Adversarial Reprogramming of Neural Networks

Gamaleldin F. Elsayed[*]
Google Brain
gamaleldin.elsayed@gmail.com

Ian Goodfellow
Google Brain
goodfellow@google.com

Jascha Sohl-Dickstein
Google Brain
jaschasd@google.com

Abstract

Deep neural networks are susceptible to *adversarial* attacks. In computer vision, well-crafted perturbations to images can cause neural networks to make mistakes such as identifying a panda as a gibbon or confusing a cat with a computer. Previous adversarial examples have been designed to degrade performance of models or cause machine learning models to produce specific outputs chosen ahead of time by the attacker. We introduce adversarial attacks that instead *reprogram* the target model to perform a task chosen by the attacker—without the attacker needing to specify or compute the desired output for each test-time input. This attack is accomplished by optimizing for a single adversarial perturbation, of unrestricted magnitude, that can be added to all test-time inputs to a machine learning model in order to cause the model to perform a task chosen by the adversary when processing these inputs—even if the model was not trained to do this task. These perturbations can be thus considered a program for the new task. We demonstrate adversarial reprogramming on six ImageNet classification models, repurposing these models to perform a counting task, as well as two classification tasks: classification of MNIST and CIFAR-10 examples presented within the input to the ImageNet model.

1 Introduction

The study of adversarial examples is often motivated in terms of the danger posed by an attacker whose goal is to cause model prediction errors by making a small change to the model's input. Such an attacker could make a self-driving car react to a phantom stop sign [1] by means of a sticker (a small L_0 perturbation), or cause an insurance company's damage model to overestimate the claim value from the resulting accident by subtly doctoring photos of the damage (a small L_∞ perturbation). With this context in mind, various methods have been proposed both to construct [2–7] and defend against [8–13] this style of adversarial attack. Thus far, the majority of adversarial attacks have consisted of *untargeted* attacks that aim to degrade the performance of a model without necessarily requiring it to produce a specific output, or *targeted* attacks in which the attacker designs an adversarial perturbation of an input to produce a specific output for that input. For example, an attack against a classifier might target a specific desired output class for each input image, or an attack against a reinforcement learning agent might induce that agent to enter a specific state [14].

In this work, we consider a more complicated attacker goal: inducing the model to perform a task chosen by the attacker, without the attacker needing to compute the specific desired output. Consider a model trained to perform some *original task*: for inputs x it produces outputs $f(x)$. Consider an adversary who wishes to perform an *adversarial task*: for inputs $\tilde{x}$ (not necessarily in the same domain as x) the adversary wishes to compute a function $g(\tilde{x})$. We show that an adversary can accomplish this by learning *adversarial reprogramming functions* $h_f(\cdot;\theta)$ and $h_g(\cdot;\theta)$ that map between the two tasks. Here, h_f converts inputs from the domain of $\tilde{x}$ into the domain of x (i.e., $h_f(\tilde{x};\theta)$ is a valid input to the function f), while h_g maps output of $f(h(\tilde{x};\theta))$ back to outputs of $g(\tilde{x})$. The parameters θ of the adversarial program are then adjusted to achieve $h_g\left(f\left(h_f\left(\tilde{x}\right)\right)\right) = g\left(\tilde{x}\right)$.

In our work, for simplicity, and to obtain highly interpretable results, we define $\tilde{x}$ to be a small image, g a function that processes small images, x a large image, and f a function that processes large

[*]Work done as a member of the Google AI Residency program (g.co/airesidency).

Preprint. Work in progress.

images. Our function h_f then just consists of drawing x in the center of the large image and θ in the borders, and h_g is simply a hard coded mapping between output class labels. However, the idea is more general; h_f (h_g) could be any consistent transformation that converts between the input (output) formats for the two tasks and causes the model to perform the adversarial task.

We refer to the class of attacks where a machine learning algorithm is repurposed to perform a new task as *adversarial reprogramming*. We refer to θ as an *adversarial program*. In contrast to most previous work in adversarial examples, the magnitude of this perturbation need not be constrained. The attack does not need to be imperceptible to humans, or even subtle, in order to be considered a success. Potential consequences of adversarial reprogramming include theft of computational resources from public facing services, and repurposing of AI-driven assistants into spies or spam bots. Risks stemming from this type of attack are discussed in more detail in Section 5.3.

It may seem unlikely that an additive offset to a neural network's input would be sufficient on its own to repurpose the network to a new task. However, this flexibility stemming only from changes to a network's inputs is consistent with results on the expressive power of deep neural networks. For instance, in [15] it is shown that, depending on network hyperparameters, the number of unique output patterns achievable by moving along a one-dimensional trajectory in input space increases exponentially with network depth. Further, [16] shows that networks can be trained to high accuracy on common tasks even if parameter updates are restricted to occur only in a low dimensional subspace. An additive offset to a neural network's input is equivalent to a modification of its first layer biases (for a convolutional network with biases shared across space, this operation effectively introduces new parameters because the additive input is not subject to the sharing constraint), and therefore an adversarial program corresponds to an update in a low dimensional parameter subspace. Finally, successes in transfer learning have shown that representations in neural networks can generalize to surprisingly disparate tasks. The task of reprograming a trained network may therefore be easier than training a network from scratch — a hypothesis we explore experimentally.

