Information Sciences 632 (2023) 130–143

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Why adversarial reprogramming works, when it fails, and how to tell the difference

Yang Zheng [a], Xiaoyi Feng [a], Zhaoqiang Xia [a], Xiaoyue Jiang [a], Ambra Demontis [b,*], Maura Pintor [b], Battista Biggio [b], Fabio Roli [a,c]

[a] Northwestern Polytechnical University, Xi'an, China
[b] University of Cagliari, Italy
[c] University of Genoa, Italy

ARTICLE INFO

Keywords:
Adversarial machine learning
Adversarial reprogramming
Neural networks
Transfer learning

ABSTRACT

Adversarial reprogramming allows repurposing a machine-learning model to perform a different task. For example, a model trained to recognize animals can be reprogrammed to recognize digits by embedding an adversarial program in the digit images provided as input. Recent work has shown that adversarial reprogramming may not only be used to abuse machine-learning models provided as a service, but also beneficially, to improve transfer learning when training data is scarce. However, the factors affecting its success are still largely unexplained. In this work, we develop a first-order linear model of adversarial reprogramming to show that its success inherently depends on the size of the average input gradient, which grows when input gradients are more aligned, and when inputs have higher dimensionality. The results of our experimental analysis, involving fourteen distinct reprogramming tasks, show that the above factors are correlated with the success and the failure of adversarial reprogramming.

1. Introduction

Adversarial reprogramming is a technique that repurposes a machine learning model, originally trained for a task, to perform a different chosen task, without retraining or fine-tuning it. This technique optimizes an adversarial perturbation (*adversarial program*) that can be applied to the model's inputs to make the model perform the chosen task. For example, a model trained to recognize certain classes of samples from a *source* domain (e.g., ImageNet objects) can be reprogrammed to classify samples belonging to a different, *target* domain (e.g., MNIST handwritten digits). To this end, one should first establish a mapping function between the class labels of the source domain and those of the target domain (e.g., the handwritten digit "0" can be associated to the ImageNet class "tench", the handwritten digit "1" to the ImageNet class "goldfish", etc.). Once such a class mapping is established, all the target-domain samples are modified to embed the adversarial program, i.e., an adversarial perturbation equal for all the samples (*universal*) optimized against the target model to have the samples of the target domain assigned to the desired source-domain classes. An example of adversarial program for reprogramming an ImageNet model to classify handwritten digits is shown in Fig. 1 (*top*). In this case, the adversarial program consists of a frame surrounding the input image, but such programs, as well as other adversarial perturbations, can also be optimized to be superimposed on the input samples. While reprogramming was initially proposed for

* Corresponding author.
E-mail address: ambra.demontis@unica.it (A. Demontis).

https://doi.org/10.1016/j.ins.2023.02.086
Received 15 August 2022; Received in revised form 29 January 2023; Accepted 26 February 2023

Available online 1 March 2023

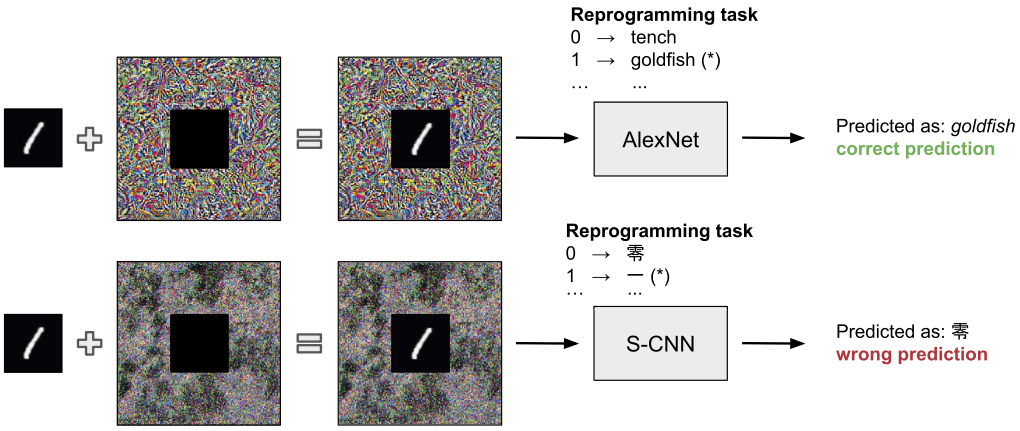

Fig. 1. Adversarial reprogramming of AlexNet [1], trained on the ImageNet dataset (*top*), and CWNet [2], trained on Chinese digits (*bottom*), to recognize MNIST handwritten digits. Each digit class is mapped to a different class among those predicted by the target model (e.g., for AlexNet, 0 to *tench*, 1 to *goldish*, etc.). In the example, the handwritten digit 1 is embedded in both adversarial programs expected to be classified as *goldfish* by AlexNet and as the Chinese digit 1 by CWNet. However, while AlexNet can be successfully reprogrammed, reprogramming fails for CWNet.

images, its application on other domains, such as text [38] and audio [37] classification, is also possible. Additionally, reprogramming was also shown to be useful for repurposing models across different domains [36,16]. For example the authors of [16] have shown that a model trained to perform classification on the ImageNet dataset can be reprogrammed to perform sentiment analysis and topic classification.

Adversarial reprogramming was originally introduced to abuse machine-learning models provided as a service and steal computational resources by repurposing a model to perform a task chosen by the attacker. For instance, an online service that uses deep networks originally trained to classify images of objects can be reprogrammed to recognize numbers and digits for solving CAPTCHAs[1] and automating the creation of fake accounts [3]. However, it has been recently shown that adversarial reprogramming can also be used for beneficial tasks like transfer learning, in scenarios with scarce data and constrained resources. The authors of [4] have empirically demonstrated that in such scenarios, adversarial reprogramming achieves better performance than fine-tuning, i.e., the classical approach to transfer learning in deep networks. We conjecture this is because, in scenarios with scarce data, the number of parameters that should be optimized becomes of paramount importance. If the parameters are too many, having a big dataset is necessary to learn the task, whereas a small set of parameters can be optimized even with small datasets. Fine-tuning requires modifying the model parameters, which are often a considerable number for deep neural networks. Reprogramming, instead, requires optimizing the parameters of the adversarial program, which are usually a number quite smaller. Another advantage of reprogramming over fine-tuning is that, unlike fine-tuning, reprogramming allows leaving the model as it is, thus preserving the original functionality for which it was trained. Furthermore, it can also be performed having only black-box access to the machine-learning model (without knowing its internal details, including its architecture and learned parameters) [4]. In this case, one can compute the reprogramming mask by repetitively querying the model and observing its predictions for different inputs.

Notwithstanding the practical relevance of adversarial reprogramming, the factors affecting its success are still unclear. Even the most recently published papers have not answered the following key questions: when and why adversarial reprogramming works, but most importantly, when and why it fails. In the original work that proposed adversarial reprogramming [3], the authors successfully applied this technique to solve a complex task, i.e., reprogramming an ImageNet model to classify MNIST handwritten digits. However, they also showed that reprogramming did not successfully work when trying to reprogram untrained networks to solve the same task. While starting investigating the reasons behind such failures, we tried to apply reprogramming to what we thought was a simpler task: reprogramming a small convolutional neural network (CWNet [2], described in Table 1), trained to recognize handwritten Chinese digits, so that it could recognize the MNIST handwritten (Arabic) digits, as depicted in Fig. 1 (*bottom*). Despite our efforts in tuning the hyperparameters, we found that adversarial reprogramming surprisingly fails on this intuitively simpler task. Our conjecture was based on the fact that both the source and the target domain in this case are digit recognition problems. However, as we will show later, task similarity does not help adversarial reprogramming.

