Adversarial Turing Patterns from Cellular Automata

Nurislam Tursynbek[1], Ilya Vilkoviskiy[1], Maria Sindeeva[1], Ivan Oseledets[1]

[1]Skolkovo Institute of Science and Technology

nurislam.tursynbek@gmail.com, reminguk@gmail.com, maria.sindeeva@skoltech.ru, i.oseledets@skoltech.ru

Abstract

State-of-the-art deep classifiers are intriguingly vulnerable to universal adversarial perturbations: single disturbances of small magnitude that lead to misclassification of most inputs. This phenomena may potentially result in a serious security problem. Despite the extensive research in this area, there is a lack of theoretical understanding of the structure of these perturbations. In image domain, there is a certain visual similarity between patterns, that represent these perturbations, and classical Turing patterns, which appear as a solution of non-linear partial differential equations and are underlying concept of many processes in nature. In this paper, we provide a theoretical bridge between these two different theories, by mapping a simplified algorithm for crafting universal perturbations to (inhomogeneous) cellular automata, the latter is known to generate Turing patterns. Furthermore, we propose to use Turing patterns, generated by cellular automata, as universal perturbations, and experimentally show that they significantly degrade the performance of deep learning models. We found this method to be a fast and efficient way to create a data-agnostic quasi-imperceptible perturbation in the black-box scenario. The source code is available at https://github.com/NurislamT/advTuring.

Introduction

Deep neural networks have shown success in solving complex problems for different applications ranging from medical diagnoses to self-driving cars, but recent findings surprisingly show they are not safe and vulnerable to well-designed negligibly perturbed inputs (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2015), called adversarial examples, compromising people's confidence in them. Moreover, most of modern defenses to adversarial examples are found to be easily circumvented (Athalye, Carlini, and Wagner 2018). One reason why adversarial examples are hard to defend against is the difficulty of constructing a theory of the crafting process of them.

Intriguingly, adversarial perturbations can be transferable across inputs. Universal Adversarial Perturbations (UAPs), single disturbances of small magnitude that lead to misclassification of most inputs, were presented in image domain by Moosavi-Dezfooli et. al (Moosavi-Dezfooli et al. 2017),

where authors proposed iterative strategy of gradually pushing a data point to the decision boundary. However, to construct a successful perturbation thousands of images were needed, whereas Khrulkov et. al (Khrulkov and Oseledets 2018) proposed an efficient algorithm of constructing UAPs with a very small number of samples. The proposed universal perturbations construct complex and interesting unusual patterns. Studying how these patterns emerge will allow better understanding the nature of adversarial examples.

We start from an interesting observation that patterns generated in (Khrulkov and Oseledets 2018) visually look very similarly to the so-called *Turing patterns* (Figure 1) which were introduced by Alan Turing in the seminal paper *"The Chemical Basis of Morphogenesis"* (Turing 1952). It describes the way in which patterns in nature such as stripes and spots can arise naturally out of a homogeneous uniform state. The original theory of Turing patterns, a two-component reaction-diffusion system, is an important model in mathematical biology and chemistry. Turing found that a stable state in the system with local interactions can become unstable in the presence of diffusion. Reaction–diffusion systems have gained significant attention and was used as a prototype model for pattern formation.

In this paper, we provide an explanation why UAPs from (Khrulkov and Oseledets 2018) bear similarity to the Turing patterns using the formalism of *cellular automata* (CA): the iterative process for generating UAPs can be approximated by such process, and Turing patterns can be easily generated by cellular automata (Young 1984). Besides, this gives a very simple way to generate new examples by learning the parameters of such automata by black-box optimization. We also experimentally show this formalism can produce examples very close to so-called single Fourier attacks by studying Fourier harmonics of the obtained examples.

The main contributions of the paper are following:

- We show that the iterative process to generate Universal Adversarial Perturbations from (Khrulkov and Oseledets 2018) can be reformulated as a cellular automata that generates Turing patterns.

- We experimentally show Turing patterns can be used to generate UAPs in a black-box scenario with high fooling rates for different networks.
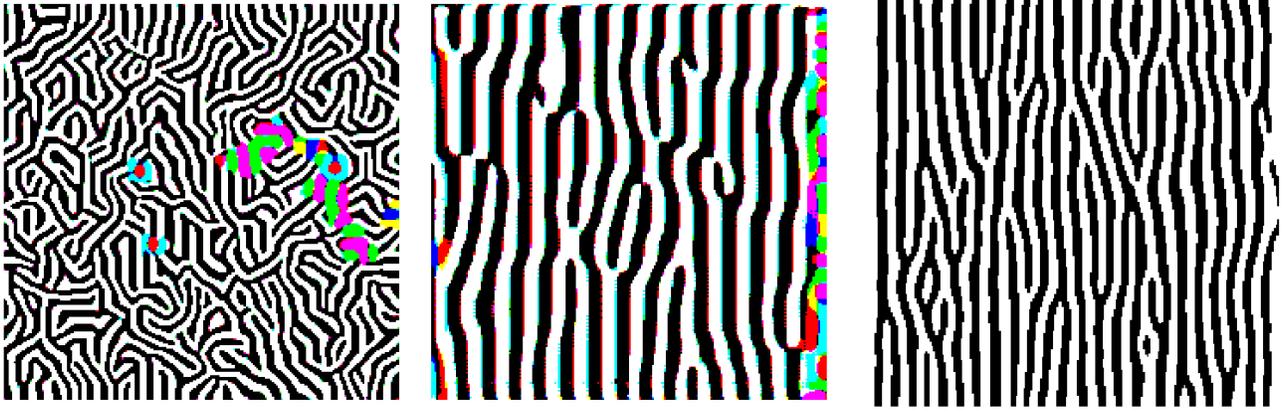
(a) Example of UAPs constructed by (Khrulkov and Oseledets 2018)

