# Generative Adversarial Networks (GANs) and adversarial examples

Theories of Deep Learning: C6.5, Lecture / Video 13

*Prof. Jared Tanner*
*Mathematical Institute*
*University of Oxford*

University of Oxford
Mathematical Institute

Oxford Mathematics

Example of a deep convolutional generator:

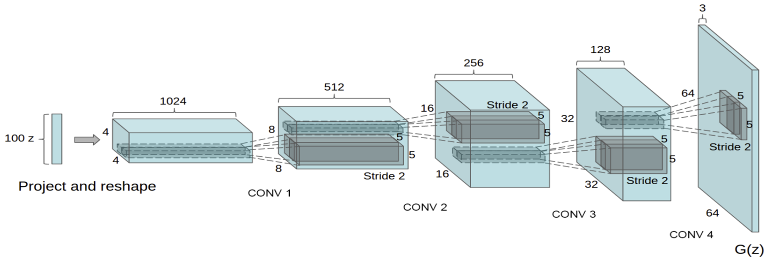



Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution $Z$ is projected to a small spatial extent convolutional representation with many feature maps.

https://arxiv.org/pdf/1511.06434.pdf

https://arxiv.org/pdf/1406.2661.pdf

Train the two network parameters using the objective

$$\min_G \max_D n^{-1}\sum_{\mu=1}^{n}\log(D(x_\mu,y_\mu))+p^{-1}\sum_p \log\left(1-D(G(z_p),y_p)\right)$$

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k=1$, the least expensive option, in our experiments.

**for** number of training iterations **do**
  **for** $k$ steps **do**
  - Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{x}^{(i)}\right)+\log\left(1-D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  **end for**
  - Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

**end for**
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

`https://arxiv.org/pdf/1406.2661.pdf`

# Generative deep nets (Radford et al. 16')

Early training examples

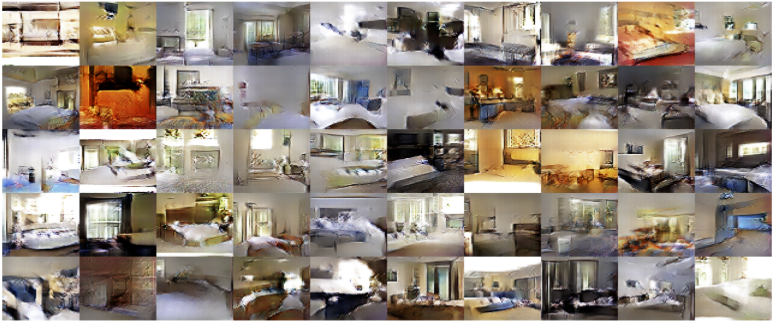


Figure 2: Generated bedrooms after one training pass through the dataset. Theoretically, the model could learn to memorize training examples, but this is experimentally unlikely as we train with a small learning rate and minibatch SGD. We are aware of no prior empirical evidence demonstrating memorization with SGD and a small learning rate.

https://arxiv.org/pdf/1511.06434.pdf

# Generative deep nets (Radford et al. 16')

Later training examples

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

https://arxiv.org/pdf/1511.06434.pdf

# Wasserstein GAN (Arjovsky et al. 17')

## Optimal transport decoder



One of the central challenges with GANs is the ability to train the parameters. Improvements have been made through choice of generative architecture (DC-GAN of Radford) and through different training objective functions (W-GAN)

**Algorithm 1** WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

**Require:** The gradient penalty coefficient $\lambda$, the number of critic iterations per generator iteration $n_{\text{critic}}$, the batch size $m$, Adam hyperparameters $\alpha, \beta_1, \beta_2$.
**Require:** initial critic parameters $w_0$, initial generator parameters $\theta_0$.