In this paper, we present the first instances of adversarial reprogramming. In Section 2, we discuss related work. In Section 3, we present a training procedure for crafting adversarial programs, which cause a neural network to perform a new task. In Section 4, we experimentally demonstrate adversarial programs that target several convolutional neural networks designed to classify ImageNet data. These adversarial programs alter the network function from ImageNet classification to: counting squares in an image, classifying MNIST digits, and classifying CIFAR-10 images. We additionally examine the susceptibility of trained and untrained networks to adversarial reprogramming. Finally, we end in Sections 5 and 6 by discussing and summarizing our results.

2 Background and Related Work

2.1 Adversarial examples

One definition of adversarial examples is that they are "inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake" [17]. They are often formed by starting with a naturally occuring image and using a gradient-based optimizer to search for a nearby image that causes a mistake [2, 18]. These attacks can be either *untargeted* (the adversary succeeds if they cause any mistake at all) or *targeted* (the adversary succeeds only if they cause the model to recognize the input as belonging to a specific incorrect class). Adversarial attacks have been also proposed for other domains like malware detection [19], generative models [20], network policies for reinforcement learning tasks [21], and network interpretations [22]. In these domains, the attack remains either untargeted (generally degrading the performance) or targeted (producing a specific output). We extend this line of work by developing reprogramming methods that aim to produce specific *functionality* rather than a specific hardcoded output.

Several authors have observed that the same modification can be applied to many different inputs in order to form adversarial examples [8, 23]. For example, Brown et al. [6] designed an "adversarial patch" that can switch the prediction of many models to one specific class (e.g. toaster) when it is placed physically in their field of view. We continue this line of work by finding a single adversarial program that can be presented with many input images to cause the model to process each image according to the adversarial program.

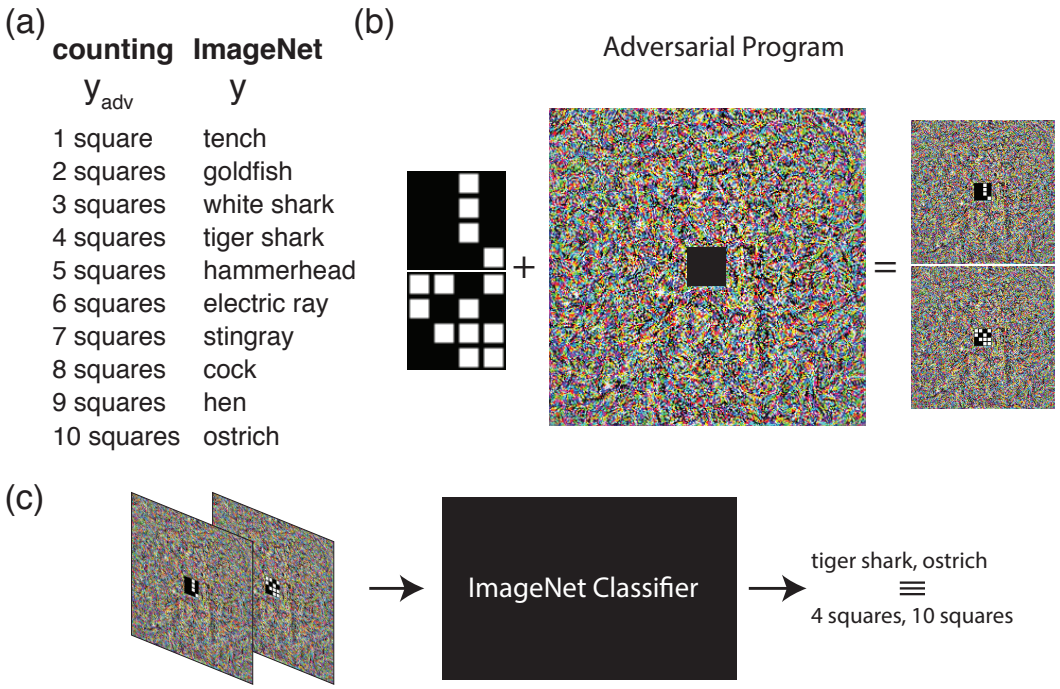

Figure 1: **Illustration of adversarial reprogramming.** (a) Mapping of ImageNet labels to adversarial task labels (squares count in an image). (b) Images from the adversarial task (left) are embedded at the center of an adversarial program (middle), yielding adversarial images (right). The adversarial program shown repurposes an Inception V3 network to count squares in images. (c) Illustration of inference with adversarial images. The network when presented with adversarial images will predict ImageNet labels that map to the adversarial task.

2.2 Transfer Learning

Transfer learning is a well studied topic in machine learning [24, 25]. The goal of transfer learning is to use knowledge obtained from one task to perform another. Neural networks possess properties that can be useful for many tasks [26]. For example, neural networks when trained on images develop features that resemble Gabor filters in early layers even if they are trained with different datasets or different training objectives such as supervised image classification [27], unsupervised density learning [28], or unsupervised learning of sparse representations [29]. Empirical work has demonstrated that it is possible to take a convolutional neural network trained to perform one task, and simply train a linear SVM classifier to make the network work for other tasks [30, 31]. These findings suggest that the task of repurposing neural networks may not require retraining all the weights of neural network. Instead, the adversary task may be simplified to only design a perturbation that effectively realign the output layer of the network for the new task. The main challenge here is whether this task can be accomplished with additive adversarial contributions to neural network inputs?