To shed light on the underlying factors affecting success and failure of adversarial reprogramming, in this work we provide a first-order linear analysis of adversarial reprogramming, starting from the observation that adversarial programs can be indeed considered as universal adversarial perturbations (Sect. 2). Our analysis shows that the success of reprogramming is inherently dependent on the size of the *average* input gradient, which grows (i) when the input gradients (i.e., the gradients of the classification loss w.r.t. the input values, computed on different target-domain samples) are more aligned, and (ii) when the number of input dimensions increases. To validate the proposed mathematical model, we carry out an extensive empirical analysis (Sect. 3) involving three different neural-network models and four datasets, resulting in fourteen distinct reprogramming tasks. Our experimental analysis shows that our first-order model of adversarial reprogramming correctly highlights the main factors behind its success

[1] Completely Automated Public Turing test to tell Computers and Humans Apart.

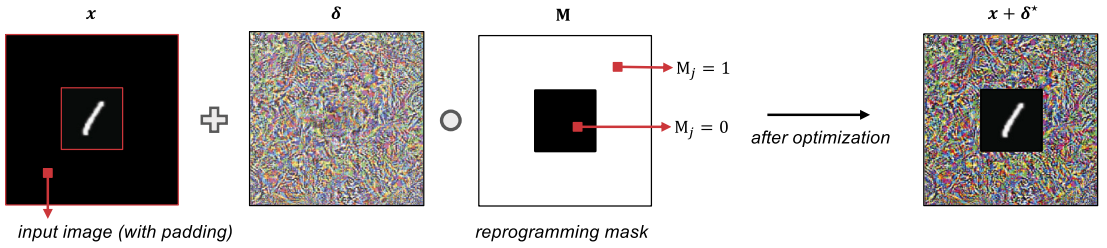

Fig. 2. Reprogramming mask **M** used to restrict the adversarial perturbation to the frame surrounding the target-domain input image, initially padded with zeros.

and failure. We conclude the paper by discussing related work (Sect. 4), along with the main contributions, limitations and future developments of our work (Sect. 5).

2. Understanding adversarial reprogramming

In this section we develop a first-order mathematical model of adversarial reprogramming, which aims to highlight the main factors underlying its success and failure. To this end, we first formalize adversarial reprogramming as a loss minimization problem (Sect. 2.1). We then show that, under linearization of the loss function, the optimal solution can be computed in closed form, and the loss reduction is proportional to the size of the average input gradient (Sect. 2.2). This in turn means that reprogramming a target model is easier when the input gradients are well aligned, and inputs are high dimensional. We conclude the section by discussing how our analysis can be extended to adversarial programs more general than those depicted in Fig. 1 (Sect. 2.3).

2.1. Problem formulation

Given a model f, trained on a source-domain dataset (e.g. ImageNet) $\mathcal{S}$, the goal of adversarial reprogramming is to find a single universal perturbation δ, that can be applied to all the samples of the target-domain data (e.g. the MNIST handwritten digits set), to make the model perform the desired task (e.g. classify Arabic numbers despite being trained to classify objects).

Let us assume that we have a source-domain dataset $\mathcal{S} = (\tilde{\boldsymbol{x}}_j, \tilde{y}_j)_{j=1}^m$ and a target-domain dataset $\mathcal{T} = (\boldsymbol{x}_i, y_i)_{i=1}^n$, consisting of m and n samples, respectively, along with their labels. In both cases, the input samples are represented as d-dimensional vectors in $\mathcal{X} = [-1, 1]^d$, while the class labels belong to different sets, respectively, $\tilde{y} \in \tilde{\mathcal{Y}}$ and $y \in \mathcal{Y}$ for the source and the target domain. We are also given a target model $f : \mathcal{X} \mapsto \tilde{\mathcal{Y}}$ which makes predictions on the source domain, parameterized by $\boldsymbol{\theta} \in \mathbb{R}^t$. To reprogram this model, we first define a class-mapping function $h : \mathcal{Y} \mapsto \tilde{\mathcal{Y}}$ that maps each label of the target domain to a label of the source domain (e.g., the target-domain label "0" to the source-domain label "tench"). We then need to optimize and embed the adversarial program in the target-domain samples.

Reprogramming mask. Even though our model and analysis can be generalized and extended to many different kinds of adversarial programs, as discussed in Sect. 2.3, we restrict our focus here on adversarial programs consisting of a frame surrounding the target-domain samples, as shown in Fig. 1 and originally considered in the seminal work in [3]. This means that the target-domain samples are assumed to be smaller than the source-domain samples, and padded with zeros to reach the required input size d; for example, MNIST handwritten digits consist of $28 \times 28 = 784$ pixels per channel, and should be padded with more than 49,000 zeros per channel to reach the input size of ImageNet models (which have $224 \times 224 = 50,176$ pixels per channel). We represent reprogramming masks as a binary vector $\mathbf{M} \in \{0, 1\}^d$, whose values are set to 0 in the region occupied by the target-domain sample, and to 1 in the surrounding frame (i.e., the portion of the image which can be modified), as shown in Fig. 2.

Under these assumptions, the optimal adversarial program $\boldsymbol{\delta}^\star$ can be obtained by solving the following optimization problem:

$$\boldsymbol{\delta}^\star \in \arg\min_{\boldsymbol{\delta} \in [-1,1]^d} L(\boldsymbol{\delta}, \mathcal{T}) = \frac{1}{n} \sum_{i=1}^{n} \ell(\boldsymbol{x}_i + \boldsymbol{\delta} \circ \mathbf{M}, h(y_i), \boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{\delta}$ is the adversarial program being optimized within the feasible domain $\mathcal{X} = [-1, 1]^d$, $\mathbf{M} \in \{0, 1\}^d$ is the reprogramming mask, the $\circ$ operator denotes element-wise vector multiplication, and ℓ is the cross-entropy loss, which is minimized when the perturbed target-domain samples are assigned to the desired source-domain classes. Note that the constraint $\boldsymbol{\delta} \in [-1, 1]^d$ is equivalent to upper bounding the ℓ_∞ norm of $\boldsymbol{\delta}$ as $\|\boldsymbol{\delta}\|_\infty \leq 1$.

Solution algorithm. In this work, we use Algorithm 1 to solve the optimization problem in Eq. (1) via stochastic gradient descent. This algorithm extends the Projected Gradient Descent (PGD) algorithm originally used in [5] to optimize adversarial perturbations within an ϵ-sized ℓ_∞-norm constraint. Our algorithm iteratively updates the adversarial program $\boldsymbol{\delta}$ to minimize the expected loss on the target-domain samples (line 3). In each iteration, the target-domain samples are randomly shuffled (line 4) and subdivided in b batches of size B. The adversarial program is then updated by iterating over the batches (line 6). In particular, the average input gradient $\boldsymbol{g}$ is first computed on the batch samples (line 7), and then the adversarial program is updated with an η-sized step (line 8) along the sign of the negative gradient (i.e., the steepest descent direction under the given ℓ_∞-norm constraint). In our case, the gradient for the i^{th} sample is computed as $\boldsymbol{g}_i = \nabla_{\boldsymbol{\delta}} \ell(\boldsymbol{x}_i, h(y_i), \boldsymbol{\theta}) \circ \mathbf{M}$, i.e., including the application of the reprogramming mask

Algorithm 1 Adversarial reprogramming via stochastic gradient descent.

Input: the target-domain dataset $\mathcal{T} = (\boldsymbol{x}_i, y_i)_{i=1}^n$, the model parameters $\boldsymbol{\theta}$, the batch size B, the number of iterations N, the step size η, and the projection operator $\Pi_{\mathcal{X}}$.
Output: the optimal adversarial program $\boldsymbol{\delta}^*$.