**while** $\theta$ has not converged **do**
  **for** $t = 1, \ldots, n_{\text{critic}}$ **do**
    **for** $i = 1, \ldots, m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    **end for**
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  **end for**
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^m -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
**end while**

https://arxiv.org/pdf/1704.00028.pdf

https://arxiv.org/pdf/1701.07875.pdf

# Wasserstein GAN (Arjovsky et al. 17')

## Examples of output from GAN architectures



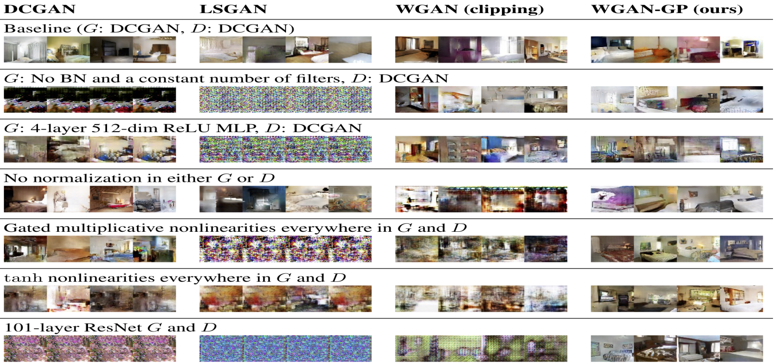



Figure 2: Different GAN architectures trained with different methods. We only succeeded in training every architecture with a shared set of hyperparameters using WGAN-GP.

https://arxiv.org/pdf/1704.00028.pdf

# Wasserstein GAN (Arjovsky et al. 17')

## Training rate

UNIVERSITY OF OXFORD
Mathematical Institute

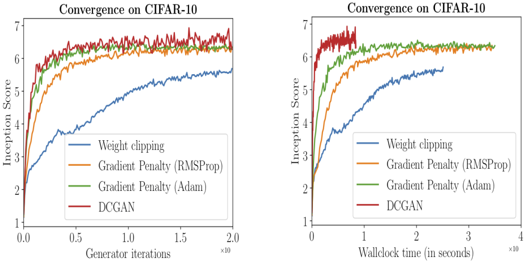



Figure 3: CIFAR-10 Inception score over generator iterations (left) or wall-clock time (right) for four models: WGAN with weight clipping, WGAN-GP with RMSProp and Adam (to control for the optimizer), and DCGAN. WGAN-GP significantly outperforms weight clipping and performs comparably to DCGAN.

`https://arxiv.org/pdf/1704.00028.pdf`

# Large scale WGAN (Karras et al. 18')

Growing the encoder/decoder complexity

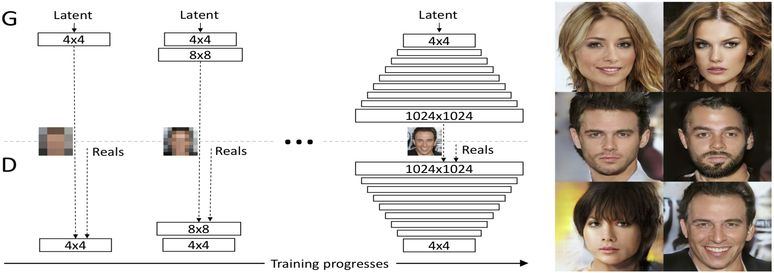



Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at $1024 \times 1024$.

https://arxiv.org/abs/1710.10196

# Large scale WGAN (Karras et al. 18')

## Examples of synthetic faces



Figure 10: Top: Our CELEBA-HQ results. Next five rows: Nearest neighbors found from the training data, based on feature-space distance. We used activations from five VGG layers, as suggested by Chen & Koltun (2017). Only the crop highlighted in bottom right image was used for comparison in order to exclude image background and focus the search on matching facial features.