3 Methods

The attack scenario that we propose here is that an adversary has gained access to the parameters of a neural network that is performing a specific task, and wishes to manipulate the function of the network using an adversarial program that can be added to the network input in order to cause the network to perform a new task. Here we assume that the network was originally designed to perform ImageNet classification, but the methods discussed here can be directly extended to other settings.

Our adversarial program is formulated as an additive contribution to network input. Note that unlike most adversarial perturbations, the adversarial program is not specific to a single image. The same

adversarial program will be applied to all images. We define the adversarial program as:

$$P = \tanh\left(W \odot M\right) \tag{1}$$

where $W \in \mathbb{R}^{n \times n \times 3}$ is the adversarial program parameters to be learned, n is the ImageNet image width, and M is a masking matrix that is 0 for image locations that corresponds to the adversarial data for the new task, otherwise 1. Note that the mask M is not required – we mask out the central region of the adversarial program purely to improve visualization of the action of the adversarial program. Also, note that we use $\tanh\left(\cdot\right)$ to bound the adversarial perturbation to be in $(-1, 1)$ – the same range as the (rescaled) ImageNet images the target networks are trained to classify.

Let, $\tilde{x}_i \in \mathbb{R}^{k \times k \times 3}$ be a sample from the dataset to which we wish to apply the adversarial task, where $k < n$. $\tilde{X}_i \in \mathbb{R}^{n \times n \times 3}$ is the equivalent ImageNet size image with $\tilde{x}_i$ placed in the proper area, defined by the mask M. The corresponding adversarial image is then:

$$X_{adv} = h_f\left(\tilde{X}; W\right) = \tilde{X} + P. \tag{2}$$

Let $P(y|X)$ be the probability that an ImageNet classifier gives to ImageNet label $y \in \{1, \ldots, 1000\}$, given an input image X. We define a hard-coded mapping function $h_g(y_{adv})$ that maps a label from an adversarial task y_{adv} to a set of ImageNet labels. For example, if an adversarial task has 10 different classes ($y_{adv} \in \{1, \ldots, 10\}$), $h_g\left(\cdot\right)$ may be defined to assign the first 10 classes of ImageNet, any other 10 classes, or multiple ImageNet classes to the adversarial labels. Our adversarial goal is thus to maximize the probability $P(h_g(y_{adv})|X_{adv})$. We set up our optimization problem as

$$\hat{W} = \underset{W}{\operatorname{argmin}}\left(-\log P(h_g(y_{adv})|X_{adv}) + \lambda||W||_2^2\right), \tag{3}$$

where λ is the coefficient for a weight norm penalty, to reduce overfitting. We optimize this loss with Adam while exponentially decaying the learning rate. Hyperparameters are given in Appendix A. Note that after the optimization the adversarial program has a minimal computation cost from the adversary's side as it only requires computing X_{adv} (Equation 2), and mapping the resulting ImageNet label to the correct class. In other words, during inference the adversary needs only store the program and add it to the data, thus leaving the majority of computation to the target network.

One interesting property of adversarial reprogramming is that it must exploit nonlinear behavior of the target model. This is in contrast to traditional adversarial examples, where attack algorithms based on linear approximations of deep neural nets are sufficient to cause high error rate [8]. Consider a linear model that receives an input $\tilde{x}$ and a program θ concatenated into a single vector: $x = [\tilde{x}, \theta]^T$. Suppose that the weights of the linear model are partitioned into two sets, $v = [v_{\tilde{x}}, v_\theta]^T$. The output of the model is $v^T x = v_{\tilde{x}}^T \tilde{x} + v_\theta^T \theta$. The adversarial program θ adapts the effective biases $v_\theta^T \theta$ but cannot adapt the weights applied to the input $\tilde{x}$. The adversarial program θ can thus bias the model toward consistently outputting one class or the other but cannot change the way the input is processed. For adversarial reprogramming to work, the model must contain a term that involves nonlinear interactions of $\tilde{x}$ and θ. A deep neural net with nonlinear activation functions satisfies this requirement.

4 Results

To demonstrate the feasibility of adversarial reprogramming, we crafted adversarial programs targeted at six ImageNet models. In each case, we reprogrammed the network to perform three different adversarial tasks: counting squares, MNIST classification, and CIFAR-10 classification. The weights of all trained models were obtained from [32], and top-1 ImageNet precisions are shown in Table Supp. 1. We additionally examined whether adversarial training conferred resistance to adversarial reprogramming, and compared the susceptibility of trained networks to random networks.