1: $\boldsymbol{\delta} \leftarrow \mathbf{0}$, $\boldsymbol{\delta}^* \leftarrow \boldsymbol{\delta}$, $\text{loss}_{\delta^*} \leftarrow \infty$
2: $t \leftarrow 0$
3: **for** $t < N$ **do**
4: Randomly shuffle the samples in $\mathcal{T}$
5: $b \leftarrow 0$
6: **for** $b < \lfloor \frac{n}{B} \rfloor$ **do**
7: $\boldsymbol{g} \leftarrow \frac{1}{B} \sum_{k=B \cdot b}^{B \cdot b + B - 1} \boldsymbol{g}_k$ (average input gradient on the current batch)
8: $\boldsymbol{\delta} \leftarrow \boldsymbol{\delta} - \eta \, \text{sign}(\boldsymbol{g})$
9: $\boldsymbol{\delta} \leftarrow \Pi_{\mathcal{X}}(\boldsymbol{\delta})$
10: $b \leftarrow b + 1$
11: **end for**
12: $\text{loss}_{\delta} = L(\boldsymbol{\delta}, \mathcal{T})$ (compute loss as given in Eq. (1))
13: **if** $\text{loss}_{\delta} < \text{loss}_{\delta^*}$ **then**
14: $\boldsymbol{\delta}^* \leftarrow \boldsymbol{\delta}$
15: $\text{loss}_{\delta^*} \leftarrow \text{loss}_{\delta}$
16: **end if**
17: $t \leftarrow t + 1$
18: **end for**
19: **return** $\boldsymbol{\delta}^\star$

M. After updating $\boldsymbol{\delta}$, the algorithm projects the program onto the feasible space $\mathcal{X} = [-1, 1]^d$, using the box-projection operator $\Pi_{\mathcal{X}}$ (line 9). The algorithm finally returns the adversarial program $\boldsymbol{\delta}^\star$ that achieves the minimum classification loss across the whole optimization process (line 19).

2.2. A first-order model of adversarial reprogramming

We are now ready to introduce the linear model proposed in this work to better understand which underlying factors mostly affect the success of adversarial reprogramming.

Linearization. Reprogramming aims to minimize the loss in Eq. (1) to have the target-domain samples classified as desired within the source-domain classes. Let us linearize the loss in Eq. (1):

$$L(\boldsymbol{\delta}) \approx \frac{1}{n} \sum_{i=1}^{n} \ell(\boldsymbol{x}_i, h(y_i), \boldsymbol{\theta}) + \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\delta}^\top \underbrace{\nabla_{\boldsymbol{x}} \ell(\boldsymbol{x}_i, h(y_i), \boldsymbol{\theta}) \circ \mathbf{M}}_{\boldsymbol{g}_i}, \tag{2}$$

where $\boldsymbol{g}_i$ is the *masked* input gradient, i.e., the input gradient computed for the i^{th} test sample, multiplied by the reprogramming mask $\mathbf{M}$. This amounts to zeroing the values of the input gradient for which the mask is zero. This approximation may not only hold for sufficiently-small input perturbations. It may also hold for larger perturbations if the classification function is linear or has a small curvature (e.g., if it is strongly regularized).

Source and target domain alignment. The first term in Eq. (2), i.e., the loss $\frac{1}{n} \sum_i \ell(\boldsymbol{x}_i, h(y_i), \boldsymbol{\theta})$, measures how difficult is to reprogram a given machine-learning model from the source to the target domain, without applying any input perturbation. If this loss term is small, indeed, it means that the unperturbed samples from the target domain are already consistently assigned to the desired source-domain classes with high probability; thus, even small perturbations may provide high reprogramming accuracy. Conversely, when this one-to-one mapping between source and target classes is only weakly present, this term takes on larger values, meaning that reprogramming may be more challenging. However, our experiments in Sect. 3 show that this term does not play a significant role for reprogramming accuracy, since the source-domain samples and the target-domain samples are normally quite different.

Loss decrement. The second term in Eq. (2) is the loss decrement that can be achieved when optimizing $\boldsymbol{\delta}$. It can be rewritten as:

$$\Delta L(\boldsymbol{\delta}) = \boldsymbol{\delta}^\top \frac{1}{n} \sum_i \boldsymbol{g}_i = \boldsymbol{\delta}^\top \boldsymbol{g}, \tag{3}$$

being $\boldsymbol{g}$ the average input gradient. Minimizing the scalar product $\boldsymbol{\delta}^\top \boldsymbol{g}$ under the constraint $\boldsymbol{\delta} \in [-1, 1]^d$ given in Eq. (1) is equivalent to minimizing an inner product over a unit-norm ℓ_∞ ball. It is not difficult to see that, in this case, the optimal perturbation is given as $\boldsymbol{\delta}^\star = -\text{sign}(\boldsymbol{g})$, i.e., it is found by setting each component $\delta_j^\star$, for $j = 1, \ldots, d$, equal to the negative sign of the corresponding element in $\boldsymbol{g}$. Substituting $\boldsymbol{\delta}^\star$ into Eq. (3), we obtain that:

$$\Delta L(\boldsymbol{\delta}^\star) = -\text{sign}(\boldsymbol{g})^\top \boldsymbol{g} = -\sum_{j=1}^{d} \text{sign}(g_j) g_j = -\sum_{j=1}^{d} |g_j| = -\|\boldsymbol{g}\|_1 . \tag{4}$$

This result highlights that reprogramming is more successful when the ℓ_1-norm of the average input gradient $\boldsymbol{g}$ is larger, which in turn means that reprogramming is expected to work better when:

- the input gradients $\boldsymbol{g}_i$ are more aligned, as this would increase the ℓ_1 norm of the average input gradient $\boldsymbol{g}$;
- the number of input dimensions d is higher, as the ℓ_1 norm of $\boldsymbol{g}$ scales linearly with the number of input dimensions.

2.2.1. Gradient alignment

To measure the alignment of the input gradients $\boldsymbol{g}_i$ w.r.t. their average $\boldsymbol{g}$, obtained considering the linearized loss, we introduce the gradient-alignment metric r:

$$r = \frac{\|\boldsymbol{g}\|_1}{\frac{1}{n}\sum_i \|\boldsymbol{g}_i\|_1} \in [0,1]\,. \tag{5}$$

This metric equals zero only when the average vector is $\boldsymbol{g} = \boldsymbol{0}$, while it equals one when the ℓ_1 norm of $\boldsymbol{g}$ is equal to the average of the ℓ_1 norms of the input gradients. Note that the latter case does not require the input gradients to be all equal to the average $\boldsymbol{g}$, conversely to what one may intuitively think. To ensure that $r = 1$, indeed, it suffices that the input gradients $\boldsymbol{g}_i$ are *orthogonal*. For example, consider a simple case in which $\boldsymbol{g}_1 = (0,0,1)$ and $\boldsymbol{g}_2 = (1,0,0)$, with $\boldsymbol{g} = (0.5,0,0.5)$. It is not difficult to see that the average ℓ_1 norm of $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ equals 1 as well as the ℓ_1 norm of $\boldsymbol{g}$. This means that gradient alignment is maximized even if the input gradients are nearly orthogonal, which is quite likely to happen especially in high-dimensional input spaces (at least under the assumption that they are independent and randomly generated).

2.2.2. Reprogramming mask size

The other main factor affecting the success of adversarial reprogramming, as anticipated before, is the number of input dimensions d. In particular, not only gradient alignment is expected to increase when the input space is high dimensional, but also the ℓ_1 norm of the average input gradient is expected to grow linearly with the number of input dimensions d. Note however that, when using a reprogramming mask, the number of dimensions (i.e., pixels) used to optimize the adversarial program is not equal to d, but rather to the size of the reprogramming frame surrounding the input image. We will thus control the actual size of the adversarial program in our experiments by varying the number of pixels that are used by the adversarial program, i.e., by changing the size of the reprogramming mask. The size of the reprogramming mask can be measured by simply counting the number of ones in $\mathbf{M}$, e.g., by computing $\|\mathbf{M}\|_1$.

2.3. Extension to other perturbation models

As discussed in Sect. 2.2, the success of reprogramming mostly depends on the minimization of the second term of Eq. (2), whose minimum, under linearization, is given as $\Delta L(\boldsymbol{\delta}^\star) = \boldsymbol{\delta}^\top \boldsymbol{g} = -\|\boldsymbol{g}\|_1$.