https://arxiv.org/abs/1710.10196

# Understanding individual units in a DNN (Bau et al. 20’)

Single units which reliably detect object classes

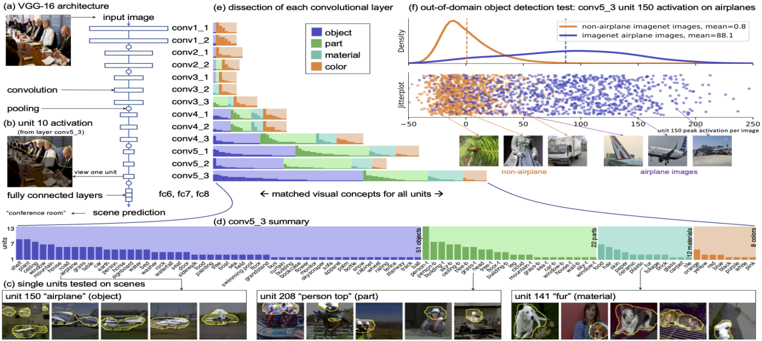



**Fig. 1.** The emergence of single-unit object detectors within a VGG-16 scene classifier. (a) VGG-16 consists of 13 convolutional layers, `conv1_1` through `conv5_3`, followed by three fully connected layers, `fc6,7,8`. (b) The activation of a single filter on an input image can be visualized as the region where the filter activates beyond its top 1% quantile level. (c) Single units are scored by matching high-activating regions against a set of human-interpretable visual concepts; each unit is labeled with its best-matching concept and visualized with maximally-activating images. (d) Concepts that match units in the final convolutional layer are summarized, showing a broad diversity of detectors for objects, object parts, materials, and colors. Many concepts are associated with multiple units. (e) Comparing all the layers of the network reveals that most object detectors emerge at the last convolutional layers. (f) Although the training set contains no object labels, unit 150 emerges as an ‘airplane’ object detector that activates much more strongly on airplane objects than non-airplane objects, as tested against a dataset of labeled object images not previously seen by the network. The jitter plot shows peak activations for the unit on randomly sampled 1,000 airplane and 1,000 non-airplane Imagenet images, and the curves show the kernel density estimates of these activations.

https://arxiv.org/abs/2009.05041

# Understanding individual units in a DNN (Bau et al. 20')

Single units which reliably generate object classes

UNIVERSITY OF OXFORD
Mathematical Institute

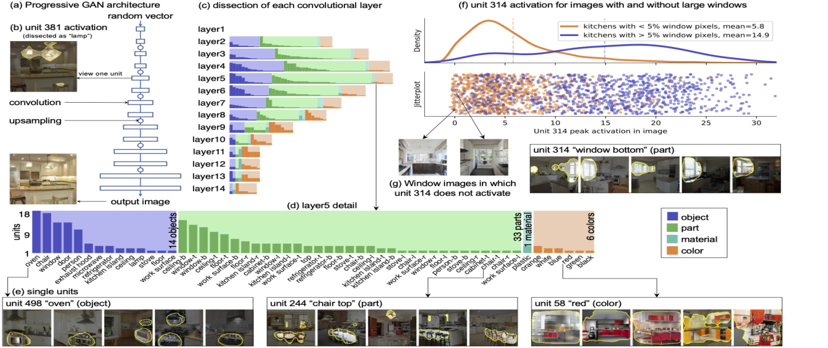



**Fig. 3.** The emergence of object- and part-specific units within a Progressive GAN generator'(19). (a) The analyzed Progressive GAN consists of 15 convolutional layers that transform a random input vector into a synthesized image of a kitchen. (b) A single filter is visualized as the region of the output image where the filter activates beyond its top 1% quantile level; note that the filters are all precursors to the output. (c) Dissecting all the layers of the network shows a peak in object-specific units at `layer5` of the network. (d) A detailed examination of `layer5` shows more part-specific units than objects, and many visual concepts corresponding to multiple units. (e) Units do not correspond to exact pixel patterns: a wide range of visual appearances for ovens and chairs are generated when an oven or chair part unit are activated. (f) When a unit specific to window parts is tested as a classifier, on average the unit activates more strongly on generated images that contain large windows than images that do not. The jitter plot shows the peak activation of unit 314 on 800 generated images that have windows larger than 5% of the image area as estimated by a segmentation algorithm, and 800 generated images that do not. (g) Some counterexamples: images for which unit 314 does not activate but where windows are synthesized nevertheless.