(b) Turing patterns

Figure 1: Visual similarity of Universal Adversarial Perturbations by (Khrulkov and Oseledets 2018) and Turing patterns

Background

Universal Adversarial Perturbations

Adversarial perturbations are small disturbances added to the inputs that cause machine learning models to make a mistake. In (Szegedy et al. 2013) authors discovered these noises by solving the optimization problem:

$$\min_{\varepsilon} \|\varepsilon\|_2 \quad s.t. \ \mathcal{C}(\mathbf{x} + \varepsilon) \neq \mathcal{C}(\mathbf{x}), \quad (1)$$

where $\mathbf{x}$ is an input object and $\mathcal{C}(\cdot)$ is a neural network classifier. It was shown that the solution to the minimization model (1) leads to perturbations imperceptible to human eye.

Universal adversarial perturbation (Moosavi-Dezfooli et al. 2017) is a small ($\|\varepsilon\|_p \leq L$) noise that makes classifier to misclassify the fraction $(1 - \delta)$ of inputs from the given dataset μ. The goal is to make δ as small as possible and find a perturbation ε such that:

$$\mathbb{P}_{\mathbf{x} \sim \mu} \left[\mathcal{C}(\mathbf{x} + \varepsilon) \neq \mathcal{C}(\mathbf{x})\right] \geq 1 - \delta \quad s.t. \ \|\varepsilon\|_p \leq L, \quad (2)$$

In (Khrulkov and Oseledets 2018) an efficient way of computing such UAPs was proposed, achieving relatively high fooling rates using only small number of inputs. Consider an input $\mathbf{x} \in \mathbb{R}^d$, its i-th layer output $\mathbf{f}_i(\mathbf{x}) \in \mathbb{R}^{d_i}$ and Jacobian matrix $\mathbf{J}_i(x) = \left.\frac{\partial \mathbf{f}_i(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}} \in \mathbb{R}^{d_i \times d}$. For a small input perturbation $\varepsilon \in \mathbb{R}^d$, using first-order Taylor expansion $\mathbf{f}_i(\mathbf{x} + \varepsilon) \approx \mathbf{f}_i(\mathbf{x}) + \mathbf{J}_i(\mathbf{x})\varepsilon$, authors find that to construct a UAP, it is sufficient to maximize the sum of norms of Jacobian matrix product with perturbation for a small batch of inputs X_b, constrained with perturbation norm ($\|\varepsilon\|_p = L$ is obtained by multiplying the solution by L):

$$\sum_{\mathbf{x}_j \in X_b} \|\mathbf{J}_i(\mathbf{x}_j)\varepsilon\|_q^q \to \max, \qquad s.t. \ \|\varepsilon\|_p = 1. \quad (3)$$

To solve the optimization problem (3) the Boyd iteration (Boyd 1974) (generalization of the power method to the problem of computing generalized singular vectors) is found

to provide fast convergence:

$$\varepsilon_{t+1} = \frac{\psi_{p'}(\mathbf{J}_i^T(X_b)\psi_q(\mathbf{J}_i(X_b)\varepsilon_t))}{\|\psi_{p'}(\mathbf{J}_i^T(X_b)\psi_q(\mathbf{J}_i(X_b)\varepsilon_t))\|_p}, \quad (4)$$

where $\frac{1}{p'} + \frac{1}{p} = 1$ and $\psi_r(z) = \text{sign}(z)|z|^{r-1}$, and $\mathbf{J}_i(X_b) \in \mathbb{R}^{bd_i \times d}$ for a batch X_b with batch size b is given as a block matrix:

$$\mathbf{J}_i(X_b) = \begin{bmatrix} \mathbf{J}_i(\mathbf{x}_1) \\ \vdots \\ \mathbf{J}_i(\mathbf{x}_b) \end{bmatrix}. \quad (5)$$

For the case of $p = \infty$ (4) takes the form:

$$\varepsilon_{t+1} = \text{sign}(\mathbf{J}_i^T(X_b)\psi_q(\mathbf{J}_i(X_b)\varepsilon_t)). \quad (6)$$

Equation (6) is the first crucial point in our study, and we will show, how they connect to Turing patterns. We now proceed with describing the background behind these patterns and mathematical correspondence between Equation (6) and emergence of Turing patterns as cellular automata is described in next Section.

Turing Patterns as Cellular Automata

In his seminal work (Turing 1952) Alan Turing studied the emergence theory of patterns in nature such as stripes and spots that can arise out of a homogeneous uniform state. The proposed model was based on the solution of reaction-diffusion equations of two chemical morphogens (reagents):

$$\frac{\partial n_1(i,j)}{\partial t} = -\mu_1 \nabla^2 n_1(i,j) + a(n_1(i,j), n_2(i,j)),$$
$$\frac{\partial n_2(i,j)}{\partial t} = -\mu_2 \nabla^2 n_2(i,j) + b(n_1(i,j), n_2(i,j)). \quad (7)$$

Here, $n_1(i,j)$ and $n_2(i,j)$ are concentrations of two morphogens in the point with coordinates (i,j) μ_1 and μ_2 are scalar coefficients. a and b are nonlinear functions, with at least two points (i,j), satisfying (i,j) :
$$\begin{cases} a(n_1(i,j), n_2(i,j)) = 0 \\ b(n_1(i,j), n_2(i,j)) = 0. \end{cases}$$

Turing noted the solution presents alternating patterns with specific size that does not depend on the coordinate and describes the stationary solution, which interpolates between zeros of a and b.