Interestingly, it is not difficult to see that the ℓ_1 norm of $\boldsymbol{g}$ naturally appears here as it corresponds to the dual norm of the ℓ_∞ norm used in Eq. (1) to bound the adversarial perturbation $\boldsymbol{\delta}$. While this result may not be surprising at first sight, as the dual norm is obtained by definition as the maximization of a scalar product over a unit ball [6], it allows us to extend our model to adversarial programs optimized with different perturbation constraints. For example, one may use a generic ℓ_p-norm constraint (with $p = 1, 2, \infty$) bounded by a small perturbation size ϵ, i.e., $\|\boldsymbol{\delta}\|_p \leq \epsilon$, to superimpose the adversarial program in an imperceptible manner on the input image, and found that the optimal $\Delta L(\boldsymbol{\delta}^\star)$ is simply given as $\Delta L(\boldsymbol{\delta}^\star) = -\epsilon \|\boldsymbol{g}\|_q$, being q the dual norm of p.

We conclude this section by pointing out that, while similar findings have been derived in [7] to study the behavior of adversarial examples in high-dimensional spaces, our model extends the analysis in [7] to account for adversarial perturbations which are optimized *over* different input samples (and not just on a single input image). Such perturbations do not only include adversarial programs, but they also encompass universal adversarial perturbations [8] and robust adversarial examples [9,10], making our model readily applicable to study and quantify also the effectiveness of such attacks.

3. Experimental analysis

We report here an extensive experimental analysis aimed to evaluate the impact of the factors identified by the mathematical model presented in Sect. 2 on reprogramming accuracy.

3.1. Experimental setup

We consider 14 different reprogramming tasks and 3 model architectures, focusing on deep neural networks trained to perform computer-vision tasks. In the following, we provide the details required to reproduce our empirical analysis.

Datasets. Our experimental analysis considers four different computer-vision datasets: two object recognition datasets, i.e., ImageNet and CIFAR-10, and two datasets containing images of handwritten Arabic (MNIST) and Chinese (HCL2000) digits.

ImageNet[2] is one of the largest publicly-available computer-vision datasets. It contains images belonging to 1,000 categories subdivided in around 1.2 million training images and 150,000 test images. The images are collected from Internet by search engines and labeled by humans via crowdsourcing. We use this dataset only as a source-domain dataset, as many pretrained models on ImageNet are readily available.

[2] https://www.image-net.org/.

Table 1
Architecture of the CWNet network trained on digit images in [2].

Layer Type	Dimension
Conv. + ReLU	32 filters (3×3)
Conv. + ReLU	32 filters (3×3)
Max Pooling	2×2
Conv. + ReLU	64 filters (3×3)
Conv. + Dropout (0.5%) + ReLU	64 filters (3×3)
Max Pooling	2×2
Fully Connected + ReLU	200 units
Fully Connected + ReLU	200 units
Softmax	10 units

CIFAR-10[3] is a ten-class image-classification dataset made up of small resolution (32×32) color images, subdivided into 50,000 training and 10,000 test images.
MNIST[4] is a ten-class dataset containing images of handwritten Arabic digits. It consists of grayscale images of size 28×28, subdivided in 60,000 training and 10,000 test images.
HCL2000[5] is a large-scale handwritten Chinese character database [11], containing grayscale images of size 64×64. We consider only the ten classes corresponding to the Chinese digits, which result into 7,000 training and 3,000 test images.

Dataset splits. To train the models, we have used the full training dataset of the source domain dataset. We have used 5000 samples randomly sampled from the training dataset of the target domain dataset to compute the adversarial program and 1000 samples, randomly sampled from the test set of the target domain dataset to compute the performance metrics.

Preprocessing. We rescale the input images in $\mathcal{X}=[-1,1]^d$, with $d=224\times224\times3$, to match the input size of the considered models. This requires padding input images with zeros if they are smaller in size, and replicating the image content on each color channel for single-channel grayscale images (like in the case of MNIST and Chinese digits).

Classifiers. We consider three different neural-network architectures: the CWNet network proposed in [2] (Table 1); and the pretrained (and thus trained on the full training dataset) ImageNet models AlexNet [1] and EfficientNet [12]. All the considered models have input size $d=224\times224\times3$, and apply z-score batch normalization before processing the input samples. CWNet is only trained using MNIST and HCL2000 as the source-domain data. AlexNet and EfficientNet are instead used with ImageNet, MNIST and HCL2000 as source-domain datasets. For AlexNet and EfficientNet, we also consider a setting in which their weights are set to random values, which amounts to considering their *untrained* versions as done in [3]. When training the considered models on MNIST and HCL2000, we run stochastic gradient descent for 10 epochs, with step size, momentum, and batch size respectively equal to 0.001, 0.9, and 10. For the step size, the number of epochs, and batch size, we chose the values that lead to a higher accuracy on a validation dataset of 1000 samples sampled from the training dataset belonging to the source domain.

Adversarial programs. To optimize the adversarial program $\boldsymbol{\delta}$, we use Algorithm 1. Before optimizing it, we fix the class-mapping h as a function that maps orderly the first ten classes of the source dataset to the first ten classes of the target dataset, as done in [3]. (The first class of the source domain dataset is mapped to the first class of the target domain dataset, the second of the source domain in the second of the target domain dataset, etc.). As in [3], for the datasets having more than classes than the target domain dataset, which in our case is only ImageNet, which contains more than ten classes, we only consider the first ten classes (i.e., tench, goldfish, white shark, tiger shark, hammerhead, electric ray, stingray, cock, hen, ostrich). Namely, we crop the output layer so that it contains 10 elements (one for each of the abovementioned ten classes, and we consider as the predicted class, the one that received a higher score between these ten classes). Note that the choice of the mapping function may affect the accuracy; however as we will explain in the following, it will not impact our claims. We set the step size $\eta=0.005$, $N=100$ epochs, and we use batches of $B=50$ samples, sampled from a larger set of $5,000$ images randomly drawn from the training set of the target-domain dataset $\mathcal{T}$.

Performance metrics. We consider different metrics to evaluate the performance of reprogramming along with the underlying factors affecting its success. In particular, we consider three metrics:

- *reprogramming accuracy (RA)*, i.e. the ratio of samples of the target-domain dataset correctly classified after the application of the adversarial program. A higher RA means that the adversarial program is more effective in repurposing the model, thus the higher, the better;
- *domain alignment (DA)*, evaluated as the model's accuracy on the target-domain samples padded with zeros, i.e., before optimizing the adversarial program $\boldsymbol{\delta}$. A higher DA means that the first term of Eq. (2) is low, which means that the unperturbed samples from the target domain are already consistently assigned to the desired source-domain classes with high probability. Thus, even small perturbations may provide high reprogramming accuracy.

[3] https://www.cs.toronto.edu/~kriz/cifar.html.
[4] http://yann.lecun.com/exdb/mnist/.
[5] http://www.pris.net.cn/introduction/teacher/lichunguang.

Table 2
Results of reprogramming AlexNet, EfficientNet, and CWNet from different source ($\mathcal{S}$) to target ($\mathcal{T}$) domains. U means that the network has not been trained (its weights are randomly initialized). For each reprogramming task, the table reports domain alignment (DA), reprogramming accuracy (RA), and gradient alignment (Eq. (5)) computed before (r_0) and after (r_N) optimizing the adversarial program δ. For all the reported metrics, the higher the value the better. The cases in which reprogramming works well/poorly/badly are highlighted in green/yellow/red.