4.1 Counting squares

To illustrate the adversarial reprogramming procedure, we start with a simple adversarial task. That is counting the number of squares in an image. We generated images of size $36 \times 36 \times 3$ that include 9×9 white squares with black frames. Each square could appear in 16 different position in the image, and the number of squares ranged from 1 to 10. The squares were placed randomly on

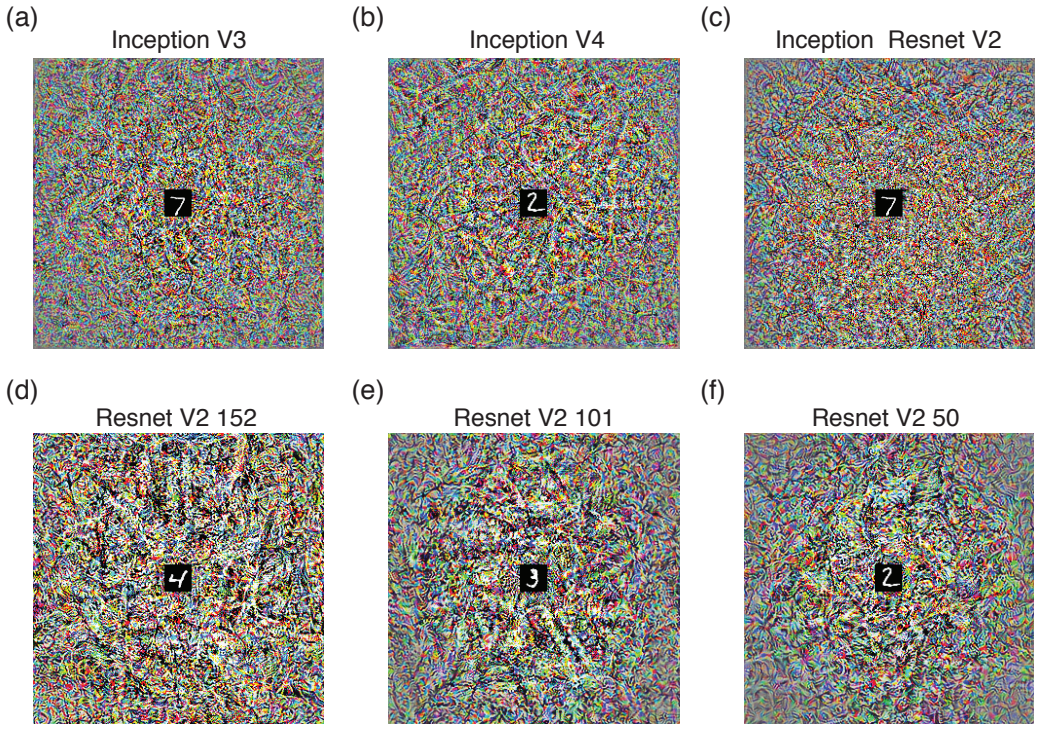

Figure 2: **Examples of adversarial programs for MNIST classification.** (a-f) Adversarial programs which cause six ImageNet models to instead function as MNIST classifiers. Each program is shown being applied to one MNIST digit.

gridpoints (Figure 1b left). We embedded these images in an adversarial program (Figure 1b middle). The resulting images are of size $299 \times 299 \times 3$ with the $36 \times 36 \times 3$ images of the squares at the center (Figure 1b right). Thus, the adversarial program is simply a frame around the counting task images. We trained one adversarial program per ImageNet model, such that the first 10 ImageNet labels represent the number of squares in each image (Figure 1c). Note that the labels we used from ImageNet have no relation to the labels of the new adversarial task. For example, a 'White Shark' has nothing to do with counting 3 squares in an image, and an 'Ostrich' does not at all resemble 10 squares. We then evaluated the accuracy in the task by sampling 100,000 images and comparing the network prediction to the number of squares in the image.

Despite the dissimilarity of ImageNet labels and adversarial labels, and that the adversarial program is equivalent simply to a first layer bias, the adversarial program masters this counting task for all networks (Table 1). These results demonstrate the vulnerability of neural networks to reprogramming on this simple task using only additive contributions to the input.

4.2 MNIST classification

In this section, we demonstrate adversarial reprogramming on somewhat more complex task of classifying MNIST digits. We measure *test* and train accuracy, so it is impossible for the adversarial program to have simply memorized all training examples. Similar to the counting task, we embedded MNIST digits of size $28 \times 28 \times 3$ inside a frame representing the adversarial program, we assigned the first 10 ImageNet labels to the MNIST digits, and trained an adversarial program for each ImageNet model. Figure 2 shows examples of the adversarial program for each network being applied.

Our results show that ImageNet networks can be successfully reprogramed to function as an MNIST classifier by presenting an additive adversarial program. The adversarial program additionally generalized well from the training to test set, suggesting that the reprogramming does not function purely by memorizing train examples, and is not brittle to small changes in the input. One interesting observation is that the adversarial programs targeted at Inception architectures are qualitatively

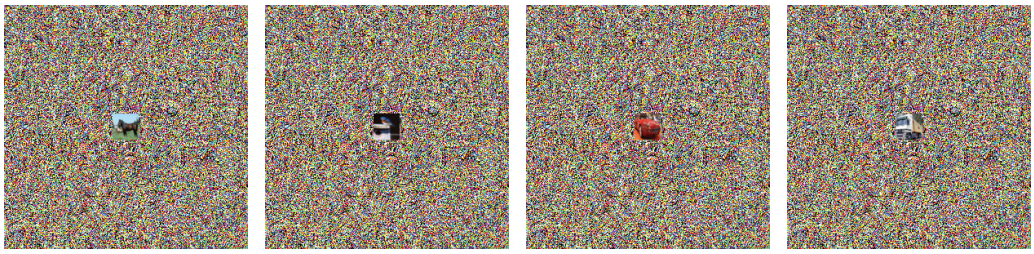

Figure 3: **Examples of adversarial images for CIFAR-10 classification.** An adversarial program repurposing an Inception V3 model to instead function as an CIFAR-10 classifier is shown being applied to four CIFAR-10 images.