https://arxiv.org/abs/2009.05041

# Editing individual units in a DNN (Bau et al. 20')

Removing units of classes decreases their generation

UNIVERSITY OF OXFORD
Mathematical Institute

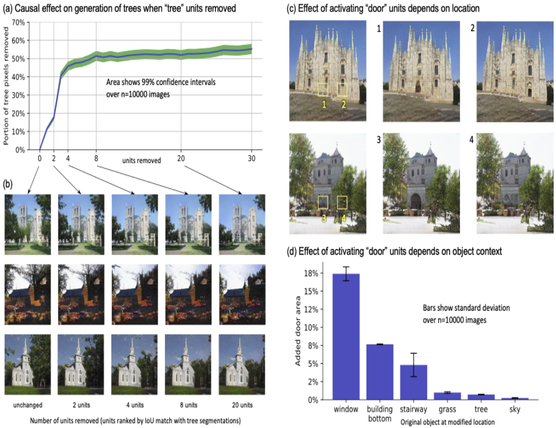



**Fig. 4.** The causal effect of altering units within a GAN generator. (a) When successively larger sets of units are removed from a GAN trained to generate outdoor church scenes, the tree area of the generated images is reduced. Removing 20 tree units removes more than half the generated tree pixels from the output. (b) Qualitative results: removing tree units affects trees while leaving other objects intact. Building parts that were previously occluded by trees are rendered as if revealing the objects that were behind the trees. (c) Doors can be added to buildings by activating 20 door units. The location, shape, size, and style of the rendered door depends on the location of the activated units. The same activation levels produce different doors, or no door at all (case 4) depending on locations. (d) Similar context dependence can be seen quantitatively: doors can be added in reasonable locations such as at the location of a window, but not in abnormal locations such as on a tree or in the sky.

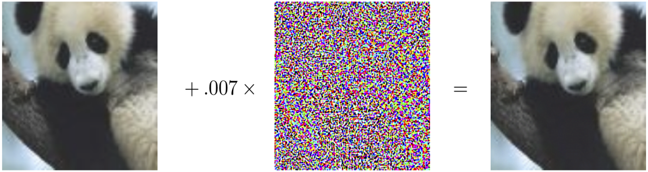
https://arxiv.org/abs/2009.05041

Adversarial Attacks for misclassification.

# Adversarial misclassification for deep nets (Goodfellow et al. 15')

Imperceptible perturbation changes classification






| $\boldsymbol{x}$ | $\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ | $\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ |
|---|---|---|
| "panda" 57.7% confidence | "nematode" 8.2% confidence | "gibbon" 99.3 % confidence |

Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our $\epsilon$ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers.

`https://arxiv.org/pdf/1412.6572.pdf`

# DeepFool algorithm (Moosavi-Dezfooli et al. 15')

Many algorithms exist for computing adversarial examples

**Algorithm 2** DeepFool: multi-class case

**input:** Image $\boldsymbol{x}$, classifier $f$.
**output:** Perturbation $\hat{\boldsymbol{r}}$.

Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
**while** $\hat{k}(\boldsymbol{x}_i) = \hat{k}(\boldsymbol{x}_0)$ **do**
  **for** $k \neq \hat{k}(\boldsymbol{x}_0)$ **do**
    $\boldsymbol{w}'_k \leftarrow \nabla f_k(\boldsymbol{x}_i) - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
    $f'_k \leftarrow f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
  **end for**
  $\hat{l} \leftarrow \arg\min_{k \neq \hat{k}(\boldsymbol{x}_0)} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_2}$
  $\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_2^2} \boldsymbol{w}'_{\hat{l}}$
  $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$
  $i \leftarrow i + 1$
**end while**
**return** $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$

Alternative to Goodfellow approach of
$\hat{r}(x_\mu) = \epsilon \text{sign}(\text{grad}_x \updownarrow(\theta; x_\mu, y_\mu))$.
`https://arxiv.org/pdf/1511.04599.pdf`

# DeepFool algorithm (Moosavi-Dezfooli et al. 15')

Many algorithms exist for computing adversarial examples



| Classifier | Test error | $\hat{\rho}_{\text{adv}}$ [DeepFool] | time | $\hat{\rho}_{\text{adv}}$ [4] | time | $\hat{\rho}_{\text{adv}}$ [18] | time |
|---|---|---|---|---|---|---|---|
| LeNet (MNIST) | 1% | $2.0 \times 10^{-1}$ | 110 ms | 1.0 | 20 ms | $2.5 \times 10^{-1}$ | $> 4$ s |
| FC500-150-10 (MNIST) | 1.7% | $1.1 \times 10^{-1}$ | 50 ms | $3.9 \times 10^{-1}$ | 10 ms | $1.2 \times 10^{-1}$ | $> 2$ s |
| NIN (CIFAR-10) | 11.5% | $2.3 \times 10^{-2}$ | 1100 ms | $1.2 \times 10^{-1}$ | 180 ms | $2.4 \times 10^{-2}$ | >50 s |
| LeNet (CIFAR-10) | 22.6% | $3.0 \times 10^{-2}$ | 220 ms | $1.3 \times 10^{-1}$ | 50 ms | $3.9 \times 10^{-2}$ | >7 s |
| CaffeNet (ILSVRC2012) | 42.6% | $2.7 \times 10^{-3}$ | 510 ms* | $3.5 \times 10^{-2}$ | 50 ms* | - | - |
| GoogLeNet (ILSVRC2012) | 31.3% | $1.9 \times 10^{-3}$ | 800 ms* | $4.7 \times 10^{-2}$ | 80 ms* | - | - |