$\mathcal{S}$	$\mathcal{T}$	Model	DA	RA	r_0	r_N
ImageNet	HCL2000	EfficientNet	5.3%	98.1%	18.4%	21.14%
ImageNet	HCL2000	AlexNet	2.5%	97.2%	19.3%	19.35%
ImageNet	MNIST	EfficientNet	14.3%	90.6%	17.5%	20.33%
ImageNet	MNIST	AlexNet	11.6%	90.1%	29.0%	18.75%
ImageNet	CIFAR-10	EfficientNet	10.2%	51.1%	13.5%	10.54%
ImageNet	CIFAR-10	AlexNet	9.3%	46.0%	13.6%	11.28%
MNIST	HCL2000	AlexNet	12.8%	49.1%	14.5%	9.11%
MNIST	HCL2000	CWNet	9.5%	45.3%	16.4%	13.59%
HCL2000	MNIST	AlexNet	9.9%	21.8%	92.1%	5.27%
HCL2000	MNIST	CWNet	9.9%	19.1%	92.8%	4.82%
U	MNIST	AlexNet	11.8%	20.5%	5.0%	6.0%
U	HCL2000	AlexNet	9.0%	28.7%	6.7%	6.36%
U	MNIST	EfficientNet	11.8%	11.8%	4.0%	3.38%
U	HCL2000	EfficientNet	9.0%	9.0%	6.9%	5.72%

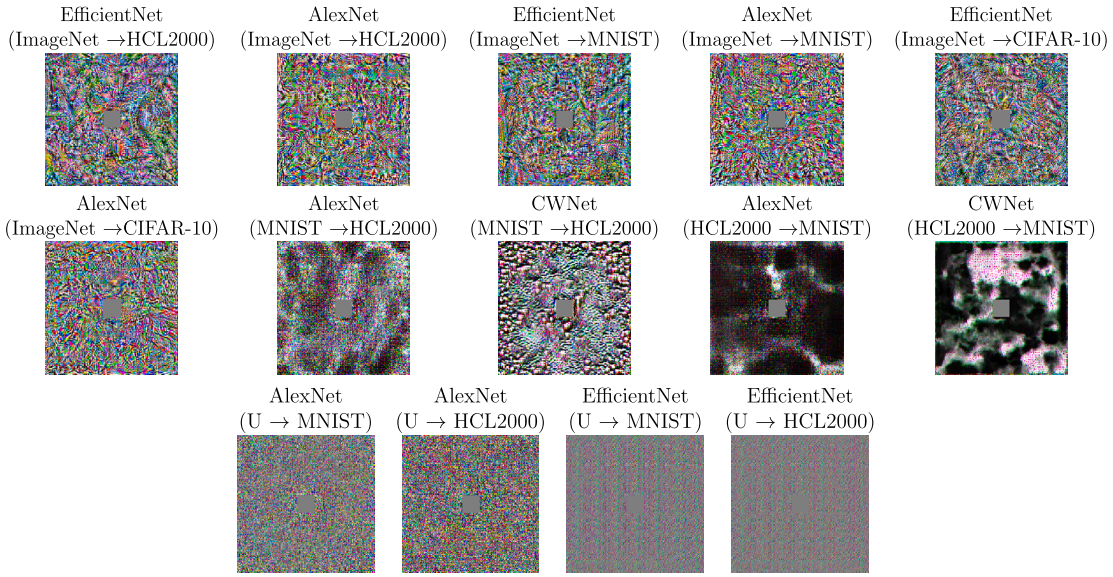

Fig. 3. Adversarial programs optimized to repurpose the networks in Table 2 from different source to target ($\mathcal{S} \to \mathcal{T}$) domains.

- *gradient alignment (Eq. (5)) before (r_0) and after (r_N) reprogramming*, i.e., after, respectively, the first and the last iteration of Algorithm 1. Accordingly to our analysis, the gradient alignment helps perform reprogramming; thus, it is better when it is high. (Intuitively, if the perturbations that should be applied to each sample to reprogram the model are aligned, finding a single universal perturbation to reprogram the model is easier).

3.2. Experimental results

The experimental results on the given 14 reprogramming tasks are reported in Table 2, while the corresponding adversarial programs are shown in Fig. 3. As highlighted in Table 2 with different colors, adversarial reprogramming may exhibit different *reprogramming accuracy* (RA) values: it may work remarkably well (RA $\geq$ 90%), it may work poorly (RA $\approx$ 50%), or it may even completely fail (RA $\leq$ 30%). In the remainder of this section, we analyze the impact on reprogramming accuracy of the main factors identified by our mathematical model in Sect. 2, i.e., (*i*) the alignment between source and target domain, (*ii*) the alignment of the input gradients, and (*iii*) the size of the reprogramming mask. In the following paragraphs (one for each of these factors), we recap our claims (in italic), then present the empirical results. Note that from our experimental analysis, all these factors help perform reprogramming. However, in the following, we assess to which extent they contribute to the reprogramming success in practice.

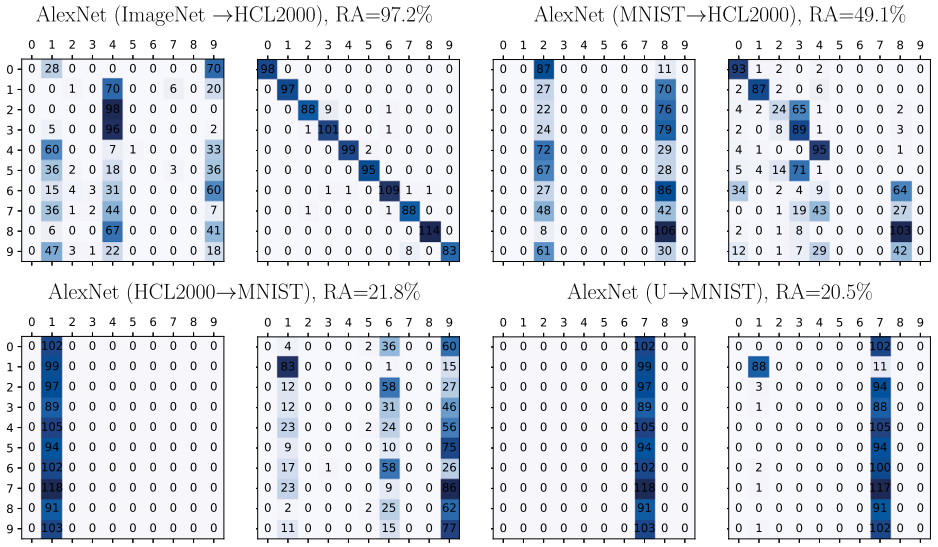

Fig. 4. Confusion matrices (true-vs-predicted classes) on four representative cases. For each case, we report the confusion matrix before (*left*) and after (*right*) reprogramming.

Source and target domain alignment. Let us now analyze the influence of the source-target domain alignment on reprogramming accuracy. *As we have explained in Sect. 2, reprogramming accuracy also depends on the classifier loss on the target-domain dataset before reprogramming (the first term of Eq. (2)). (It determines the reprogramming accuracy before optimizing the program).* In Table 2, we report the accuracy of the classifier on the target-domain dataset before reprogramming, referred to as *domain alignment* (DA). The table shows that DA is usually quite low (around 10%), even when reprogramming accuracy is very high (RA > 90%), which means that the source and target alignment is not correlated with the reprogramming accuracy. Namely, this factor does not have a prominent impact on reprogramming accuracy. Even if the source and target domain are quite different, reprogramming can be successful because the program optimization impacts more than the reprogramming accuracy of the model on the target domain before optimizing the program. To explain why domain alignment is low, in Fig. 4 we report the confusion matrices computed before (left plot) and after (right plot) reprogramming. These matrices show that, before reprogramming, the target-domain samples are often assigned to a single class or a few classes. The reason is that, before reprogramming, the target-domain samples are simply padded with zeros, thus being more likely to be classified similarly.

Gradient alignment. Here, we analyze the impact of gradient alignment on reprogramming accuracy. *As we have explained in Sect. 2, the gradient alignment influences how much the reprogramming accuracy increase when optimizing the program (the higher, the better).* For each classifier and reprogramming task, we compute the gradient-alignment metric r (Eq. (5)) before (r_0) and after reprogramming (r_N). The reason is that, as previously explained, before reprogramming, the target-domain samples are simply padded with zeros and tend to be assigned to one or few classes. Accordingly, the input gradients computed before optimizing the program are not expected to be really informative, while they are expected to be more informative when the program is optimized and reprogramming accuracy starts increasing.

