacinto, and Roli, 2013, Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, and Fergus, 2014] to a next level. Like the universal adversarial perturbation of Moosavi-Dezfooli, Fawzi, Fawzi, and Frossard [2017], a single adversarial program is combined with every input from the adversarial task, but in contrast to the former where the goal is to cause

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

natural images to be misclassified with high probability, adversarial reprogramming seeks a high accuracy of classification for the given adversarial task.

Adversarial reprogramming is also related to transfer learning [Raina, Battle, Lee, Packer, and Ng, 2007, Mesnil, Dauphin, Glorot, Rifai, Bengio, Goodfellow, Lavoie, Muller, Desjardins, Warde-Farley, Vincent, Courville, and Bergstra, 2011], however with important differences. Whereas transfer learning methods use the knowledge obtained from one task as a base for learning to perform another, and allow model parameters to be changed for the new task, in adversarial reprogramming the new task may bear no similarity with the old, and the model cannot be altered but has to be manipulated through its input.

The latter features are suggestive of potential nefarious uses of adversarial reprogramming, and Elsayed et al. [2019] list several, such as repurposing computational resources to perform a task which violates the ethics code of system providers. However, its further explorations have demonstrated utility for virtuous deployments to medical image classification [Tsai, Chen, and Ho, 2020], natural language processing [Hambardzumyan, Khachatrian, and May, 2021, Neekhara, Hussain, Dubnov, and Koushanfar, 2019, Neekhara, Hussain, Du, Dubnov, Koushanfar, and McAuley, 2022], molecule toxicity prediction [Vinod, Chen, and Das, 2020], and time series classification [Yang, Tsai, and Chen, 2021]. In the last three works, adversarial reprogramming was achieved between different domains, e.g. repurposing neural networks trained on ImageNet to perform classification of DNA sequences, and of natural language sentiments and topics.

In spite of the variety of fruitful applications, it is still largely a mystery when adversarial reprogramming is possible and why. In the only work in this direction, Zheng, Feng, Xia, Jiang, Demontis, Pintor, Biggio, and Roli [2021] proposed the alignment $\|g\|_1/(1/n \sum_i \|g_i\|_1)$ of the gradients g_i of the inputs from the adversarial task with their average g as the main indication of whether adversarial reprogramming will succeed. However, their experiments did not show a significant correlation between accuracy after reprogramming and the gradient alignment before reprogramming. The correlation was statistically significant with the gradient alignment after reprogramming, but that is unsurprising and it is unclear how to use it for predicting success.

Specifically, a central question on adversarial reprogramming is:

> Can neural networks with random weights be adversarially reprogrammed, and more generally, how does training impact adversarial reprogrammability?

First, addressing this question is important for assessing scope of two claims made in the literature:

***"[Adversarial] reprogramming usually fails when applied to untrained networks"* [Zheng et al., 2021].** In addition to networks trained on ImageNet, Elsayed et al. [2019] experimented using the same architectures untrained, i.e. with random weights, and obtained generally poor results. Similarly, the experimental results of Neekhara et al. [2019] and Zheng et al. [2021] for random networks are significantly worse than their experimental results for trained networks. However, they remarked that this was surprising given the known richness of random networks (see e.g. He, Wang, and Hopcroft [2016c], Lee, Bahri, Novak, Schoenholz, Pennington, and Sohl-Dickstein [2018]), and that it was possibly due to simple reasons such as poor scaling of the random weights.

***"The original task the neural networks perform is important for adversarial reprogramming"* [Elsayed et al., 2019].** Nevertheless a number experimental results including those of Tsai et al. [2020], Neekhara et al. [2022], Vinod et al. [2020], Yang et al. [2021], Zheng et al. [2021] demonstrated successful adversarial reprogramming between tasks that are seemingly unrelated (e.g. repurposing networks trained on ImageNet to act as HCL2000 [Zhang, Guo, Chen, and Li, 2009] classifiers) or from different domains, so the interaction between the original and adversarial tasks remains unclear.

Second, the question above is important because of implications for the following two applied considerations:

Disentangling architecture and training as factors in adversarial reprogrammability. Clarifying the respective bearings on adversarial reprogramming success of the network architecture and of the task (if any) it was trained for would improve decision making, either to maximise the success in beneficial scenarios or to minimise it in detrimental ones.

Managing the cost of adversarial reprogramming. Understanding when training the network is not essential, and when training for longer does not help or even hinders adversarial reprogrammability, should make it possible to control better the economic and environmental costs.

1.1 Our contributions

We initiate a theoretical study of adversarial reprogramming. In it we focus on two-layer neural networks with ReLU activation, and on adversarial tasks given by the Bernoulli distributions of Schmidt, Santurkar, Tsipras, Talwar, and Mądry [2018] over hypercube vertices. The latter are binary classification data models in which the two classes are represented by opposite hypercube vertices, and when sampling a data point for a given class, we flip each coordinate of the corresponding class vertex with a certain probability. This probability is a parameter that controls the difficulty of the classification task. These data models are inspired by the MNIST dataset because MNIST images are close to binary (many pixels are almost fully black or white). The remaining parameters are the radius of the hypercube, the input dimension of the neural network, and its width.

We prove that, in this setting, for networks with random weights, adversarial programs exist that achieve arbitrarily high accuracy on the Bernoulli adversarial tasks. This holds for a wide variety of parameter regimes, provided the network width is no greater than its input dimension. The adversarial programs we construct depend on the weights of the network and on the class vertices (i.e. the direction) of the Bernoulli data model, and their Euclidean length is likely to be close to the square root of the input dimension. We present these results in Section 2.