Table 1: The adversarial robustness of different classifiers on different datasets. The time required to compute one sample for each method is given in the time columns. The times are computed on a Mid-2015 MacBook Pro without CUDA support. The asterisk marks determines the values computed using a GTX 750 Ti GPU.

Average relative error of adversarial example $\hat{r}(x)$ such that $f(x) \neq f(x + \hat{r}(x))$: $\hat{\rho}_{adv}(f) = |\mathcal{D}|^{-1} \sum_{x \in \mathcal{D}} \frac{\|\hat{r}(x)\|_2}{\|x\|_2}$

https://arxiv.org/pdf/1511.04599.pdf

# Rotations and Translations for CNNs (Engstrom et al. 18')

Adversarial action in space of a known invariant

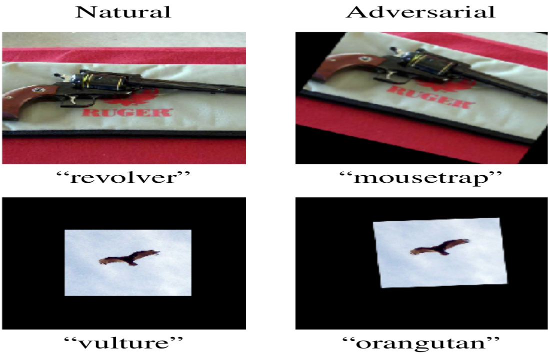



Figure 1: Examples of adversarial transformations and their predictions in the standard and "black canvas" setting.

https://arxiv.org/pdf/1712.02779.pdf

# Rotations and Translations for CNNs (Engstrom et al. 18')

## Loss landscape over known invariant

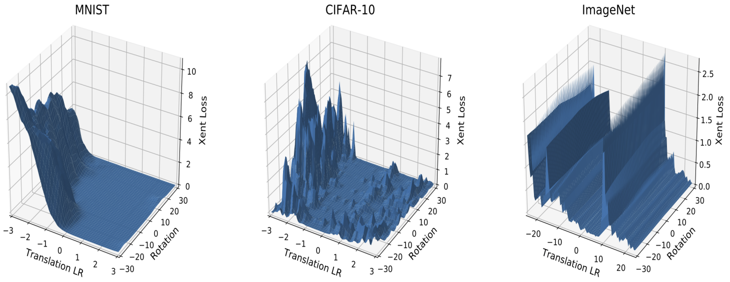



Figure 3: Loss landscape of a random example for each dataset when performing left-right translations and rotations. Translations and rotations are restricted to 10% of the image pixels and 30 deg respectively. We observe that the landscape is significantly non-concave, making rendering FO methods for adversarial example generation powerless. Additional examples are visualized in Figure 9 of the Appendix.

`https://arxiv.org/pdf/1712.02779.pdf`

# Universal adversary (Moosavi-Dezfooli et al. 16')

A single perturbation for many classes



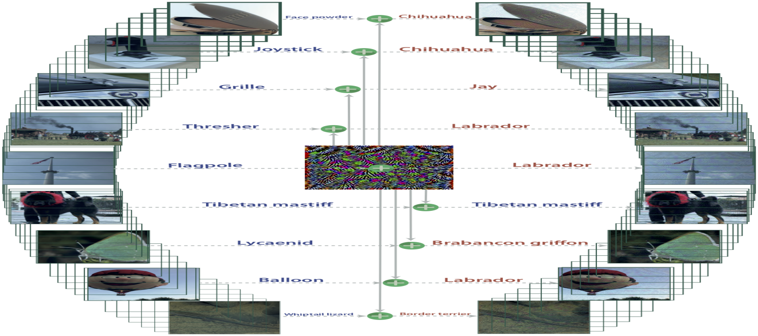



Figure 1: When added to a natural image, a universal perturbation image causes the image to be misclassified by the deep neural network with high probability. *Left images:* Original natural images. The labels are shown on top of each arrow. *Central image:* Universal perturbation. *Right images:* Perturbed images. The estimated labels of the perturbed images are shown on top of each arrow.