The values of r_0 and r_N for the given reprogramming tasks are reported in Table 2. In Fig. 5 we also report the correlation between gradient alignment and reprogramming accuracy, computed using Pearson (P), Spearman (S), and Kendall (K) methods, along with the corresponding permutation tests and p-values. The correlation between reprogramming accuracy RA and gradient alignment before reprogramming (r_0) is not significant ($p > 0.05$) mostly due to the presence of two outlying observations (i.e., AlexNet HCL2000 → MNIST, and CWNet HCL2000 → MNIST). The correlation values are much higher and statistically significant, instead, when considering gradient alignment after reprogramming (r_N), e.g., the correlation computed with the Pearson coefficient is 0.98 with $p < 1e-8$. These results show that gradient alignment, especially when computed after reprogramming (r_N), is strongly and positively correlated with reprogramming accuracy, thus confirming the intuition provided by our mathematical model.

Reprogramming mask size. We finally analyze the impact of the reprogramming mask size on reprogramming accuracy. *As we have explained in Sect. 2, the reprogramming mask size influences how much the reprogramming accuracy increase when optimizing the program (the higher, the better).* To this end, we consider reprogramming masks of increasing sizes, using 64, 128, and 224 as their width and height values. We report reprogramming accuracy and the proposed measures computed for each classifier and reprogramming task in Table 3, and the corresponding correlation tests in Fig. 6. While the right plot in Fig. 6 shows again that reprogramming accuracy is strongly correlated with the gradient alignment r_N, also for such reprogramming cases, the left plot shows that reprogramming accuracy is also correlated with the reprogramming mask size, confirming again the soundness of the proposed mathematical model. From the values in Table 3, it should also be noted that the reprogramming mask size can have a relevant impact on reprogramming accuracy, e.g., in the case AlexNet ImageNet → MNIST, there is an accuracy difference of 15% between the performance obtained with the smallest and that obtained with the largest mask size considered.

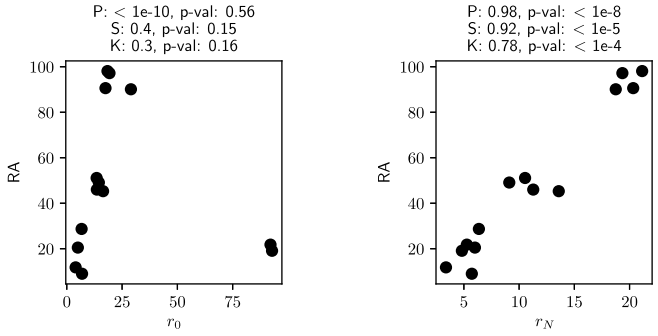

Fig. 5. Correlation between the reprogramming accuracy (RA) vs the gradient alignment computed before r_0 (left) and after r_N (right) optimizing the program.

Table 3
Results for programs with different reprogramming mask sizes. See the caption of Table 2 for further details.

Mask Size	$\mathcal{S}$	$\mathcal{T}$	Model	RA	r_0	r_N
224	ImageNet	HCL2000	EfficientNet	98.1%	18.4%	21.14%
128	ImageNet	HCL2000	EfficientNet	96.0%	18.5%	14.52%
64	ImageNet	HCL2000	EfficientNet	89.9%	18.6%	17.81%
224	ImageNet	HCL2000	AlexNet	97.2%	19.3%	19.35%
128	ImageNet	HCL2000	AlexNet	97.1%	17.4%	16.82%
64	ImageNet	HCL2000	AlexNet	88.4%	16.1%	16.39%
224	ImageNet	MNIST	EfficientNet	90.6%	17.5%	20.33%
128	ImageNet	MNIST	EfficientNet	84.0%	18.2%	17.15%
64	ImageNet	MNIST	EfficientNet	70.6%	17.3%	8.17%
224	ImageNet	MNIST	AlexNet	90.1%	29.0%	18.75%
128	ImageNet	MNIST	AlexNet	84.6%	17.0%	11.36%
64	ImageNet	MNIST	AlexNet	76.5%	16.7%	7.17%

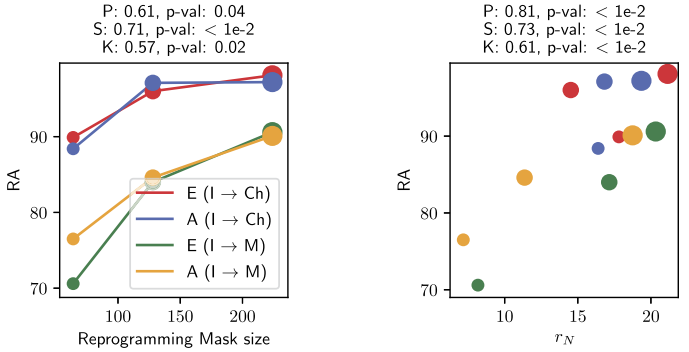

Fig. 6. Reprogramming accuracy (RA) versus reprogramming mask size (left) and the gradient alignment computed after reprogramming r_N (right). Considering AlexNet (A), EfficientNet (E), and the datasets: ImageNet (I), HCL2000 (Ch), MNIST (M).

On the choice of the label mapping. The label mapping function can affect accuracy, as shown in [4]. As explained by the authors of that paper, the accuracy of the reprogrammed model on the target task can be improved in two ways. The first is to map the labels depending on the models' predictions before reprogramming (see "frequency-based mapping" in [4]), and this is always applicable. The second is the so-called multi-label mapping that maps one single label of the target domain into multiple labels of the source domain. This method is applicable only when the number of classes of the source domain is bigger than the number of classes of the target domain. The only reprogramming tasks (between the ones considered in our work) to which this technique is applicable are the ones that reprogram models trained on ImageNet to work on the HCL2000, MNIST, and CIFAR-10 datasets. These are cases in which reprogramming works better; hence, an eventual accuracy improvement would not affect our study's outcome. In the other reprogramming tasks, in which reprogramming does not work, the source datasets have the same number of classes as the target datasets; therefore, the only technique that could be applied is frequency-based mapping. However, this technique will only have a minimal impact on the accuracy because, as visible from the confusion matrices in Fig. 4, the model assigns all the samples of the target domain to very few classes. We have checked, and it is also true for the confusion matrices we have not reported in

the manuscript. Therefore, frequency-based mapping can find the best mapping so that the majority of the samples belonging to those classes are predicted correctly. However, this will help with at most 3 of the ten classes (the other classes are predicted only sporadically). Moreover, from [4], it is clear that this technique can achieve only marginal improvements. Using this technique instead of random mapping, they have always gained less than 5% accuracy (see the difference between "AR + Rand mapping" and "AR+ Freq. Mapping" in Figure 3 of [4]. Therefore, also, in this case, applying this technique, our claim will not change.

Summary of the results. In this work, we develop a first-order linear model that expresses the reprogramming optimization problem as a sum of two terms. The first term measures how difficult it is to reprogram a given machine-learning model from the source to the target domain without optimizing the adversarial program. The second term is the loss decrement that we can obtain by optimizing the adversarial program and, as we showed, depends on the size of the average input gradient. From this mathematical model and our empirical study of the relationship between these terms and the reprogramming accuracy, we derived the answer to the question asked in the title: why adversarial reprogramming works, when it fails, and how to tell the difference? Adversarial reprogramming consists of applying a single (universal) perturbation to the target-domain samples to move them into a region of the decision space where they are classified as desired. It works because, when the size of the average input gradient of the target model is large enough, a small universal perturbation is enough to reprogram the model. Which happens when (i) the gradients that should be applied to every single sample to reprogram the model are sufficiently aligned, and (ii) the size of the reprogramming mask is sufficiently large. The latter plays a much more marginal role than the former, as small mask sizes are usually already sufficient to achieve good reprogramming performance. Also, the source and target domains alignment does not have a relevant impact on the reprogramming accuracy, as it works successfully even when the source and target domains are poorly aligned. Adversarial reprogramming fails when the size of the average input gradient is not large enough, as in the paradigmatic example of reprogramming failure we described in the introduction (Fig. 1 bottom). In that example, the goal was to reprogram a small convolutional neural network (CWNet), trained to recognize handwritten Chinese digits (depicted in Fig. 1 (bottom)), to recognize the MNIST handwritten (Arabic) digits. However, in that case, reprogramming fails as the gradient alignment after reprogramming (r_N), which we have shown to be correlated with the reprogramming accuracy, is very low (see Table 2). The intuition is that if the perturbations you should apply to every sample to reprogram the model are not sufficiently similar (the input gradients are not aligned), you cannot find a single perturbation able to reprogram the model. Namely, a single perturbation that can be employed to move the samples of the different classes into a region of the decision space where they are classified as desired. To summarize, the main findings of our extensive experimental analysis are:

- source and target domain alignment, in practice, does not affect reprogramming accuracy;
- gradient alignment plays a key role in the success of reprogramming;
- larger reprogramming mask sizes facilitate reprogramming but have a much more marginal impact on its success.