Young et. al (Young 1984) proposed to generate Turing patterns by a discrete cellular automata as following. Let us consider a 2D grid of cells. Each cell (i, j) is equipped with a number $n(i, j) \in \{0, 1\}$. The sum of cells, neighbouring with the current cell (i, j) within the radius r_i is multiplied by w, while the sum of values of cells, neighbouring with the current cell (i, j) between radii r_{in} and r_{out}, is multiplied by -1. If the total sum of these two terms is positive, the new value of the cell is 1, otherwise 0. This process can be written by introducing the convolutional kernel $Y(m, l)$:

$$Y(m, l) = \begin{cases} w & \text{if } |m|^2 + |l|^2 < r_{in}^2, \\ -1 & \text{if } r_{out}^2 > |m|^2 + |l|^2 > r_{in}^2. \end{cases} \quad (8)$$

The coefficient w is found from the condition that the sum of elements of kernel Y is set to be 0:

$$\sum_m \sum_l Y(m, l) = 0. \quad (9)$$

The update rule is written as:

$$n_{k+1}(i, j) = \frac{1}{2} \left[\text{sign} \left(Y \star n_k(i, j) \right) + 1 \right], \quad (10)$$

where $\star$ denotes a usual 2D convolution operator:

$$Y \star n(i, j) = \sum_m \sum_l Y(m, l) n(i + m, j + l). \quad (11)$$

For convenience, instead of $\{0, 1\}$, we can use ± 1 and rewrite (10) (considering condition (9)) as:

$$n_{k+1}(i, j) = \text{sign} \left(Y \star n_k(i, j) \right). \quad (12)$$

Note that (12) is similar to (6), and lets investigate this connection in more details.

Connection between Boyd iteration and Cellular Automata

We restrict our study to the case $q = 2$. Then $\psi_2(z) = z$ and Equation (6) can be rewritten as:

$$\varepsilon_{t+1} = \text{sign}(\mathbf{J}_i^T(X_b) \mathbf{J}_i(X_b) \varepsilon_t). \quad (13)$$

In (Khrulkov and Oseledets 2018) it was shown that perturbations of the first layers give the most promising results. Initial layers of most of the state-of-the-art image classification neural networks are convolutional followed by nonlinear activation functions σ (e.g. $\text{ReLU}(z) = \max(0, z)$). If the network does not include other operations (such as pooling or skip connections) the output of the i-th layer is:

$$\mathbf{f}_i(\mathbf{x}) = \sigma(\mathbf{M}_i \mathbf{f}_{i-1}(\mathbf{x})), \quad (14)$$

where $\mathbf{M}_i \in \mathbb{R}^{d_i \times d_{i-1}}$ is a matrix of the 2D convolution operator of the i-th layer, corresponding to the convolutional kernel $\mathbf{K}_i$. Using chain rule, the Jacobian in (6) is:

$$\mathbf{J}_i(\mathbf{x}) = \frac{\partial \mathbf{f}_i(\mathbf{x})}{\partial \mathbf{f}_{i-1}(\mathbf{x})} \cdots \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{D}_i(\mathbf{x}) \mathbf{M}_i \cdots \mathbf{D}_1(\mathbf{x}) \mathbf{M}_1, \quad (15)$$

where $\mathbf{D}_j(\mathbf{x}) = \text{diag}(\theta(\mathbf{M}_j \mathbf{f}_{j-1}(\mathbf{x}))) \in \mathbb{R}^{d_j \times d_j}$ and $\theta(z) = \frac{\partial \text{ReLU}(z)}{\partial z} = \begin{cases} 1, \text{if } z > 0 \\ 0, \text{if } z < 0, \end{cases}$.

The update matrix from Equation (13) is then:

$$\mathbf{J}_i^T(X_b) \mathbf{J}_i(X_b) = \begin{bmatrix} \mathbf{J}_i^T(\mathbf{x}_1) \cdots \mathbf{J}_i^T(\mathbf{x}_b) \end{bmatrix} \begin{bmatrix} \mathbf{J}_i(\mathbf{x}_1) \\ \vdots \\ \mathbf{J}_i(\mathbf{x}_b) \end{bmatrix} =$$
$$= \sum_{\mathbf{x} \in X_b} \mathbf{J}_i^T(\mathbf{x}) \mathbf{J}_i(\mathbf{x}). \quad (16)$$

Performance of the UAPs from (Khrulkov and Oseledets 2018) increases with the increase of batch size. Then considering the limit case of sufficiently large batch size b we can approximate the averaging in (16) by the expected value:

$$\frac{\sum_{\mathbf{x} \in X_b} \mathbf{J}_i^T(\mathbf{x}) \mathbf{J}_i(\mathbf{x})}{b} \approx \mathbb{E}_{\mathbf{x}} \left[\mathbf{J}_i^T(\mathbf{x}) \mathbf{J}_i(\mathbf{x}) \right] =$$
$$= \mathbb{E} \left[\mathbf{M}_1^T \mathbf{D}_1^T(\mathbf{x}) \cdots \mathbf{M}_i^T \mathbf{D}_i^T(\mathbf{x}) \mathbf{D}_i(\mathbf{x}) \mathbf{M}_i \cdots \mathbf{D}_1(\mathbf{x}) \mathbf{M}_1 \right]. \quad (17)$$

Note that $\mathbf{D}_j^T(\mathbf{x}) \mathbf{D}_j(\mathbf{x}) = \mathbf{D}_j^2(\mathbf{x}) = \mathbf{D}_j^T(\mathbf{x}) = \mathbf{D}_j(\mathbf{x})$. To simplify Equation (17), we make additional assumption.