Table 1: **Trained ImageNet classifiers can be adversarially reprogrammed to perform a variety of tasks.** Table gives accuracy of reprogrammed networks on a counting task, MNIST classification task, and CIFAR-10 classification task.

		MNIST		CIFAR-10	
ImageNet Model	Counting	train set	test set	train set	test set
Inception V3	0.9993	0.9781	0.9753	0.7311	0.6911
Inception V4	0.9999	0.9638	0.9646	0.6948	0.6683
Inception Resnet V2	0.9994	0.9773	0.9744	0.6985	0.6719
Resnet V2 152	0.9763	0.9478	0.9534	0.6410	0.6210
Resnet V2 101	0.9843	0.9650	0.9664	0.6435	0.6301
Resnet V2 50	0.9966	0.9506	0.9496	0.6	0.5858
Inception V3 adv.		0.9761	0.9752		

different from those targeted at Resnet architectures (Figure 2). This suggests that the method of action of the adversarial program is in some sense architecture-specific.

4.3 CIFAR-10 classification

Here we implement a more challenging adversarial task. That is, crafting adversarial programs to repurpose ImageNet models to instead classify CIFAR-10 images. Some examples of the resulting adversarial images are given in Figure 3. Our results show that our adversarial program was able to increase the accuracy on CIFAR-10 from chance to a moderate accuracy (Table 1). This accuracy is near what is expected from typical fully connected networks [33] but with minimal computation cost from the adversary side at inference time. One observation is that although adversarial programs trained to classify CIFAR-10 are different from those that classify MNIST or perform counting task, the programs show some visual similarities, e.g. ResNet architecture adversarial programs seem to possess some low spatial frequency texture (Figure 4a).

4.4 Reprogramming untrained and adversarially trained networks

One important question is the degree to which susceptibility to adversarial reprogramming depends on the details of the model being attacked. To test this, we first examined attack success on an Inception V3 model that was trained on ImageNet data using adversarial training [11]. Adversarial training augments each minibatch with adversarial examples during training, and is one of the most common methods for guarding against adversarial examples. As in Section 4.2, we adversarially reprogrammed this network to classify MNIST digits. Our results (Table 1) indicate that the model trained with adversarial training is still vulnerable to reprogramming, with only a slight reduction in attack success. This shows that a standard approach to adversarial defense has little efficacy against adversarial reprogramming. This finding is likely explained by the differences between adversarial reprogramming and standard adversarial attacks. First, that the goal is to repurpose the network rather than cause it to make a specific mistake, and second that the magnitude of adversarial programs can be large, while traditional adversarial attacks are of a small perturbation magnitude.