We also prove that, in the same setting, training the network on orthogonally separable datasets can cause adversarial reprogramming to fail. Phuong and Lampert [2021] recently showed that, under several assumptions, training a two-layer ReLU network on such datasets by gradient flow makes it converge to a linear combination of two maximum-margin neurons; and subsequently Wang and Pilanci [2022] obtained in a different manner the same conclusion under the same assumptions. We provide a simpler proof of a significantly stronger result: we show that the assumptions in Phuong and Lampert [2021] and Wang and Pilanci [2022] of small initialisation, and of positive and negative support examples spanning the whole space, are not needed; and we generalise to the exponential loss function as well as the logistic one. We then observe that, for any Bernoulli data model whose direction is in a half-space of the difference of the maximum-margin neurons, and for any adversarial program, the accuracy tends to $1/2$ (i.e. approaches guessing) under a mild assumption on the growth rate of the difficulty of the data model. We present these results in Section 3.

Both for the neworks with random weights and for the networks trained for long on orthogonally separable datasets, we then show that similar theoretical results can be obtained with a different kind of adversarial task, namely those given by the Gaussian distributions also of Schmidt et al. [2018]. The latter are mixtures of two spherical multivariate Gaussians, one per data class. Please see the appendix.

In the experimental part of our work, we demonstrate that, as long as batch normalisation layers are suitably initialised, even untrained networks with random weights are susceptible to adversarial reprogramming. Both the random weighs and the batch normalisation layers are kept fixed throughout the finding of adversarial programs and their evaluation. Our experiments are conducted with six realistic network architectures and MNIST as the adversarial task. We investigate two different schemes to combine input images with adversarial programs: replacing the centre of the program by the image as was done by Elsayed et al. [2019], and scaling the image to the size of the program and then taking a convex combination of the two. Each of the two schemes has a ratio parameter, and we explore their different values. We find that the second scheme gives better results in our experiments, and that for some choices of the ratio parameter, the accuracies on the test set across all six architectures are not far below what Elsayed et al. [2019] reported for networks trained on ImageNet. Please see Section 4.

We conduct the same experiments also on the more challenging Fashion-MNIST [Xiao, Rasul, and Vollgraf, 2017] and Kuzushiji-MNIST [Clanuwat, Bober-Irizar, Kitamoto, Lamb, Yamamoto, and Ha, 2018] datasets, and obtain broadly similar results, however with lower test accuracies in several cases; they are reported in the appendix.

For a further discussion of relations with other works, please see the appendix.

We conclude the paper in Section 5, where we discuss limitations of our work and suggest directions for future work.

2 Random networks

Basic notations. We write: $[n]$ for the set $\{1, \ldots, n\}$, $\|\boldsymbol{v}\|$ for the Euclidean length of a vector $\boldsymbol{v}$, $\angle(\boldsymbol{v}, \boldsymbol{v}')$ for the angle between $\boldsymbol{v}$ and $\boldsymbol{v}'$, and $\mathbb{H}^d$ for the d-dimensional unit hypercube $\{\pm 1/\sqrt{d}\}^d$.

Two-layer ReLU networks. We consider two-layer neural networks $\mathcal{N}$ with the ReLU activation. We write d for the input dimension, k for the width, $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k \in \mathbb{R}^d$ for the weights of the first layer, and $a_1, \ldots, a_k \in \mathbb{R}$ for the weights of the second layer. For an input $\boldsymbol{x} \in \mathbb{R}^d$, the output is thus

$$\mathcal{N}(\boldsymbol{x}) \coloneqq \sum_{j=1}^{k} a_j \psi(\boldsymbol{w}_j^\top \boldsymbol{x}) \ ,$$

where $\psi(u) = \max\{u, 0\}$ is the ReLU function.

Random weights. In this section, we assume that the weights in $\mathcal{N}$ are random as follows:

- each $\boldsymbol{w}_j$ consists of d independent centred Gaussians with variance $1/d$, and
- each a_j is independently uniformly distributed in $\{\pm 1/\sqrt{k}\}$.

This distribution is as in Bubeck, Cherapanamjeri, Gidel, and Tachet des Combes [2021], and standard for theoretical investigations. It is similar to He's initialisation [He, Zhang, Ren, and Sun, 2015], with the second layer discretised for simplicity.

The variances of the weights are such that, for any input $\boldsymbol{x} \in \mathbb{R}^d$ of Euclidean length $\sqrt{d}$, each $\boldsymbol{w}_j^\top \boldsymbol{x}$ is a standard Gaussian, and for large widths k the distribution of $\mathcal{N}(\boldsymbol{x})$ is close to centred Gaussian with variance $1/2$.

Bernoulli data models. Adapting from Schmidt et al. [2018], given a hypercube vertex $\boldsymbol{\phi} \in \mathbb{H}^d$, a radius $\rho > 0$, and a class bias parameter $0 < \tau \leq 1/2$, we define the $(\boldsymbol{\phi}, \rho, \tau)$-Bernoulli distribution over $(\boldsymbol{x}, y) \in \rho\mathbb{H}^d \times \{\pm 1\}$ as follows:

- first draw the label y uniformly at random from $\{\pm 1\}$,
- then sample the data point $\boldsymbol{x}$ by taking $y\rho\boldsymbol{\phi}$ and flipping the sign of each coordinate independently with probability $1/2 - \tau$.