Assumption 0. The elements of the diagonal matrices D_j have the same scalar mean value c_j:

$$\mathbb{E}_{\mathbf{x}} \left[\mathbf{D}_j(\mathbf{x}) \right] \approx c_j \mathbf{I}. \quad (18)$$

This is based on the fact that we consider universal attack, which is input-independent, we can assume $\mathbf{x}$ as random variable and $\mathbf{D}(\mathbf{x})$ as random matrix, which is diagonal and have elements 0 or 1. Then our assumption is not unrealistic, as we assume that all diagonal elements of random diagonal matrix over expectation converge to the same number c between 0 and 1 (see Appendix). Then, taking into consideration (16), (17), (18) for the first layer:

$$\mathbf{J}_1^T(X_b) \mathbf{J}_1(X_b) \approx b \mathbb{E}_{\mathbf{x}} \left[\mathbf{M}_1^T \mathbf{D}_1^T(\mathbf{x}) \mathbf{D}_1(\mathbf{x}) \mathbf{M}_1 \right] =$$
$$= b \mathbf{M}_1^T \mathbb{E}_{\mathbf{x}} \left[\mathbf{D}_1(\mathbf{x}) \right] \mathbf{M}_1 = b c_1 \mathbf{M}_1^T \mathbf{M}_1, \quad (19)$$

i.e. taking the expected value has removed the term corresponding to the non-linearity!

Two subsequent convolutions does not result into one convolution, however the only source of error is boundary effects. Since the size of convolutional kernels $d < 10$ is much smaller than the dimension of image $N = 224$ (for Imagenet), the error is small and is up to $d/N \approx 0.5 * 10^{-2}$. It definetely could be neglected for our purposes. Moreover, as shown in (Miyato et al. 2018), this is common in convolutional neural networks for images. One can advocate this by the asymptotic theory of Toeplitz matrices (Böttcher and Grudsky 2000). Thus, we have shown that $\mathbf{J}_1^T(X_b) \mathbf{J}_1(X_b)$ has an approximate convolutional structure. To show that $\mathbf{J}_i^T(X_b) \mathbf{J}_i(X_b)$ is also convolutional, we introduce additional assumptions:

Assumption 1. Matrices $\mathbf{D}_j(\mathbf{x})$ and $\mathbf{D}_k(\mathbf{x})$ are independent for all pairs of j and k. In practice we only will check the uncorrelation instead of independency, these checks are reported in appendix.
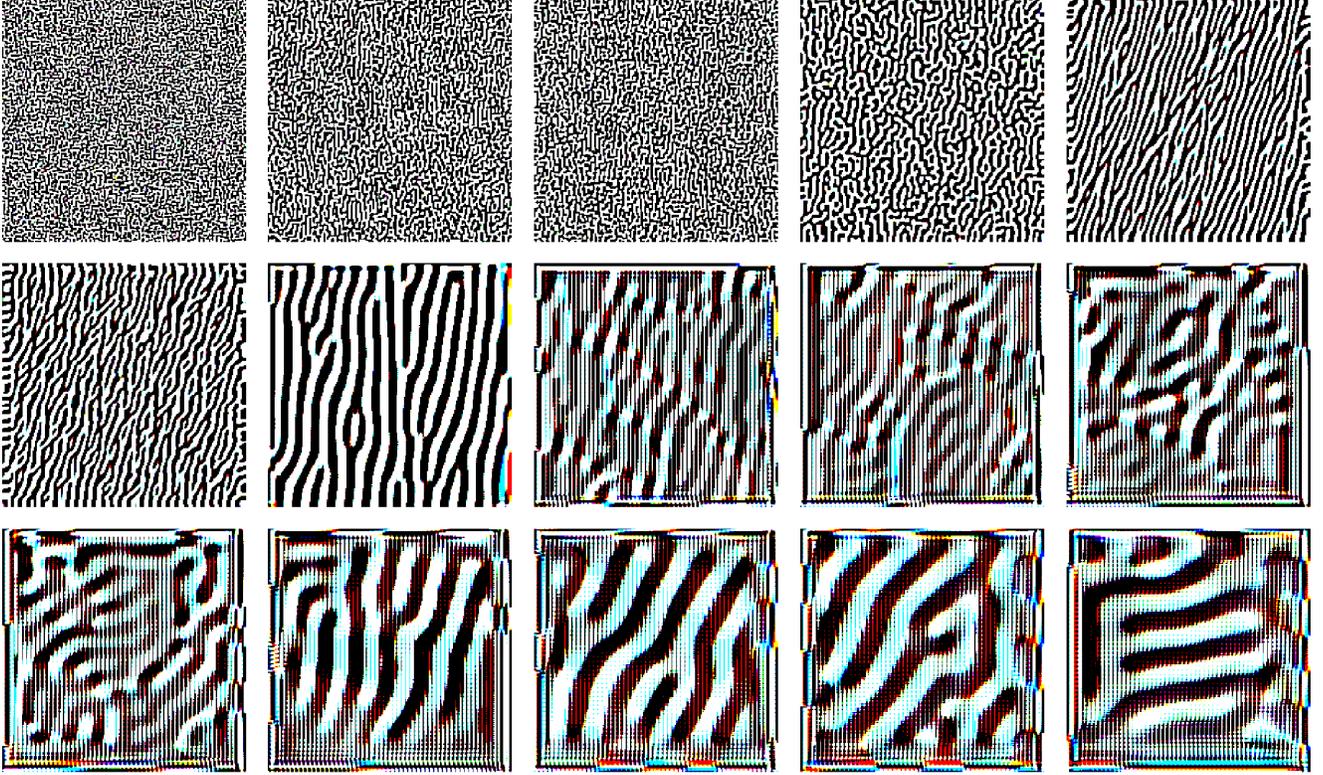
Figure 2: UAPs for different layers of VGG-19 classifier. According to the theory, the specific feature size of the patterns increases with the depth of the attacked layer. The numbering of images is row-wise. First row corresponds to Layers 1-5, second row - Layers 6-10, third row - Layers 11-15.