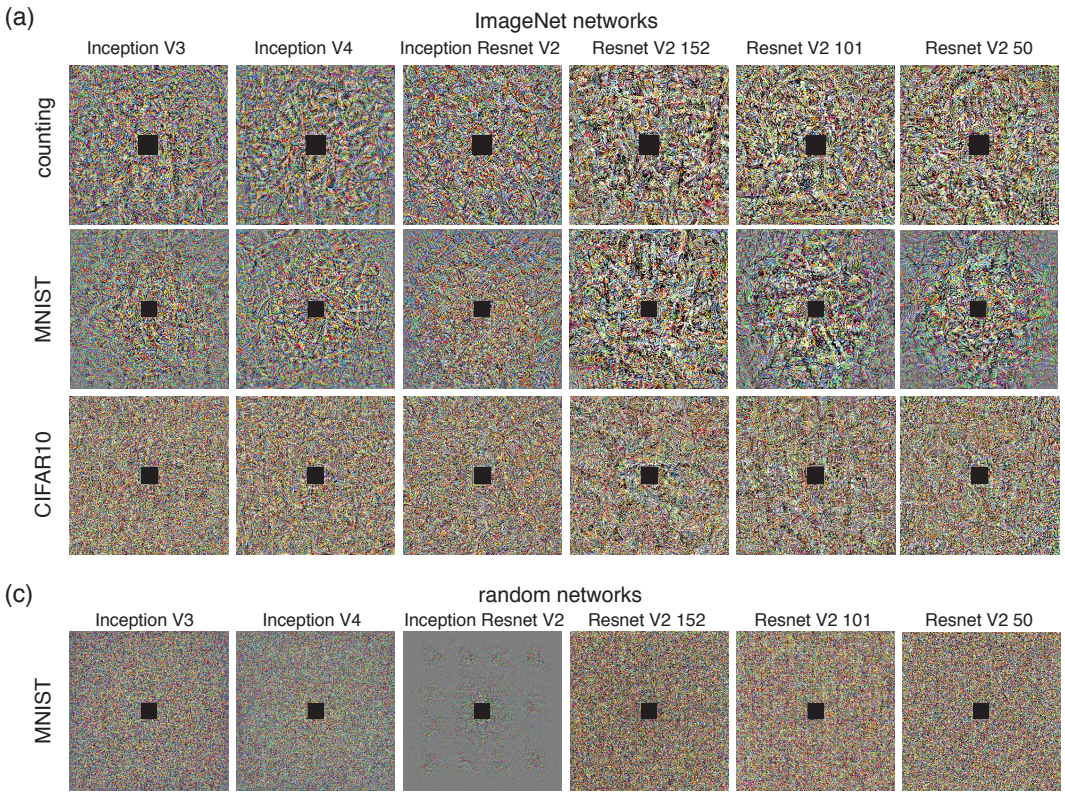

Figure 4: **Adversarial programs exhibit qualitative similarities and differences across both network and task.** (a) Top: adversarial programs targeted to repurpose networks pre-trained on ImageNet to count squares in images. Middle: adversarial programs targeted to repurpose networks pre-trained on ImageNet to function as MNIST classifiers. Bottom: adversarial programs to cause the same networks to function as CIFAR-10 classifiers. (b) Adversarial programs targeted to repurpose networks with randomly initialized parameters to function as MNIST classifiers.

To further explore dependence on the details of the model, we performed adversarial reprogramming attacks on models with random weights. We used the same models and MNIST target task as in Section 4.2 – we simply used the ImageNet models with randomly initialized rather than trained weights. MNIST classification task was easy for networks pretrained on ImageNet (Table 1). However, for random networks, training was very challenging and generally converged to a much lower accuracy (only one model could train to a similar accuracy as trained ImageNet models; see Table 2). Moreover, the appearance of the adversarial programs was qualitatively distinct from the adversarial programs obtained with networks pretrained on ImageNet (see Figure 4b).

This finding suggests that the original task the neural networks perform is important for adversarial reprogramming. This result may seem surprising, as random networks have rich structure adversarial programs might be expected to take advantage of. For example, theoretical results have shown that wide neural networks become identical to Gaussian processes, where training specific weights in intermediate layers is not necessary to perform tasks [34, 35]. Other work has demonstrated that it is possible to use random networks as generative models for images [36, 37], further supporting their potential richness. On the other hand, ideas from transfer learning suggest that networks generalize best to tasks with similar structure. Our experimental results suggest that the structure in our three adversarial tasks is similar enough to that in ImageNet that the adversarial program can benefit from training of the target model on ImageNet. They also suggest that it is possible for changes to the input of the network to take advantage of that similarity, rather than changes to the output layer as is more typical in transfer learning. However, another plausible hypothesis is that randomly initialized networks perform poorly for simpler reasons, such as poor scaling of network weights at initialization.

Table 2: **Adversarial reprogramming is less effective when it targets untrained networks.** Table gives accuracy of reprogrammed networks on an MNIST classification task. Target networks have been randomly initialized, and have not been trained.

	MNIST	
Random Model	train set	test set
Inception V3	0.4530	0.4539
Inception V4	0.1876	0.1861
Inception Resnet V2	0.1125	0.1135
Resnet V2 152	0.0986	0.1032
Resnet V2 101	0.1688	0.1756
Resnet V2 50	0.9342	0.9325

5 Discussion

5.1 Flexibility of trained neural networks

We found that trained neural networks were more susceptible to adversarial reprogramming than random networks. This suggests that the adversarial program is repurposing learned features which already exist in the network for a new task. This can be seen as a novel form of transfer learning, where the inputs to the network (equivalent to first layer biases) are modified, rather than the readout weights as is more typical. Our results suggest that dynamical reuse of neural circuits should be practical in modern artificial neural networks. This holds the promise of enabling machine learning systems which are easier to repurpose, more flexible, and more efficient due to shared compute. Indeed, recent work in machine learning has focused on building large dynamically connected networks with reusable components [38].

It is unclear whether the reduced performance when targeting random networks, and when reprogramming to perform CIFAR-10 classification, was due to limitations in the expressivity of the adversarial perturbation, or due to the optimization task in Equation 3 being more difficult in these situations. Disentangling limitations in expressivity and trainability will be an interesting direction for future work.

5.2 Beyond the image domain

We only demonstrated adversarial reprogramming on tasks in the image domain. It is an interesting area for future research whether similar attacks might succeed for audio, video, text, or other domains. Adversarial reprogramming of recurrent neural networks (RNNs) would be particularly interesting, since RNNs (especially those with attention or memory mechanisms) can be Turing complete [39]. An attacker would therefore only need to find inputs which induced the RNN to perform a small number of simple operations, such as increment counter, decrement counter, and change input attention location if counter is zero [40]. If adversarial programs can be found for these simple operations, then they could be composed to reprogram the RNN to perform a very large array of tasks.

5.3 Potential goals of an adversarial reprogramming attack

A variety of nefarious ends may be achievable if machine learning systems can be reprogrammed by a specially crafted input. The most direct of these is the simple theft of computational resources. For instance, an attacker might develop an adversarial program which causes the computer vision classifier in a cloud hosted photos service to solve image captchas and enable creation of spam accounts. If RNNs can be flexibly reprogrammed as described in Section 5.2, this computational theft might extend to more arbitrary tasks, such as mining cryptocurrency. A major danger beyond the computational theft is that an adversary may repurpose computational resources to perform a task which violates the code of ethics of the system provider.