These binary classification data models on hypercube vertices are inspired by the MNIST dataset [LeCun et al., 1998]. The class bias parameter τ controls the difficulty of the classification task, which increases as τ tends to zero, i.e. as $1/\tau$ tends to infinity.

In this section and the next, we consider adversarial tasks that are $(\boldsymbol{\phi}, \rho, \tau)$-Bernoulli data models, and investigate variations of the parameters $\boldsymbol{\phi}$, ρ and τ.

Adversarial program. In this section, we assume that $k \leq d$, i.e. the network width is no greater than the input dimension, and we define an adversarial program $\boldsymbol{p}$ which depends on the weights of the network $\mathcal{N}$ and on the direction $\boldsymbol{\phi}$ of the Bernoulli data model.

With probability 1, for all $j \in [k]$, we have that $a_j \boldsymbol{w}_j^\top \boldsymbol{\phi} \neq 0$. Let us write K^+ for $\{j \in [k] \mid a_j \boldsymbol{w}_j^\top \boldsymbol{\phi} > 0\}$, and K^- for $\{j \in [k] \mid a_j \boldsymbol{w}_j^\top \boldsymbol{\phi} < 0\}$. Then, for all $j \in [k]$, let:

$$\boldsymbol{p}'_j = \begin{cases} 0 & \text{if } j \in K^+, \\ -\sqrt{d/|K^-|} & \text{if } j \in K^-. \end{cases}$$

Since $k \leq d$, with probability 1, the weights vectors $\boldsymbol{w}_j$ are linearly independent, i.e. the $k \times d$ matrix $\boldsymbol{W}$ whose rows are $\boldsymbol{w}_j$ has a positive smallest singular value $s_{\min}(\boldsymbol{W})$. Hence $\boldsymbol{p} \in \mathbb{R}^d$ exists such that

$$\boldsymbol{p}' = \boldsymbol{W}\boldsymbol{p} \quad \text{and} \quad \frac{\|\boldsymbol{p}'\|}{s_{\max}(\boldsymbol{W})} \leq \|\boldsymbol{p}\| \leq \frac{\|\boldsymbol{p}'\|}{s_{\min}(\boldsymbol{W})} \ . \tag{1}$$

The neurons in K^+ can be thought of as "helpful" for the adversarial task, and those in K^- as "unhelpful". This definition of an adversarial program $\boldsymbol{p}$ ensures that its effect is to introduce as first-layer biases the entries of the vector $\boldsymbol{p}'$. They are 0 (i.e. do nothing) for every "helpful" neuron, and the negative value $-\sqrt{d/|K^-|}$ (i.e. reduce the contribution to the network output) for every "unhelpful" neuron.

From the inequalities in (1), we can expect $\|\boldsymbol{p}\| \approx \sqrt{d}$ if $k = o(d)$ (see the appendix for details).

Expected reprogramming accuracy. The accuracy for an adversarial task $\mathcal{D}$ is the probability that the sign of the output of the reprogrammed network conforms to the input label, i.e.

$$\mathbb{P}_{(\boldsymbol{x},y)\sim\mathcal{D}}\{y\,\mathcal{N}(\boldsymbol{p}+\boldsymbol{x}) > 0\} \ .$$

Our main result in this section is that, for sufficiently large input dimensions d, and under mild restrictions on the growth rates of the network width k, the radius ρ and the difficulty $1/\tau$ of the Bernoulli data model, in expectation over the random network weights, the reprogramming accuracy is at least $(1 - C_1\gamma)(1 - \gamma^\dagger)$, which by tuning the probability parameters γ and $\gamma^\dagger$ can be arbitrarily close to 100%. The growth rate restrictions indicate that networks with smaller widths can be reprogrammed for tasks with larger radii, but that networks with larger widths can be reprogrammed for tasks that are more difficult. The proof proceeds by analysing concentrations of the underlying distributions and involves establishing a theorem that bounds the reprogrammed network output; the details can be found in the appendix.

Corollary 1. *Suppose that*

$$k = \Theta(d^{\eta_{(k)}}) \ , \quad \rho = O(d^{\eta_{(\rho)}}) \ , \quad 1/\tau = O(d^{\eta_{(\tau)}}) \quad \textit{and} \quad 1/\tau = \omega_d(1) \ ,$$

where $\eta_{(k)}, \eta_{(\rho)}, \eta_{(\tau)} \in [0,1]$ are arbitrary constants that satisfy

$$\eta_{(\rho)} < 1 - \eta_{(k)}/2 \quad \textit{and} \quad \eta_{(\tau)} < \eta_{(k)}/2 \ .$$

Then, for sufficiently large input dimensions d, the expected accuracy of the adversarially reprogrammed network $\mathcal{N}$ on the $(\boldsymbol{\phi}, \rho, \tau)$-Bernoulli data model is arbitrarily close to 100%.

3 Implicit bias

Orthogonally separable dataset. In this section, we consider training the network on a binary classification dataset $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \{\pm 1\}$ which is orthogonally separable [Phuong and Lampert, 2021], i.e. for all $i, i' \in [n]$ we have:

$$\boldsymbol{x}_i^\top \boldsymbol{x}_{i'} > 0 \quad \text{if} \quad y_i = y_{i'} \ , \quad \text{and} \quad \boldsymbol{x}_i^\top \boldsymbol{x}_{i'} \leq 0 \quad \text{if} \quad y_i \neq y_{i'} \ .$$

In other words, every data point can act as a linear separator, although some data points from the opposite class may be exactly orthogonal to it.