Assumption 2. Diagonal elements $\mathbf{d}_j = \theta(\mathbf{M}_j\mathbf{f}_{j-1}(\mathbf{x}))$ of matrix $\mathbf{D}_j(\mathbf{x})$ have covariance matrix of convolutional structure $\mathbf{C}_j$ as linear operators of typical convolutional layers:

$$\mathbf{Cov}(\mathbf{d}_j, \mathbf{d}_j) = \mathbf{C}_j. \tag{20}$$

These two assumptions might seem to be strong, so here we discuss how realistic and legit they are.

Regarding assumption 1, we should point that matrix $\mathbf{D}$ is not the outputs of ReLU activation function, but is indicator (0 and 1) whether the feature is positive or negative, and this is reasonable to assume the independence between elements $\mathbf{D}_i$ and $\mathbf{D}_j$. Further, let us consider covariaance of matrix elements of diagonal matrices $\mathbf{Cov}(\mathbf{D}_i, \mathbf{D}_j)$. First of all, for a large enough image batch it is natural to assume that this covariance if translationally invariant, i.e there is no selected points. This means $\mathbf{Cov}(\mathbf{D}_i^{p,q}, \mathbf{D}_i^{l,m}) = C_i(p-l, q-m)$ which is nothing but the assumption 2. It is also follows that because of finite receptive field, convolution matrix C_i is decreasing far from the diagonal.

These assumptions need additional conditions for satisfactory boundary effects, however we find them natural, and realistic to hold in the main part of features. Additional quantitative experimental justifications of these assumptions are in the Appendix.

Theorem 0.1 *Given* $\mathbf{x}$ *is a random variable,* $\mathbf{A} = \mathbf{A}(\mathbf{x})$ *is a square matrix, such that* $\mathbb{E}_x[\mathbf{A}] = \mathbf{B}$ *is convolutional, and* $\mathbf{D} = \mathbf{D}(\mathbf{x})$ *is a diagonal matrix, such that* $\mathbb{E}_{\mathbf{x}}[\mathbf{D}_{ll}\mathbf{D}_{mm}] = \mathbf{C}_{lm}$ *and matrices* $\mathbf{D}$ *and* $\mathbf{A}$ *are independent. Then,* $\mathbb{E}_x[\mathbf{DAD}]$ *is a convolutional matrix.*

Proof:

$$\mathbb{E}_x[\mathbf{DAD}(\mathbf{x})]_{lm} = \mathbb{E}_x[\mathbf{D}_{ll}\mathbf{A}_{lm}\mathbf{D}_{mm}] =$$
$$= \mathbb{E}_x[\mathbf{A}_{lm}]\,\mathbb{E}_x[\mathbf{D}_{ll}\mathbf{D}_{mm}] = \mathbf{B}_{lm}\mathbf{C}_{lm} = (\mathbf{B}\circ\mathbf{C})_{lm},$$

where $\circ$ denotes element-wise product.

Applying Theorem 0.1 for each of the layers, and using $\mathbf{D}_j = \mathbf{D}_j(\mathbf{x})$ for all j in (16) and (17):

$$\mathbf{J}_i^T(X_b)\mathbf{J}_i(X_b) =$$

$$= b\mathbb{E}\left[\mathbf{M}_1^T\mathbf{D}_1^T\overbrace{\cdots\mathbf{M}_i^T\mathbf{D}_i^T\mathbf{D}_i\mathbf{M}_i}^{\text{no correlation with }\mathbf{D}_1}\cdots\mathbf{D}_1\mathbf{M}_1\right] =$$

$$= b\mathbf{M}_i^T((\cdots\mathbf{M}_2^T((\mathbf{M}_1^T\mathbf{C}_1\mathbf{M}_1)\circ\mathbf{C}_2)\mathbf{M}_2\cdots)\circ\mathbf{C}_i)\mathbf{M}_i, \tag{21}$$

which concludes that matrix $\mathbf{J}_i^T(X_b)\mathbf{J}_i(X_b)$ is convolutional matrix and Boyd algorithm (6) indeed generates cellular automata (12).

| (a) MobileNetV2 | (b) VGG-19 | (c) InceptionV3 |

Figure 3: Turing patterns generated using simplest kernels, parameterized by two numbers $l1, l2$.

Target	Random	Optimized
MobileNetV2	52.79	56.15
VGG-19	52.56	55.4
InceptionV3	45.32	47.18

Table 1: FRs of a random Turing pattern and the one optimized in l_1, l_2 and initialization. Even random Turing patterns provide competitive results, proving high universality of this attack. Better optimizations are in the tables 6,7.

| Target model	With		Without	
	250 q	**500 q**	**250 q**	**500 q**
MobileNetV2	85.62	90.91	91.67	94.47
VGG-19	79.73	75.9	76.94	77.61
InceptionV3	52.9	54.2	53.71	53.23

Table 2: FRs of patterns with and without optimization on initialization (250 and 500 queries)

The structure of patterns by (Khrulkov and Oseledets 2018) contains some interesting elements, which has specific size. We empirically find that changing the train batch does not affect much neither form of pattern, nor its fooling rate (Equation (22)), even if we use batch of white images. This supports the idea that the update matrix $\mathbf{J}_i^T(X_b)\mathbf{J}_i(X_b)$ does not depend much on batch X_b, and can be replaced with constant convolutional matrix by Equation (21). Based on Equation (21), we see that deeper the layer i we apply algorithm in Equation (6) the more convolutions are involved. As it was mentioned in Section Turing patterns present alternating patterns with specific characteristic size. Based on this intuition, we hypothesize that the specific size of the patterns generated by (Khrulkov and Oseledets 2018) increases with the depth of the attacked layer for which UAP is constructed. To ensure this, we consider VGG-19 (Simonyan and Zisserman 2014) network. Initial layers of this network are convolutional with ReLU activation. We apply algorithm to each of the first 15 layers of the network. We chose $q = 2$ in Equation (6). As illustrated in Figure 2, the typical feature size of the patterns increases with the "depth of the attack".