Adversarial programs could also be used as a novel way to achieve more traditional computer hacks. For instance, as phones increasingly act as AI-driven digital assistants, the plausibility of reprogramming someone's phone by exposing it to an adversarial image or audio file increases. As

these digital assistants have access to a user's email, calendar, social media accounts, and credit cards the consequences of this type of attack also grow larger.

6 Conclusion

In this work, we proposed a new class of adversarial attacks that aim to reprogram neural networks to perform novel adversarial tasks. Our results demonstrates for the first time the possibility of such attacks. These results demonstrate both surprising flexibility and surprising vulnerability in deep neural networks. Future investigation should address the properties and limitations of adversarial programming and possible ways to defend against it.

Acknowledgments

We are grateful to Jaehoon Lee, Sara Hooker, Simon Kornblith, Supasorn Suwajanakorn for useful comments on the manuscript. We thank Alexey Kurakin for help reviewing the code. We thank Justin Gilmer and Luke Metz for discussion surrounding the original idea.

References

[1] Ivan Evtimov et al. "Robust Physical-World Attacks on Deep Learning Models". In: *arXiv preprint arXiv:1707.08945* 1 (2017).

[2] Christian Szegedy et al. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).

[3] Nicolas Papernot et al. "The Limitations of Deep Learning in Adversarial Settings." In: *CoRR* abs/1511.07528 (2015).

[4] Nicolas Papernot et al. "Practical black-box attacks against machine learning". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM. 2017, pp. 506–519.

[5] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples". In: *arXiv preprint arXiv:1605.07277* (2016).

[6] Tom B Brown et al. "Adversarial patch". In: *arXiv preprint arXiv:1712.09665* (2017).

[7] Yanpei Liu et al. "Delving into transferable adversarial examples and black-box attacks". In: *arXiv preprint arXiv:1611.02770* (2016).

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[9] A. Kurakin, I. Goodfellow, and S. Bengio. "Adversarial Machine Learning at Scale". In: *ArXiv e-prints* (Nov. 2016). arXiv: `1611.01236 [cs.CV]`.

[10] Aleksander Madry et al. "Towards deep learning models resistant to adversarial attacks". In: *arXiv preprint arXiv:1706.06083* (2017).

[11] F. Tramèr et al. "Ensemble Adversarial Training: Attacks and Defenses". In: *ArXiv e-prints* (May 2017). arXiv: `1705.07204 [stat.ML]`.

[12] J Zico Kolter and Eric Wong. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: *arXiv preprint arXiv:1711.00851* (2017).

[13] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. "Adversarial Logit Pairing". In: *arXiv preprint arXiv:1803.06373* (2018).

[14] Yen-Chen Lin et al. "Tactics of adversarial attack on deep reinforcement learning agents". In: *arXiv preprint arXiv:1703.06748* (2017).

[15] Maithra Raghu et al. "On the expressive power of deep neural networks". In: *arXiv preprint arXiv:1606.05336* (2016).

[16] Chunyuan Li et al. "Measuring the Intrinsic Dimension of Objective Landscapes". In: *arXiv preprint arXiv:1804.08838* (2018).

[17] Ian Goodfellow et al. *Attacking Machine Learning with Adversarial Examples*. 2017. URL: `https://blog.openai.com/adversarial-example-research/`.

[18] Battista Biggio et al. “Evasion Attacks against Machine Learning at Test Time”. In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*. 2013, pp. 387–402. DOI: 10.1007/978-3-642-40994-3_25.

[19] Kathrin Grosse et al. “Adversarial Examples for Malware Detection”. In: *ESORICS 2017*. 2017, pp. 62–79. DOI: 10.1007/978-3-319-66399-9_4. URL: https://doi.org/10.1007/978-3-319-66399-9_4.

[20] Jernej Kos, Ian Fischer, and Dawn Song. “Adversarial examples for generative models”. In: *arXiv preprint arXiv:1702.06832* (2017).

[21] Sandy Huang et al. “Adversarial attacks on neural network policies”. In: *arXiv preprint arXiv:1702.02284* (2017).

[22] Amirata Ghorbani, Abubakar Abid, and James Zou. “Interpretation of Neural Networks is Fragile”. In: *arXiv preprint arXiv:1710.10547* (2017).

[23] Seyed-Mohsen Moosavi-Dezfooli et al. “Universal Adversarial Perturbations”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 86–94.

[24] Rajat Raina et al. “Self-taught learning: transfer learning from unlabeled data”. In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 759–766.

[25] Grégoire Mesnil et al. “Unsupervised and transfer learning challenge: a deep learning approach”. In: *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning workshop-Volume 27*. JMLR. org. 2011, pp. 97–111.

[26] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[28] Honglak Lee et al. “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations”. In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 609–616.

[29] Quoc V Le et al. “ICA with reconstruction cost for efficient overcomplete feature learning”. In: *Advances in neural information processing systems*. 2011, pp. 1017–1025.

[30] Ali Sharif Razavian et al. “CNN features off-the-shelf: an astounding baseline for recognition”. In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE. 2014, pp. 512–519.

[31] Jeff Donahue et al. “Decaf: A deep convolutional activation feature for generic visual recognition”. In: *International conference on machine learning*. 2014, pp. 647–655.