Additional comments. Here, we provide further comments on the experimental results. On the basis of the reprogramming accuracy reported in Table 2, we argue that reprogramming works better when the reprogramming task is simple and the function learned by the target classifier is complex. Table 2 also shows that reprogramming works well when the target model is a large model trained on a large and complex dataset and reprogrammed to perform a simple task (such as handwritten digit classification). On the other hand, it does not work well: (i) when learning on the target domain represents a complex task, (ii) when the target model is small, and (iii) when the target model is complex but trained on a simple and small dataset. Whereas the first and the second case support our hypothesis, the latter deserves further clarification. When a complex classifier is trained on a small and simple dataset, the representations learned by the different network layers tend to become very similar [13], and some of the layers thus do not alter the classifier prediction [14]. Therefore, although the classifier is complex, the function learned by the classifier remains simple.

4. Related work

In this section, we briefly review related work on reprogramming. We then focus on attacks that optimize adversarial perturbations, including universal adversarial perturbations and robust physical-world adversarial examples. Finally, we review work that provides additional insights into the vulnerability of machine-learning models to adversarial perturbations based on different first-order linear analyses.

4.1. Adversarial reprogramming

Adversarial reprogramming has been originally proposed in [3]. The authors have empirically assessed the performance of adversarial reprogramming using different trained and untrained deep neural networks. They showed that reprogramming usually fails when applied to untrained networks (i.e., neural networks with random weights), whereas it works when the target model is trained. In the latter case, reprogramming works even when the attacker can manipulate only a small subset of the image pixels. However, the authors have not explained why reprogramming works and when it fails, leaving this analysis to future work. Although subsequent work has successfully applied reprogramming in different scenarios [4,15,16], no work has analyzed the reasons behind its success and failure. It is also worth remarking that, while Yang et al. [36] have identified some factors affecting why reprogramming works in the audio domain, they have not discussed when it can fail and which factors may lead to its failure. We do believe that our work is thus the first to provide a more detailed and quantitative analysis of the impact of the main factors underlying the success and failure of adversarial reprogramming.

4.2. Universal and robust adversarial perturbations

Gradient-based attacks on machine-learning models [17–20] have been demonstrated in a variety of application domains, including computer vision and security-related tasks [21–26], even before the independent discovery of adversarial examples against deep neural networks [27].

While earlier work has focused on optimizing a different adversarial perturbation for each input sample, in [8] the authors have shown that it is even possible to optimize a single, *universal adversarial perturbation*, i.e., a fixed perturbation that can be applied to many different input samples to have them misclassified as desired. The underlying idea is to optimize the perturbation on different input samples, similar to the idea behind the optimization of robust physical-world adversarial examples [9,10], i.e., to optimize the adversarial perturbation over different transformations of the input image (e.g., subject to changes in pose, rotation, illumination).

In this work, we argue that the mathematical formulation of universal perturbations, robust adversarial examples, and adversarial reprogramming is essentially the same, i.e., all these attacks require optimizing the adversarial perturbation by averaging the loss function over different input images (even though reprogramming optimizes the perturbation to repurpose a model, while the other attacks aim to have input samples misclassified). For this reason, we believe that our analysis can be readily applied in future work to provide a better understanding of the effectiveness of both universal adversarial perturbations and robust adversarial examples.

4.3. First-order analysis of adversarial perturbations

Previous work has analyzed the vulnerability of neural networks against adversarial examples and universal adversarial perturbations. The authors of [28] have proven the existence of small universal perturbations, attributing them to the low curvature of the decision boundaries of deep neural networks [29,30]. The work in [7] is probably the closest one to ours, as it also builds on the previously-proposed idea of modeling the optimization of adversarial examples as a linear problem [31]. The same idea has also been explored to develop robust methods based on regularizing the input gradients [32–34], as well as to provide deeper insights on why adversarial examples can often *transfer* across different models [35]. The main difference between our work and the work in [7] is that our model extends their analysis to also encompass adversarial perturbations which are optimized on different samples, thereby not only including adversarial examples, but also adversarial reprogramming, universal adversarial perturbations, and robust physical-world adversarial examples.

5. Contributions, limitations and future work

Adversarial reprogramming has been originally proposed as an attack aimed to abuse machine-learning models provided as a service. However, it has been subsequently shown that such a technique may also be used beneficially, providing a valuable approach to transfer learning. Despite its great practical relevance, no previous work has explained the main factors affecting the performance of adversarial reprogramming, i.e., why it works, when it fails, and how to tell the difference.

In this work, we have overcome this limitation by providing a first-order linear analysis of adversarial reprogramming, which sheds light on the underlying factors influencing reprogramming accuracy. We have then performed an extensive experimental analysis involving different machine-learning models and fourteen different reprogramming tasks. Thanks to our theoretical and empirical analyses, we have shown that the success of reprogramming depends on the size of the average input gradient, which is larger when the input gradients are more aligned, and when inputs have higher dimensionality. Our work thus provides a first concrete step towards analyzing the success and failure of adversarial reprogramming, paving the way to future work that may enable the development of better defenses against adversarial reprogramming, and improved transfer-learning algorithms. An interesting future development of this work also includes deriving guidelines to help practitioners to select machine-learning models which are easier to repurpose for a different task, thereby facilitating the process of transfer learning.

Two limitations currently exist in our work. The first is that it is not immediately clear from our analysis whether and to which extent the number of source- and target-domain classes may have an impact on the performance of adversarial reprogramming, and this aspect certainly deserves a more detailed empirical investigation in the future. The second limitation is that we have only considered adversarial programs optimized within an l_∞-norm constraint in this work. Nevertheless, our analysis can be easily extended to other l_p-norm perturbation models, as discussed in Sect. 2.3, and it can also be exploited to provide deeper insights on attacks in which the adversarial perturbation is averaged over different input samples (including universal adversarial perturbations and robust physical-world adversarial examples), as discussed in Sect. 4.3. We firmly believe that exploring these research directions will certainly provide a promising avenue for future work.

CRediT authorship contribution statement

Yang Zheng: Investigation, Software, Visualization, Writing – original draft. **Xiaoyi Feng:** Funding acquisition, Resources, Supervision. **Zhaoqiang Xia:** Supervision, Writing – review & editing. **Xiaoyue Jiang:** Writing – original draft. **Ambra Demontis:** Methodology, Software, Visualization, Writing – review & editing. **Maura Pintor:** Methodology, Software, Visualization, Writing – review & editing. **Battista Biggio:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Fabio Roli:** Funding acquisition, Resources, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data used for this research are already publicly available and the relative link is reported in the paper.

Acknowledgements

This work was partly supported by the PRIN 2017 project RexLearn, funded by the Italian Ministry of Education, University and Research (grant no. 2017TWNMH2); by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI; and by the Key Research and Development Program of Shaanxi (Program Nos. 2022ZDLGY06-07, 2021ZDLGY15-01, 2021ZDLGY09-04 and 2021GY-004), the International Science and Technology Cooperation Research Project of Shenzhen (GJHZ20200731095204013), the National Natural Science Foundation of China (Grant No. 61772419).