https://arxiv.org/pdf/1610.08401.pdf

# Transferability between nets (Liu et al. 16')

Can transfer adversarial examples between nets



| | RMSD | ResNet-152 | ResNet-101 | ResNet-50 | VGG-16 | GoogLeNet |
|---|---|---|---|---|---|---|
| -ResNet-152 | 17.17 | 0% | 0% | 0% | 0% | 0% |
| -ResNet-101 | 17.25 | 0% | 1% | 0% | 0% | 0% |
| -ResNet-50 | 17.25 | 0% | 0% | 2% | 0% | 0% |
| -VGG-16 | 17.80 | 0% | 0% | 0% | 6% | 0% |
| -GoogLeNet | 17.41 | 0% | 0% | 0% | 0% | 5% |

Table 4: Accuracy of non-targeted adversarial images generated using the optimization-based approach. The first column indicates the average RMSD of the generated adversarial images. Cell $(i, j)$ corresponds to the accuracy of the attack generated using four models except model $i$ (row) when evaluated over model $j$ (column). In each row, the minus sign "–" indicates that the model of the row is not used when generating the attacks. Results of top-5 accuracy can be found in the appendix (Table 14).

RMSD is the $\ell^2$ energy of the perturbation.

https://arxiv.org/pdf/1611.02770.pdf

# Transferability between nets (Liu et al. 16')

Can transfer adversarial examples between nets

University of Oxford Mathematical Institute

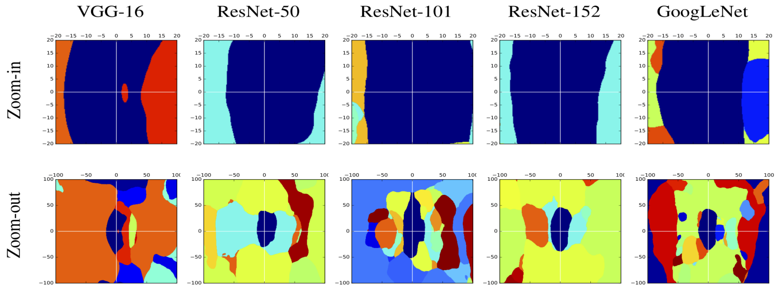



Figure 3: Decision regions of different models. We pick the same two directions for all plots: one is the gradient direction of VGG-16 (x-axis), and the other is a random orthogonal direction (y-axis). Each point in the span plane shows the predicted label of the image generated by adding a noise to the original image (e.g., the origin corresponds to the predicted label of the original image). The units of both axises are 1 pixel values. All sub-figure plots the regions on the span plane using the same color for the same label. The image is in Figure 2.

https://arxiv.org/pdf/1611.02770.pdf

# Adversarial physical object: Turtle (Athalye et al. 17')

Physical objects can be adversarial examples: 3D

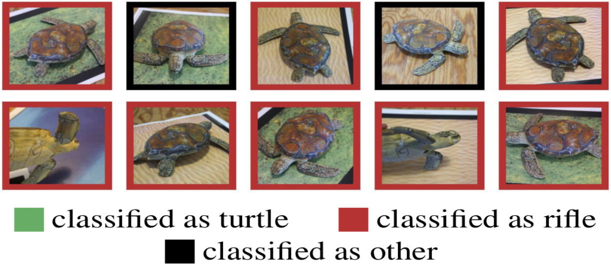



*Figure 1.* Randomly sampled poses of a 3D-printed turtle adversarially perturbed to classify as a rifle at every viewpoint[2]. An unperturbed model is classified correctly as a turtle nearly 100% of the time.

# Adversarial graffiti (Eykholt et al. 17')

Physical objects can be adversarial examples: 2D




Figure 1: The left image shows real graffiti on a Stop sign, something that most humans would not think is suspicious. The right image shows our a physical perturbation applied to a Stop sign. We design our perturbations to mimic graffiti, and thus "hide in the human psyche."