[32] *TensorFlow-Slim image classification model library*. https://github.com/tensorflow/models/tree/master/research/slim. Accessed: 2018-05-01.

[33] Zhouhan Lin, Roland Memisevic, and Kishore Konda. “How far can we go without convolution: Improving fully-connected networks”. In: *arXiv preprint arXiv:1511.02580* (2015).

[34] Alexander G de G Matthews et al. “Gaussian Process Behaviour in Wide Deep Neural Networks”. In: *arXiv preprint arXiv:1804.11271* (2018).

[35] Jaehoon Lee et al. “Deep Neural Networks as Gaussian Processes”. In: *arXiv preprint arXiv:1711.00165* (2017).

[36] Ivan Ustyuzhaninov et al. “Texture synthesis using shallow convolutional networks with random filters”. In: *arXiv preprint arXiv:1606.00021* (2016).

[37] Kun He, Yan Wang, and John Hopcroft. “A powerful generative model using random weights for the deep image representation”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 631–639.

[38] Noam Shazeer et al. “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer”. In: *arXiv preprint arXiv:1701.06538* (2017).

[39] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. “Neural programmer: Inducing latent programs with gradient descent”. In: *arXiv preprint arXiv:1511.04834* (2015).

[40] Marvin L Minsky. “Recursive unsolvability of Post’s problem of" tag" and other topics in theory of Turing machines”. In: *Annals of Mathematics* (1961), pp. 437–455.

Supplemental material

A Supplementary Tables

Table Supp. 1: Top-1 precision of models on ImageNet data

Model	Accuracy
Inception V3	0.78
Inception V4	0.802
Inception Resnet V2	0.804
Resnet V2 152	0.778
Resnet V2 101	0.77
Resnet V2 50	0.756
Inception V3 adv.	0.776

Table Supp. 2: Hyper-parameters for adversarial program training for the square counting adversarial task. For all models, we used the Adam optimizer with its default parameters while decaying the learning rate exponentially during training. We distributed training data across a number of GPUs (each GPU receive 'batch' data samples). We then performed synchronized updates of the adversarial program parameters.

ImageNet Model	λ	batch	GPUS	learn rate	decay	epochs/decay	steps
Inception V3	0.01	50	4	0.05	0.96	2	100000
Inception V4	0.01	50	4	0.05	0.96	2	100000
Inception Resnet V2	0.01	50	4	0.05	0.96	2	100000
Resnet V2 152	0.01	20	4	0.05	0.96	2	100000
Resnet V2 101	0.01	20	4	0.05	0.96	2	60000
Resnet V2 50	0.01	20	4	0.05	0.96	2	100000

Table Supp. 3: Hyper-parameters for adversarial program training for MNIST classification adversarial task. For all models, we used the Adam optimizer with its default parameters while decaying the learning rate exponentially during training. We distributed training data across a number of GPUs (each GPU receive 'batch' data samples). We then performed synchronized updates of the adversarial program parameters. (The Model Inception V3 adv is pretrained on ImageNet data using adversarial training method.

ImageNet Model	λ	batch	GPUS	learn rate	decay	epochs/decay	steps
Inception V3	0.05	100	4	0.05	0.96	2	60000
Inception V4	0.05	100	4	0.05	0.96	2	60000
Inception Resnet V2	0.05	50	8	0.05	0.96	2	60000
Resnet V2 152	0.05	50	8	0.05	0.96	2	60000
Resnet V2 101	0.05	50	8	0.05	0.96	2	60000
Resnet V2 50	0.05	100	4	0.05	0.96	2	60000
Inception V3 adv.	0.01	50	6	0.05	0.98	4	100000

Table Supp. 4: Hyper-parameters for adversarial program training for CIFAR-10 classification adversarial task. For all models, we used ADAM optimizer with its default parameters while decaying the learning rate exponentially during training. We distributed training data on number of GPUS (each GPU receive 'batch' data samples). We then performed synchronized updates of the adversarial program parameters.

ImageNet Model	λ	batch	GPUS	learn rate	decay	epochs/decay	steps
Inception V3	0.01	50	6	0.05	0.99	4	300000
Inception V4	0.01	50	6	0.05	0.99	4	300000
Inception Resnet V2	0.01	50	6	0.05	0.99	4	300000
Resnet V2 152	0.01	30	6	0.05	0.99	4	300000
Resnet V2 101	0.01	30	6	0.05	0.99	4	300000
Resnet V2 50	0.01	30	6	0.05	0.99	4	300000

Table Supp. 5: Hyper-parameters for adversarial program training for MNIST classification adversarial task. For all models, we used the Adam optimizer with its default parameters while decaying the learning rate exponentially during training. We distributed training data across a number of GPUs (each GPU receive 'batch' data samples). We then performed synchronized updates of the adversarial program parameters.

Random Model	λ	batch	GPUS	learn rate	decay	epochs/decay	steps
Inception V3	0.01	50	4	0.05	0.96	2	100000
Inception V4	0.01	50	4	0.05	0.96	2	100000
Inception Resnet V2	0.01	50	4	0.05	0.96	2	60000
Resnet V2 152	0.01	20	4	0.05	0.96	2	60000
Resnet V2 101	0.01	20	4	0.05	0.96	2	60000
Resnet V2 50	0.01	50	4	0.05	0.96	2	60000