Turing Patterns as Black-box Universal Adversarial Perturbations

The connection in the previous section suggests the use of Turing patterns to craft UAPs in a black-box setting, learning parameters of CA by directly maximizing the fooling rate.

As described earlier, complex Turing patterns can be modeled with only several parameters. In order to test the performance of Turing patterns as black-box UAPs for a specific network, we learn this parameters in a black-box scenario by maximizing the *fooling rate*, without accessing the model's architecture. For a classifier f and N images, the fooling rate of Turing pattern $\varepsilon(\theta)$ with parameters θ is:

$$\text{FR} = \frac{\sum\limits_{i=1}^{N}[\arg\max f(\mathbf{x}_i) \neq \arg\max f(\mathbf{x}_i + \varepsilon(\theta))]}{N}. \quad (22)$$

There are several options how to parametrize CA, and we discuss them one-by-one. For all approaches the target function is optimized on 512 random images from ImageNet (Russakovsky et al. 2015) train dataset, and perturbation is constrained as $\|\varepsilon\|_\infty \leq 10$. Following (Meunier, Atif, and Teytaud 2019), we used evolutionary algorithm CMA-ES (Hansen, Müller, and Koumoutsakos 2003) by Nevergrad (Rapin and Teytaud 2018) as a black-box optimization algorithm. Fooling rates are calculated for 10000 Imagenet validation images for torchvision pretrained models(VGG-19 (Simonyan and Zisserman 2014), InceptionV3 (Szegedy et al. 2016), MobilenetV2 (Sandler et al. 2018)).

Simple CA. Here, we fix the kernel Y in Equation (12) to be $L \times L$ (we find that $L = 13$ produces best results), with elements filled by -1, except for the inner central rectangle of size $l_1 \times l_2$ with constant elements such that the sum of all elements of Y is 0. Besides l_1 and l_2, the ini-
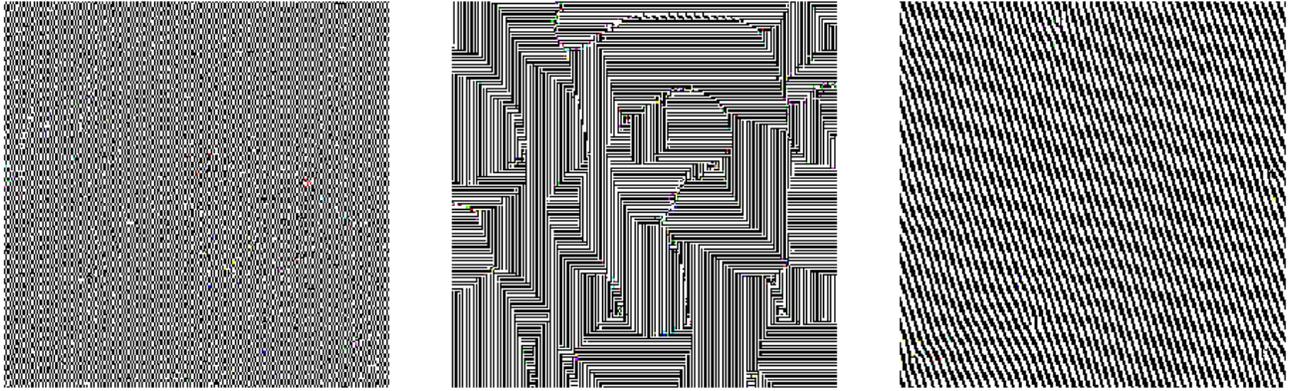
(a) MobileNetV2   (b) VGG-19   (c) InceptionV3

Figure 4: Turing patterns generated using optimization on initialization and convolutional kernel Y

Threshold / Mixing	max	$max - 1$	$0.9 * max$
Summation	77.5	78.0	77.9
Pointwise	78.0	77.7	78.0

Table 3: Experiments with DFT frequencies for VGG-19

| Threshold / Mixing	max	$max - 1$	$0.9 * max$
Summation	53.1	50.6	51.0
Pointwise	53.3	53.6	53.5

Table 4: Experiments with DFT frequencies for InceptionV3

tialization of $n(i, j)$ from Eq. (12) is added as parameters. To reduce black-box queries, initialization is chosen to be 7×7 square tiles (size 32×32) (Meunier, Atif, and Teytaud 2019) for each of the 3 maps representing each image channel. $\theta = (l_1, l_2, \texttt{initialization})$. Resulting UAPs are shown in Fig. 3. Improvement over the random Turing pattern (with random, not optimized parameters) is shown in Table 1 and full results are in Table 5. We should point, here, our method is black-box, i.e. without access to the architecture and weights, and we do not compare it to (Khrulkov and Oseledets 2018) as they consider white-box scenario with full access to the network. Moreover, even if we consider black-box scenario, our results using more advanced optimization are significantly better in terms of fooling rate (see Tables 6-7) even comparing to the white-box.

CA with kernel optimization. In this scenario, all elements of the kernel Y are considered as unknown parameters. Optimizing both over Y and the initialization maps we got a substantial increase in fooling rates (see Fig. 4 for perturbations and Table 6 for fooling rate results). Results for different kernel sizes are shown in Fig. 5a.