Gradient flow with exponential or logistic loss. For two-layer ReLU networks with input dimension d and width k as before (but without the assumption $k \leq d$, which is not needed in this section), we denote the vector of all weights by

$$\boldsymbol{\theta} \coloneqq (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, a_1, \ldots, a_k) \in \mathbb{R}^{k(d+1)} \ ,$$

and we write $\mathcal{N}_{\boldsymbol{\theta}}$ for the network whose weights are the coordinates of the vector $\boldsymbol{\theta}$.

The empirical loss of $\mathcal{N}_{\boldsymbol{\theta}}$ on S is $\mathcal{L}(\boldsymbol{\theta}) \coloneqq \sum_{i=1}^n \ell(y_i \mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x}_i))$, where ℓ is either the exponential $\ell_{\exp}(u) = e^{-u}$ or the logistic $\ell_{\log}(u) = \ln(1 + e^{-u})$ loss function.

A trajectory of gradient flow is a function $\boldsymbol{\theta}(t) : [0, \infty) \to \mathbb{R}^{k(d+1)}$ that is an arc, i.e. it is absolutely continuous on every compact subinterval, and that satisfies the differential inclusion

$$\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} \in -\partial\mathcal{L}(\boldsymbol{\theta}(t)) \quad \text{for almost all} \quad t \in [0, \infty) \ ,$$

where $\partial\mathcal{L}$ denotes the Clarke subdifferental [Clarke, 1975] of the locally Lipschitz function $\mathcal{L}$.

Gradient flow is gradient descent with infinitesimal step size. We work with the Clarke subdifferential in order to handle the non-differentiability of the ReLU function at zero: $\partial\psi(0)$ is the whole interval $[0, 1]$. At points of continuous differentiability, the Clarke subdifferential amounts to the gradient, e.g. $\partial\mathcal{L}(\boldsymbol{\theta}) = \{\nabla\mathcal{L}(\boldsymbol{\theta})\}$. For some further background, see the appendix.

Initialisation of network weights. In this section, we assume that the initialisation is

balanced: for all $j \in [k]$, at time $t = 0$ we have $|a_j| = \|\boldsymbol{w}_j\| > 0$; and

live: for both signs $s \in \{\pm 1\}$ there exist $i_s \in [n]$ and $j_s \in [k]$ such that $y_{i_s} = s$ and at time $t = 0$ we have $y_{i_s} a_{j_s} \psi(\boldsymbol{w}_{j_s}^\top \boldsymbol{x}_{i_s}) > 0$.

The balanced assumption has featured in previous work (see e.g. Phuong and Lampert [2021], Lyu, Li, Wang, and Arora [2021]). It ensures that it remains to hold throughout the training, and that the signs of the second-layer weights a_j do not change (see the proof of Theorem 2). The live assumption (present probabilistically in Phuong and Lampert [2021]) is mild: it states that at least one positively initialised neuron is active for at least one positive input, and the same for negative ones.

Convergence of gradient flow. Our main result in this section establishes that the early phase of training necessarily reaches a point where the empirical loss is less than $\ell(0)$, which implies that then every input is classified correctly by the network. Perhaps surprisingly, no small initialisation assumption is needed, however the proof makes extensive use of orthogonal separability of the dataset (see the appendix, which contains all proofs for this section).

Theorem 2. *There exists a time t_0 such that $\mathcal{L}(\boldsymbol{\theta}(t_0)) < \ell(0)$.*

This enables us to apply to the late phase recent results of Lyu and Li [2020], Ji and Telgarsky [2020], Lyu et al. [2021] and obtain the next corollary, which is significantly stronger than the main result of Phuong and Lampert [2021], extending it to exponential loss, and showing that assumptions of small initialisation, and of positive and negative support examples spanning the whole space, are not needed. The corollary establishes that each neuron converges to one of three types: a scaling of the maximum-margin vector for the positive data class, a scaling of the maximum-margin vector for the negative data class, or zero. The two maximum-margin vectors are defined as follows: for both signs $s \in \{\pm 1\}$, let $I_s \coloneqq \{i \in [n] \mid y_i = s\}$, and let $\boldsymbol{v}_s$ be the unique minimiser of the quadratic problem

$$\text{minimise} \quad \frac{1}{2}\|\boldsymbol{v}\|^2 \quad \text{subject to} \quad \forall i \in I_s : \boldsymbol{v}^\top \boldsymbol{x}_i \geq 1 \ .$$

That a trajectory $\boldsymbol{\theta}(t)$ converges in direction to a vector $\widetilde{\boldsymbol{\theta}}$ means $\lim_{t\to\infty} \boldsymbol{\theta}(t)/\|\boldsymbol{\theta}(t)\| = \widetilde{\boldsymbol{\theta}}/\|\widetilde{\boldsymbol{\theta}}\|$.