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), NIPS 25, 2012, pp. 1097–1105.
[2] N. Carlini, D.A. Wagner, Towards evaluating the robustness of neural networks, in: IEEE Symp. on Sec. and Privacy, 2017, pp. 39–57.
[3] G.F. Elsayed, I. Goodfellow, J. Sohl-Dickstein, Adversarial reprogramming of neural networks, in: ICLR, 2019.
[4] Y.-Y. Tsai, P.-Y. Chen, T.-Y. Ho, Transfer learning without knowing: reprogramming black-box machine learning models with scarce data and limited resources, in: H.D. III, A. Singh (Eds.), ICML, in: PMLR, vol. 119, 2020, pp. 9614–9624.
[5] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, in: ICLR, 2018.
[6] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.
[7] C.-J. Simon-Gabriel, Y. Ollivier, L. Bottou, B. Schölkopf, D. Lopez-Paz, First-order adversarial vulnerability of neural networks and input dimension, in: K. Chaudhuri, R. Salakhutdinov (Eds.), ICLM, in: PMLR, vol. 97, 2019, pp. 5809–5817.
[8] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, P. Frossard, Universal adversarial perturbations, in: CVPR, 2017.
[9] A. Athalye, L. Engstrom, A. Ilyas, K. Kwok, Synthesizing robust adversarial examples, in: ICLR, 2018.
[10] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, D. Song, Robust physical-world attacks on deep learning visual classification, in: CVPR, 2018, pp. 1625–1634.
[11] H. Zhang, J. Guo, G. Chen, C. Li, HCL2000 - a large-scale handwritten Chinese character database for handwritten character recognition, in: Int'l Conf. on Doc. Analysis and Rec., 2009, pp. 286–290.
[12] M. Tan, Q. Le, EfficientNet: rethinking model scaling for convolutional neural networks, in: K. Chaudhuri, R. Salakhutdinov (Eds.), ICML, in: PMLR, vol. 97, 2019, pp. 6105–6114.
[13] T. Nguyen, M. Raghu, S. Kornblith, Do wide and deep networks learn the same things? Uncovering how neural network representations vary with width and depth, in: ICLR, 2021.
[14] R.K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: NIPS, vol. 2, 2015, pp. 2377–2385.
[15] P. Neekhara, S. Hussain, S. Dubnov, F. Koushanfar, Adversarial reprogramming of text classification neural networks, in: Conf. on Empirical Methods in NLP and the 9th Int'l Joint Conf. on NLP (EMNLP-IJCNLP), 2019, pp. 5216–5225.
[16] P. Neekhara, S. Hussain, J. Du, S. Dubnov, F. Koushanfar, J. McAuley, Cross-modal adversarial reprogramming, ArXiv, 2021.
[17] B. Biggio, B. Nelson, P. Laskov, Poisoning attacks against support vector machines, in: J. Langford, J. Pineau (Eds.), ICML, 2012, pp. 1807–1814.
[18] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, F. Roli, Evasion attacks against machine learning at test time, in: H. Blockeel, K. Kersting, S. Nijssen, F. Železný (Eds.), ECML PKDD, in: LNCS, vol. 8190, 2013, pp. 387–402.
[19] B. Biggio, F. Roli, Wild patterns: ten years after the rise of adversarial machine learning, Pattern Recognit. 84 (2018) 317–331.
[20] A.D. Joseph, B. Nelson, B.I.P. Rubinstein, J. Tygar, Adversarial Machine Learning, Cambridge University Press, 2018.
[21] M. Melis, A. Demontis, B. Biggio, G. Brown, G. Fumera, F. Roli, Is deep learning safe for robot vision? Adversarial examples against the iCub humanoid, in: ICCVW (ViPAR), IEEE, 2017, pp. 751–759.
[22] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, F. Roli, Yes, machine learning can be more secure! A case study on Android malware detection, IEEE Trans. Dependable Secure Comput. 16 (4) (2019) 711–724.
[23] F. Pierazzi, F. Pendlebury, J. Cortellazzi, L. Cavallaro, Intriguing properties of adversarial ML attacks in the problem space, in: IEEE Symp. on Sec. and Privacy, 2020, pp. 1332–1349.
[24] D. Maiorca, A. Demontis, B. Biggio, F. Roli, G. Giacinto, Adversarial detection of flash malware: limitations and open issues, Comput. Secur. 96 (2020) 101901.
[25] H.S. Anderson, A. Kharkar, B. Filar, D. Evans, P. Roth, Learning to evade static PE machine learning malware models via reinforcement learning, ArXiv.
[26] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, A. Armando, Functionality-preserving black-box optimization of adversarial Windows malware, IEEE TIFS.
[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: ICLR, 2014.
[28] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, P. Frossard, S. Soatto, Robustness of classifiers to universal perturbations: a geometric perspective, in: ICLR, 2018.
[29] A. Fawzi, S.-M. Moosavi-Dezfooli, P. Frossard, S. Soatto, Empirical study of the topology and geometry of deep networks, in: CVPR, 2018, pp. 3762–3770.
[30] S.-M. Moosavi-Dezfooli, A. Fawzi, J. Uesato, P. Frossard, Robustness via curvature regularization, and vice versa, in: CVPR, 2019.
[31] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: ICLR, 2015.
[32] C. Lyu, K. Huang, H.-N. Liang, A unified gradient regularization family for adversarial examples, in: ICDM, IEEE Comp. Soc., Los Alamitos, CA, USA, 2015, pp. 301–309.
[33] D. Varga, A. Csiszárik, Z. Zombori, Gradient regularization improves accuracy of discriminative models, ArXiv.
[34] A.S. Ross, F. Doshi-Velez, Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing Their Input Gradients, AAAI, 2018.
[35] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, F. Roli, Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks, USENIX Sec., 2019.
[36] C. Yang, Y. Tsai, P. Chen, Voice2series: reprogramming acoustic models for time series classification, ICML (2021) 11808–11819.

[37] H. Yen, P. Ku, C. Yang, H. Hu, S. Siniscalchi, P. Chen, Y. Tsao, A study of low-resource speech commands recognition based on adversarial reprogramming, ArXiv, 2021.
[38] P. Neekhara, S. Hussain, S. Dubnov, F. Koushanfar, Adversarial reprogramming of text classification neural networks, in: EMNLP-IJCNLP, 2019, pp. 5216–5225.

Yang Zheng received his M.S. degree from the School of Electronic Engineering, Xi'an University of Posts & Telecommunications, China, in 2018. He is currently pursuing his Ph.D. in the School of Electronics and Information, Northwestern Polytechnical University. His current research interests include secure machine learning and deep learning.

Xiaoyi Feng is a Professor with the School of Electronics and Information, Northwestern Polytechnical University. She has authored or coauthored more than 100 papers in journals and conferences. Her current research interests include computer vision, image process, radar imagery, and recognition.

Zhaoqiang Xia is an Associate Professor in the School of Electronics and Information, Northwestern Polytechnical University. He has authored or coauthored more than 60 papers in international journals and conferences. His current research interests include visual processing, search and recognition, and statistical machine learning.

Xiaoyue Jiang received the M.S. and Ph.D. degree from Northwestern Polytechnical University, Xi'an, China, in 2003 and 2006, respectively. She is an associate professor with the School of Electronics and Information, Northwestern Polytechnical University since 2012. Her research interests include image processing, computer vision and machine learning.

Ambra Demontis is an Assistant Professor at the University of Cagliari, Italy. She received her M.Sc. degree (Hons.) in Computer Science and her Ph.D. degree in Electronic Engineering and Computer Science from the University of Cagliari, Italy. Her research interests include secure machine learning, kernel methods, biometrics, and computer security.

Maura Pintor is a Postdoctoral Researcher at the University of Cagliari, Italy. She received her Ph.D. degree in Electronic and Computer Engineering (with honors) from the University of Cagliari (Italy) in 2022. Her research interests include adversarial machine learning and machine learning explainability methods, with applications in cybersecurity.

Battista Biggio (MSc 2006, PhD 2010) is Assistant Professor at the University of Cagliari, Italy, and co- founder of the company Pluribus One. His research interests include adversarial machine learning and cybersecurity. He is Senior Member of the IEEE and of the ACM, and Member of the IAPR and ELLIS.

Fabio Roli is a Full Professor of Computer Science at the University of Genoa, Italy. He has been appointed Fellow of the IEEE and Fellow of the International Association for Pattern Recognition. He is a recipient of the Pierre Devijver Award for his contributions to statistical pattern recognition.