Optimization with random initialization map. Here, we remove the initialization map from the optimized parameters select it randomly. As experiments show (see Table 2), the performance of patterns generated without optimized initialization maps does not differ significantly from the pattern with optimized initialization maps. Thus, the initialization maps for pattern generation can be randomly initialized, and less queries can be made without significant loss in fooling rates. Results for different kernel sizes are shown in Fig. 5b.
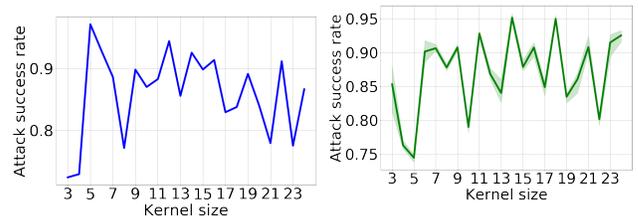
(a) Optimization over kernel and initialization maps

(b) Optimization over kernel only, averaged over 10 initializations (min and max highlighted)

Figure 5: Pattern fooling rate (vertical axis) dependence on the kernel size (horizontal axis). This figure shows that there is no dependence on the kernel size.

Another resource for optimization are different strategies for dealing with the channels of the image (This might produce color patches). The detailed experiments are given in the Appendix, the results for the best case are in Table 7.

Similarity to Fourier attacks

Some of the patterns are similar to the ones generated as single Fourier harmonics, in (Tsuzuku and Sato 2019). To compare correspondence, we took the discrete Fourier transform (DFT) of the pattern and cut all the frequencies with the amplitude below some threshold (we used 2 cases of thresholds: $max - 1$ and $0.9 \max$) and then reversed the DFT for all three pattern channels. We applied this transformation to the patterns acquired using different channel mixing approaches (2 (summation) and 4 (pointwise) described

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	56.15	56.96	45.16
VGG-19	50.49	55.4	49.98
InceptionV3	49.55	48.76	47.18

Table 5: Optimization on l_1, l_2, and initialization

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	90.59	61.27	35.15
VGG-19	62.15	75.64	31.91
InceptionV3	65.69	76.40	53.93

Table 6: Optimization on kernel Y and initialization

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	94.78	57.90	35.04
VGG-19	73.15	77.50	43.06
InceptionV3	63.90	64.46	53.10

Table 7: Summation channel mixing

	ResNet	GoogleNet	VGG-16
SFA	40.1	44.1	53.3
Ours	**61.4**	**58.2**	**81.2**

Table 8: Performance of Turing Patterns (2 parameters) and SFA: Single Fourier Attack (Tsuzuku and Sato 2019) (224*224*3 parameters) in a black-box scenario

above). As a result, in most cases neither the fooling rates (see Table 3, 4) nor the visual representation of the pattern (Appendix, Fig. 6) did not change drastically, which leads us to the conclusion that in these cases it is indeed a Single Fourier Attack that is taking place. However, in the case of patterns generated using summation channel mixing to attack InceptionV3 it is not true (Appendix, Fig. 7). This means that while such methods can generate Single Fourier Attacks, not all the attacks generated using these methods are SFA, i.e. our approach can generate more diverse UAPs. Our work compared to (Tsuzuku and Sato 2019) has much less parameters and performs better (see Table 8).

Related Work

Adversarial Perturbations. Intriguing vulnerability of neural networks in (Szegedy et al. 2013; Biggio et al. 2013) proposed numerous techniques to construct white-box (Goodfellow, Shlens, and Szegedy 2015; Carlini and Wagner 2017; Moosavi-Dezfooli, Fawzi, and Frossard 2016; Madry et al. 2017) and black-box (Papernot et al. 2017; Brendel, Rauber, and Bethge 2017; Chen and Pock 2017; Ilyas et al. 2018) adversarial examples. Modern countermeasures against adversarial examples have been shown to be brittle (Athalye, Carlini, and Wagner 2018). Interestingly, input-agnostic Universal Adversarial Perturbations were discovered recently (Moosavi-Dezfooli et al. 2017). Several methods were proposed to craft UAPs (Khrulkov and Oseledets 2018; Mopuri, Garg, and Babu 2017; Tsuzuku and Sato 2019). Mopuri et al. (Mopuri, Garg, and Babu 2017) used activation-maximization approach. Khrulkov et al. (Khrulkov and Oseledets 2018) proposed efficient use of $(p, q)-$singular vectors to perturb hidden layers. Tsuzuku et al. (Tsuzuku and Sato 2019) used Fourier harmonics of convolutional layers without any activation function. Though UAPs were first found in image classification, they exist in other areas as well, such as semantic segmentation (Metzen et al. 2017), text classifiers (Behjati et al. 2019), speech and audio recognition (Abdoli et al. 2019; Neekhara et al. 2019).

Turing Patterns. Initially introduced by Alan Turing in (Turing 1952), Turing patterns have attracted a lot of attention in pattern formation studies. Patterns like fronts, hexagons, spirals and stripes are found as a result of Turing reaction–diffusion equations (Kondo and Miura 2010). In nature, Turing patterns has served as a model of fish skin pigmentation (Nakamasu et al. 2009), leopard skin patterns (Liu, Liaw, and Maini 2006), seashell patterns (Fowler, Meinhardt, and Prusinkiewicz 1992). Interesting to Neuroscience community, might be connection of Turing patterns to brain (Cartwright 2002; Lefèvre and Mangin 2010).

Cellular Automata in Deep Learning. Relations between Neural Networks and Cellular Automata were discovered in several works (Wulff and Hertz 1993; Gilpin 2019; Mordvintsev et al. 2020). Cellular automaton dynamics with neural network was studied in (Wulff and Hertz 1993). In (Gilpin 2019) authors generate cellular automata as convolutional neural network. In (Mordvintsev et al. 2020) it was shown that training cellular automata using neural networks can organize cells into shapes of complex structure.