Corollary 3. *As the time tends to infinity, we have that the empirical loss converges to zero, the Euclidean norm of the weights converges to infinity, and the weights converge in direction to some $\boldsymbol{\theta}$ such that for all $j \in [k]$ we have $|a_j| = \|\boldsymbol{w}_j\|$, and if $a_j \neq 0$ then*

$$\frac{\boldsymbol{w}_j}{\|\boldsymbol{w}_j\|} = \frac{\boldsymbol{v}_{\operatorname{sgn}(a_j)}}{\sum_{\operatorname{sgn}(a_{j'})=\operatorname{sgn}(a_j)} a_{j'}^2} \ .$$

Thanks to homogeneity, the sign of the network output does not depend on the norm of the weights, only on their direction. Examining networks whose weights are directional limits as in Corollary 3 is therefore informative of consequences for adversarial reprogramming of training for long. The following result tells us that, for any Bernoulli data model whose direction is in a half-space of the difference of the maximum-margin vectors, and for any adversarial program, the accuracy tends to $1/2$ provided that the difficulty $1/\tau$ of the data model increases slower than the square root of the input dimension d. The latter assumption is considerably weaker than in the results of Schmidt et al. [2018] on the Bernoulli data model, where the bound is in terms of the fourth root. The statement also tells us that this failure cannot be fixed by choosing in advance a different mapping from the class labels of the original task to the class labels of the adversarial task. Since we consider binary classification tasks here, the mapping can be represented by a multiplier $m \in \{\pm 1\}$.

Proposition 4. *Suppose network weights $\boldsymbol{\theta}$ are is in Corollary 3, class label mapping $m \in \{\pm 1\}$ is arbitrary, data model $\mathcal{D}$ is any $(\boldsymbol{\phi}, \rho, \tau)$-Bernoulli distribution such that $m \cos \angle(\boldsymbol{v}_1 - \boldsymbol{v}_{-1}, \boldsymbol{\phi}) < 0$, and adversarial program $\boldsymbol{p}$ is arbitrary. Then we have that*

$$\mathbb{P}_{(\boldsymbol{x},y)\sim\mathcal{D}}\{m\, y\, \mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{p} + \boldsymbol{x}) > 0\} \ \leq \ \frac{1}{2} + \frac{1}{2}\, e^{-2d\tau^2 \cos^2 \angle(\boldsymbol{v}_1 - \boldsymbol{v}_{-1}, \boldsymbol{\phi})} \ .$$

4 Experiments

Network architectures and initialisation. Our experiments are conducted using the following six network architectures: ResNet-50 [He, Zhang, Ren, and Sun, 2016a], ResNet-50V2, ResNet-101V2,

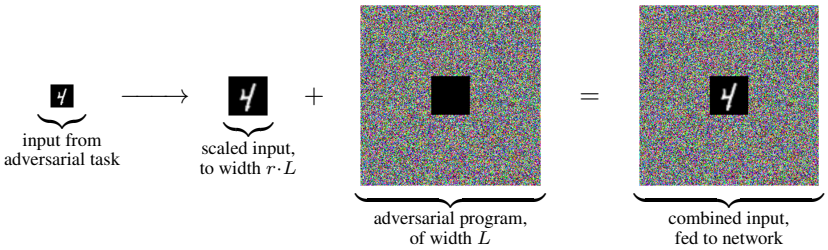

Figure 1: Scheme 1 for combining input images with adversarial programs. In this example, the width and height of the adversarial program are 224 and the parameter r equals $2^{-20/9} \approx 0.214$, so the input image is scaled to width and height $r \cdot 224$ rounded, which is 48.

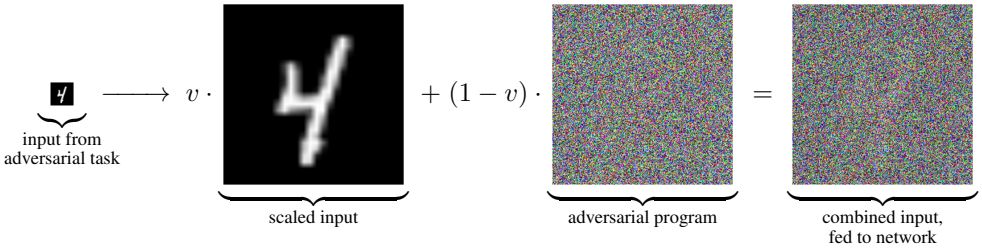

Figure 2: Scheme 2 for combining input images with adversarial programs. In this example, the sizes of the scaled input image and the adversarial program are 224×224. The parameter v equals $2^{-40/9} \approx 0.046$, so the weight of the input image in the convex combination with the adversarial program is approximately 4.6%, which makes it faintly visible.

ResNet-152V2 [He, Zhang, Ren, and Sun, 2016b], Inception-v3 [Szegedy, Vanhoucke, Ioffe, Shlens, and Wojna, 2016], and EfficientNet-B0 [Tan and Le, 2019].

We use the networks exactly as implemented in Keras in TensorFlow 2.8.0 including the method for randomly initialising the trainable weights. For biases, this means they are initialised with 0. For all other trainable weights, mostly, the Glorot uniform initialiser [Glorot and Bengio, 2010] is used in this implementation. EfficientNet is an exception, where many layers instead use a truncated normal distribution that has mean 0 and standard deviation $\sqrt{2/\text{number of output units}}$.

Details of how we randomly initialise batch normalisation layers can be found in the appendix.

Combining input images with adversarial programs. Our adversarial programs are colour images whose sizes match the expected input size of the networks. This is 224×224 for all networks except Inception-v3, where it is 299×299.

We use two different schemes to combine input images with the adversarial program. The first scheme, used by Elsayed et al. [2019], is to take the adversarial program and overwrite a portion of it by the input image. We do this in such a way that the input image is, up to rounding, centred in the adversarial program. We can vary the construction by scaling the input image up or down before applying this procedure. In particular, we use a parameter $r \in [0, 1]$ and scale the input image, using bilinear interpolation, in such a way that r times the width of the adversarial program is equal to the width of the scaled image. That we focus on the width is not important because all our inputs and programs are square. An illustration is shown in Figure 1.

Our second scheme involves scaling the image to the same size as the adversarial program and then taking a convex combination of the two. We use a parameter $v \in [0, 1]$ to specify how much weight the input image should get in this convex combination. Specifically, the combined image is obtained by calculating $v \cdot I + (1 - v) \cdot P$, where I is the input image and P is the adversarial program. An illustration is shown in Figure 2.

Adversarial task dataset. Elsayed et al. [2019] evaluate adversarial reprogramming on random networks using the MNIST [LeCun et al., 1998] dataset. In other words, they were asking whether it

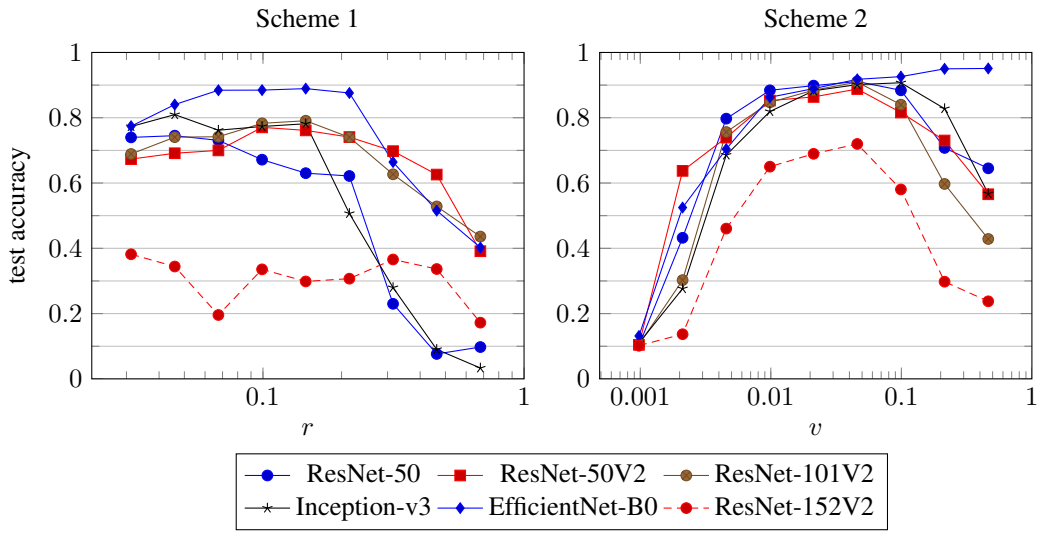

Figure 3: The accuracy achieved by the adversarial program on the MNIST test set for different parameters of the two schemes of combining input images with adversarial programs. The horizontal axes are logarithmic. The values plotted are averages over 5 trials, which are listed together with the standard deviations in the appendix.

is possible to repurpose a random network for the task of classifying the handwritten digits from the MNIST dataset. We use the same dataset, which consists of 60,000 training images and 10,000 test images, for our experiments. It is available under the Creative Commons Attribution Share-Alike 3.0 licence.

The networks we use classify inputs into 1,000 classes. We map the 10 labels of the MNIST dataset onto the first 10 of these classes.

For additional experimental results on the Fashion-MNIST and Kuzushiji-MNIST datasets, please see the appendix.

Finding and evaluating adversarial programs. Internally, we represent adversarial programs using unconstrained weights. We then apply a softsign function to the weights to map them into the range $(-1, 1)$, and further shift and scale the program such that the pixel values lie in the same range that is used for the input images. The program is initialised in such a way that after the application of the softsign function, each value lies uniformly at random in $(-1, 1)$.

We use the 60,000 training images to run an Adam optimiser [Kingma and Ba, 2015] with learning rate 0.01 and a batch size of 50 to optimise the unconstrained weights of the adversarial program. We report the accuracy on the 10,000 test images after 20 epochs, please see Figure 3.

The experiments were mainly run on two internal clusters utilising a mix of NVIDIA GPUs such as GeForce RTX 3080 Ti, Quadro RTX 6000, GeForce RTX 3060, GeForce RTX 2080 Ti, and GeForce GTX 1080. Depending on the network, optimising a single adversarial program for 20 epochs takes between 30 minutes and 1.5 hours on a standard desktop computer with two NVIDIA GeForce RTX 3080 Ti GPUs.

We did not explore different optimisers and learning rates, since our first choices already resulted in suitable adversarial programs for these random networks. We only reduced the batch size to 50 after first trying 100, in order to reduce the requirement on GPU memory and be able to easily run the experiments on a wider range of hardware. However, we did extensively explore the two different schemes of combing input images with adversarial programs, and different values for the respective parameters r and v. For each network, and each value of r and v, we ran 5 experiments, each with a new random initialisation of the network, and are reporting the average of the test accuracy.

Discussion. Overall, the second scheme of combining input images with adversarial programs appears to give better and more reliable results in our experiments. For both schemes, the choice of parameters is important. Clearly, when r or v is 0, the input image is not visible to the network at all. On the other hand, when r or v is 1, there no longer is an adversarial program. In most cases, best results are achieved when the adversarial program is significantly larger (either by actual size in the first scheme, or in terms of pixel value ranges in the second scheme) than the input image.