Conclusion

The key technical point of our work is that a complicated Jacobian in the Boyd iteration can be replaced without sacrificing the fooling rate by a single convolution, leading to a remarkably simple low-parametric cellular automata that generates universal adversarial perturbations. It also explains the similarity between UAPs and Turing patterns. It means that the non-linearities are effectively averaged out from the process. This idea could be potentially useful for other problems involving the Jacobians of the loss functions for complicated neural networks, e.g. spectral normalization.

This work studies theory of interesting phenomena of Universal Adversarial Perturbations. Fundamental susceptibility of deep CNNs to small alternating patterns, that are easily generated, added to images was shown in this work and their connection to Turing patterns, a rigorous theory in mathematical biology that describes many natural patterns. This might help to understand the theory of adversarial perturbations, and in the future to robustly defend from them.

Acknowlegements

References

Abdoli, S.; Hafemann, L. G.; Rony, J.; Ayed, I. B.; Cardinal, P.; and Koerich, A. L. 2019. Universal adversarial audio perturbations. *arXiv preprint arXiv:1908.03173* .

Athalye, A.; Carlini, N.; and Wagner, D. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420* .

Behjati, M.; Moosavi-Dezfooli, S.-M.; Baghshah, M. S.; and Frossard, P. 2019. Universal adversarial attacks on text classifiers. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 7345–7349. IEEE.

Biggio, B.; Corona, I.; Maiorca, D.; Nelson, B.; Šrndić, N.; Laskov, P.; Giacinto, G.; and Roli, F. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, 387–402. Springer.

Böttcher, A.; and Grudsky, S. M. 2000. *Toeplitz matrices, asymptotic linear algebra and functional analysis*, volume 67. Springer.

Boyd, D. W. 1974. The power method for lp norms. *Linear Algebra and its Applications* 9: 95–101.

Brendel, W.; Rauber, J.; and Bethge, M. 2017. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248* .

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

Cartwright, J. H. 2002. Labyrinthine Turing pattern formation in the cerebral cortex. *arXiv preprint nlin/0211001* .

Chen, Y.; and Pock, T. 2017. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE transactions on pattern analysis and machine intelligence* 39(6): 1256–1272.

Fowler, D. R.; Meinhardt, H.; and Prusinkiewicz, P. 1992. Modeling seashells. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, 379–387.

Gilpin, W. 2019. Cellular automata as convolutional neural networks. *Physical Review E* 100(3): 032402.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*. URL http://arxiv.org/abs/1412.6572.

Hansen, N.; Müller, S. D.; and Koumoutsakos, P. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation* 11(1): 1–18.

Ilyas, A.; Engstrom, L.; Athalye, A.; and Lin, J. 2018. Black-box adversarial attacks with limited queries and information. *arXiv preprint arXiv:1804.08598* .

Khrulkov, V.; and Oseledets, I. 2018. Art of singular vectors and universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8562–8570.

Kondo, S.; and Miura, T. 2010. Reaction-diffusion model as a framework for understanding biological pattern formation. *science* 329(5999): 1616–1620.

Lefèvre, J.; and Mangin, J.-F. 2010. A reaction-diffusion model of human brain development. *PLoS computational biology* 6(4): e1000749.

Liu, R.; Liaw, S.; and Maini, P. 2006. Two-stage Turing model for generating pigment patterns on the leopard and the jaguar. *Physical review E* 74(1): 011914.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* .

Metzen, J. H.; Kumar, M. C.; Brox, T.; and Fischer, V. 2017. Universal adversarial perturbations against semantic image segmentation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2774–2783. IEEE.

Meunier, L.; Atif, J.; and Teytaud, O. 2019. Yet another but more efficient black-box adversarial attack: tiling and evolution strategies. *arXiv preprint arXiv:1910.02244* .

Miyato, T.; Kataoka, T.; Koyama, M.; and Yoshida, Y. 2018. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957* .

Moosavi-Dezfooli, S.-M.; Fawzi, A.; Fawzi, O.; and Frossard, P. 2017. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1765–1773.

Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2574–2582.

Mopuri, K. R.; Garg, U.; and Babu, R. V. 2017. Fast feature fool: A data independent approach to universal adversarial perturbations. *arXiv preprint arXiv:1707.05572* .

Mordvintsev, A.; Randazzo, E.; Niklasson, E.; and Levin, M. 2020. Growing neural cellular automata. *Distill* 5(2): e23.

Nakamasu, A.; Takahashi, G.; Kanbe, A.; and Kondo, S. 2009. Interactions between zebrafish pigment cells responsible for the generation of Turing patterns. *Proceedings of the National Academy of Sciences* 106(21): 8429–8434.

Neekhara, P.; Hussain, S.; Pandey, P.; Dubnov, S.; McAuley, J.; and Koushanfar, F. 2019. Universal adversarial perturbations for speech recognition systems. *arXiv preprint arXiv:1905.03828* .

Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z. B.; and Swami, A. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 506–519.

Rapin, J.; and Teytaud, O. 2018. Nevergrad-A gradient-free optimization platform. *version 0.2. 0, https://GitHub. com/FacebookResearch/Nevergrad* .

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115(3): 211–252.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* .

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* .

Tsuzuku, Y.; and Sato, I. 2019. On the Structural Sensitivity of Deep Convolutional Networks to the Directions of Fourier Basis Functions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 51–60.

Turing, A. M. 1952. The chemical basis of morphogenesis. *Bulletin of mathematical biology* 52(1-2): 153–197.

Wulff, N.; and Hertz, J. A. 1993. Learning cellular automaton dynamics with neural networks. In *Advances in Neural Information Processing Systems*, 631–638.

Young, D. A. 1984. A local activator-inhibitor model of vertebrate skin patterns. *Mathematical Biosciences* 72(1): 51–58.