In the second scheme in particular, we see accuracies on the test set which are lower, but not much lower than what Elsayed et al. [2019] reported for networks trained on ImageNet. For $v \approx 0.046$ for example, we see accuracies of 91.8%, 91.1%, 90.9%, 90.2%, 88.8%, 72.0% for EfficientNet-B0, ResNet-50, ResNet-101V2, Inception-v3, ResNet-50V2, ResNet-152V2, respectively. This suggests that, while training, say, on ImageNet may impact the possibility of finding suitable adversarial programs, such a training may be less important than previously thought and other factors are of significant importance.

5 Conclusion and future work

We proved the first theoretical results on adversarial reprogrammability of neural networks, in which we focused on architectures with two layers and ReLU activations, and on Bernoulli and Gaussian adversarial tasks. Provided the input dimension is sufficiently large, and for a wide variety of parameter regimes, our results show that: firstly, arbitrarily high reprogramming accuracies are achievable in expectation for networks with random weights; and secondly, reprogramming accuracies that are no better than guessing may be unavoidable for networks that were trained for long with small learning rates on orthogonally separable datasets.

In the theoretical results that conclude arbitrarily high expected reprogramming accuracies, we assumed that the width of the random network is no grater than its input dimension, which is similar to the assumption on widths in e.g. Daniely and Shacham [2020] and enabled us to show existence of suitable adversarial programs by matrix inversion. Interesting directions for future work include relaxing this assumption, extending the whole theoretical analysis to deeper networks, and investigating derivation of adversarial programs by gradient methods.

It would also be interesting to consider more permissive data models; however, our theoretical results on the failure of adversarial reprogramming on networks that were trained for long rely on implicit bias properties of gradient methods, and in that area separability assumptions on training data are common and appear challenging to lift (see e.g. Lyu et al. [2021]).

The outcomes of our experiments, which are on six realistic convolutional network architectures designed for image classification, and on three adversarial tasks provided by the MNIST, Fashion-MNIST and Kuzushiji-MNIST datasets, are consistent with our theoretical results on high reprogramming accuracies. Both the experimental plots of test accuracies and the theoretical bounds of network outputs indicate existence of "sweet spot" maximising parameter values.[1] Since in the experiments, the network architectures have widths, depths and other features that are currently beyond our theoretical assumptions, and the adversarial programs are derived by gradient descent, their outcomes provide further motivation for extending the theory as suggested above.

A conclusion that emerges across our experimental results is that the EfficientNet-B0 architecture tends to be more susceptible to adversarial reprogramming, and the ResNet-152V2 architecture tends to be less susceptible, than the remaining four. We suggest for future work investigating the causes of this, as well as seeking to reprogram random networks for adversarial tasks that are more difficult than MNIST, Fashion-MNIST and Kuzushiji-MNIST. In the context of the latter, it would be interesting to consider also vision transformer [Dosovitskiy, Beyer, Kolesnikov, Weissenborn, Zhai, Unterthiner, Dehghani, Minderer, Heigold, Gelly, Uszkoreit, and Houlsby, 2021] architectures.

Another direction for experimental work is to verify that training for long can cause adversarial reprogramming to fail.

[1] The varying of the scaling parameters r and v in the experiments corresponds in the Bernoulli data model case to varying the radius ρ while keeping all other parameters fixed, and in the Gaussian data model case to varying the mean radius ϱ and the variance parameter ς while keeping fixed their ratio and all other parameters.

References

Peter L. Bartlett, Sébastien Bubeck, and Yeshwanth Cherapanamjeri. Adversarial Examples in Multi-Layer Random ReLU Networks. In *NeurIPS*, pages 9241–9252, 2021.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion Attacks against Machine Learning at Test Time. In *ECML PKDD*, pages 387–402, 2013. 1

Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial Patch. *CoRR*, abs/1712.09665, 2017.

Sébastien Bubeck, Yeshwanth Cherapanamjeri, Gauthier Gidel, and Rémi Tachet des Combes. A single gradient step finds adversarial examples on random two-layers neural networks. In *NeurIPS*, pages 10081–10091, 2021. 4

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep Learning for Classical Japanese Literature. *CoRR*, abs/1812.01718, 2018. 3

Frank H. Clarke. Generalized gradients and applications. *Transactions of the American Mathematical Society*, 205:247–262, 1975. 5

Amit Daniely and Hadas Shacham. Most ReLU Networks Suffer from ℓ^2 Adversarial Perturbations. In *NeurIPS*, 2020. 9

Damek Davis, Dmitriy Drusvyatskiy, Sham M. Kakade, and Jason D. Lee. Stochastic Subgradient Method Converges on Tame Functions. *Found. Comput. Math.*, 20(1):119–154, 2020.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021. 9

Joydeep Dutta, Kalyanmoy Deb, Rupesh Tulshyan, and Ramnik Arora. Approximate KKT points and a proximity measure for termination. *J. Glob. Optim.*, 56(4):1463–1499, 2013.

Gamaleldin F. Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial Reprogramming of Neural Networks. In *ICLR*, 2019. 1, 2, 3, 7, 9

Justin Gilmer, Luke Metz, Fartash Faghri, Samuel S. Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian J. Goodfellow. Adversarial Spheres. In *ICLR (Workshop)*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010. 7

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. WARP: Word-level Adversarial ReProgramming. In *ACL/IJCNLP*, pages 4921–4933, 2021. 2

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV*, pages 1026–1034, 2015. 4

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016a. 6

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. In *ECCV*, pages 630–645, 2016b. 7

Kun He, Yan Wang, and John E. Hopcroft. A Powerful Generative Model Using Random Weights for the Deep Image Representation. In *NeurIPS*, pages 631–639, 2016c. 2

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Mądry. Adversarial Examples Are Not Bugs, They Are Features. In *NeurIPS*, pages 125–136, 2019.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, pages 448–456, 2015.

Ziwei Ji and Matus Telgarsky. Directional convergence and alignment in deep learning. In *NeurIPS*, 2020. 6

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 8

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, University of Toronto, 2009. 1

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. 1, 4, 7

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. In *ICLR*, 2018. 2

Kaifeng Lyu and Jian Li. Gradient Descent Maximizes the Margin of Homogeneous Neural Networks. In *ICLR*, 2020. 6

Kaifeng Lyu, Zhiyuan Li, Runzhe Wang, and Sanjeev Arora. Gradient Descent on Two-layer Nets: Margin Maximization and Simplicity Bias. In *NeurIPS*, pages 12978–12991, 2021. 6, 9

Grégoire Mesnil, Yann N. Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian J. Goodfellow, Erick Lavoie, Xavier Muller, Guillaume Desjardins, David Warde-Farley, Pascal Vincent, Aaron C. Courville, and James Bergstra. Unsupervised and Transfer Learning Challenge: a Deep Learning Approach. In *ICML Workshop on Unsupervised and Transfer Learning*, pages 97–110, 2011. 2

Andrea Montanari and Yuchen Wu. Adversarial Examples in Random Neural Networks with General Activations. *CoRR*, abs/2203.17209, 2022.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal Adversarial Perturbations. In *CVPR*, pages 86–94, 2017. 1

Paarth Neekhara, Shehzeen Hussain, Shlomo Dubnov, and Farinaz Koushanfar. Adversarial Reprogramming of Text Classification Neural Networks. In *EMNLP-IJCNLP*, pages 5215–5224, 2019. 2

Paarth Neekhara, Shehzeen Hussain, Jinglong Du, Shlomo Dubnov, Farinaz Koushanfar, and Julian J. McAuley. Cross-modal Adversarial Reprogramming. In *WACV*, pages 2898–2906, 2022. 2

Mary Phuong and Christoph H. Lampert. The inductive bias of ReLU networks on orthogonally separable data. In *ICLR*, 2021. 3, 5, 6

Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *ICML*, pages 759–766, 2007. 2

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015. 1

Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Mądry. Adversarially Robust Generalization Requires More Data. In *NeurIPS*, pages 5019–5031, 2018. 3, 4, 6

Adi Shamir, Odelia Melamed, and Oriel BenShmuel. The Dimpled Manifold Model of Adversarial Examples in Machine Learning. *CoRR*, abs/2106.10151, 2021.

Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In *CCS*, pages 1528–1540, 2016.

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The Implicit Bias of Gradient Descent on Separable Data. *J. Mach. Learn. Res.*, 19:70:1–70:57, 2018.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014. 1

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, pages 2818–2826, 2016. 7

Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*, pages 6105–6114, 2019. 7

Yun-Yun Tsai, Pin-Yu Chen, and Tsung-Yi Ho. Transfer Learning without Knowing: Reprogramming Black-box Machine Learning Models with Scarce Data and Limited Resources. In *ICML*, pages 9614–9624, 2020. 2

Gal Vardi, Gilad Yehudai, and Ohad Shamir. Gradient Methods Provably Converge to Non-Robust Networks. *CoRR*, abs/2202.04347, 2022.

Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. In Yonina C. Eldar and Gitta Kutyniok, editors, *Compressed Sensing*, pages 210–268. Cambridge University Press, 2012. URL http://arxiv.org/abs/1011.3027.

Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 2018.

Ria Vinod, Pin-Yu Chen, and Payel Das. Reprogramming Language Models for Molecular Representation Learning. *CoRR*, abs/2012.03460, 2020. 2

Martin J. Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 2019.

Yifei Wang and Mert Pilanci. The Convex Geometry of Backpropagation: Neural Network Gradient Flows Converge to Extreme Points of the Dual Convex Program. In *ICLR*, 2022. 3

Yunjuan Wang, Enayat Ullah, Poorya Mianjy, and Raman Arora. Adversarial Robustness is at Odds with Lazy Training. *CoRR*, abs/2207.00411, 2022.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*, abs/1708.07747, 2017. 3

Chao-Han Huck Yang, Yun-Yun Tsai, and Pin-Yu Chen. Voice2Series: Reprogramming Acoustic Models for Time Series Classification. In *ICML*, pages 11808–11819, 2021. 2

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Honggang Zhang, Jun Guo, Guang Chen, and Chun-Guang Li. HCL2000 - A Large-scale Handwritten Chinese Character Database for Handwritten Character Recognition. In *ICDAR*, pages 286–290, 2009. 2

Yang Zheng, Xiaoyi Feng, Zhaoqiang Xia, Xiaoyue Jiang, Ambra Demontis, Maura Pintor, Battista Biggio, and Fabio Roli. Why Adversarial Reprogramming Works, When It Fails, and How to Tell the Difference. *CoRR*, abs/2108.11673, 2021. 2

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] They are determined by the assumptions of the theoretical results and by the restrictions of the experiments ran, and we stated both.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] They stem from potential nefarious uses of adversarial reprogramming, which we indicated on page 2 with a reference to a fuller discussion in the literature.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] In the paper together with its appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 and the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See the appendix.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]