Table 5: A camouflage art attack on GTSRB-CNN. See example images in Table 1. The targeted-attack success rate is 80% (true class label: Stop, target: Speed Limit 80).

| Distance & Angle | Top Class (Confid.) | Second Class (Confid.) |
|---|---|---|
| 5′ 0° | Speed Limit 80 (0.88) | Speed Limit 70 (0.07) |
| 5′ 15° | Speed Limit 80 (0.94) | Stop (0.03) |
| 5′ 30° | Speed Limit 80 (0.86) | Keep Right (0.03) |
| 5′ 45° | Keep Right (0.82) | Speed Limit 80 (0.12) |
| 5′ 60° | Speed Limit 80 (0.55) | Stop (0.31) |
| 10′ 0° | Speed Limit 80 (0.98) | Speed Limit 100 (0.006) |
| 10′ 15° | Stop (0.75) | Speed Limit 80 (0.20) |
| 10′ 30° | Speed Limit 80 (0.77) | Speed Limit 100 (0.11) |
| 15′ 0° | Speed Limit 80 (0.98) | Speed Limit 100 (0.01) |
| 15′ 15° | Stop (0.90) | Speed Limit 80 (0.06) |
| 20′ 0° | Speed Limit 80 (0.95) | Speed Limit 100 (0.03) |
| 20′ 15° | Speed Limit 80 (0.97) | Speed Limit 100 (0.01) |
| 25′ 0° | Speed Limit 80 (0.99) | Speed Limit 70 (0.0008) |
| 30′ 0° | Speed Limit 80 (0.99) | Speed Limit 100 (0.002) |
| 40′ 0° | Speed Limit 80 (0.99) | Speed Limit 100 (0.002) |

# Provable defense: convex polytope pt. 1 (Wong et al. 17)

Ensuring neighbourhood also classified correctly

University of Oxford Mathematical Institute

Possible output of net $f_\theta(\cdot)$ from bounded perturbation is a non-convex set, say $\mathcal{Z}_\epsilon(x) = \{f_\theta(x+\delta) : \; \|\delta\|_\infty \le \epsilon\}$. A convex outer-polytope of $\mathcal{Z}_\epsilon(x)$, say $\mathcal{Z}_\epsilon^{conv}(x)$, can be computed by replacing the input to each activation with a 2D convex set:

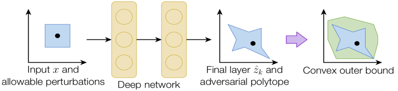



*Figure 1.* Conceptual illustration of the (non-convex) adversarial polytope, and an outer convex bound.

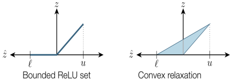



*Figure 2.* Illustration of the convex ReLU relaxation over the bounded set $[\ell, u]$.

Requires knowledge of lower and upper bound for each input to a nonlinear activation. Let $c = e_\ell - e_{\ell'}$ or $c = 2e_\ell - 1_{|class|}$ and solve:

$$\min_{\hat{z}_\ell \in \hat{\mathcal{Z}}_\epsilon(x)} \quad c^T \hat{z}_\ell \quad \text{and if nonnegative then robust to } \epsilon \text{ perturbation.}$$

https://arxiv.org/pdf/1711.00851.pdf

# Provable defense: convex polytope pt. 2 (Wong et al. 17)

Algorithm to determine range of pre-activation values

**Algorithm 1** Computing Activation Bounds

**input:** Network parameters $\{W_i, b_i\}_{i=1}^{k-1}$, data point $x$, ball size $\epsilon$

*// initialization*

$\hat{\nu}_1 := W_1^T$

$\gamma_1 := b_1^T$

$\ell_2 := x^T W_1^T + b_1^T - \epsilon \|W_1^T\|_{1,:}$

$u_2 := x^T W_1^T + b_1^T + \epsilon \|W_1^T\|_{1,:}$

*//* $\|\cdot\|_{1,:}$ *for a matrix here denotes* $\ell_1$ *norm of all columns*

**for** $i = 2, \ldots, k-1$ **do**

form $\mathcal{I}_i^-, \mathcal{I}_i^+, \mathcal{I}_i$; form $D_i$ as in (10)

*// initialize new terms*

$\nu_{i,\mathcal{I}_i} := (D_i)_{\mathcal{I}_i} W_i^T$

$\gamma_i := b_i^T$

*// propagate existing terms*

$\nu_{j,\mathcal{I}_j} := \nu_{j,\mathcal{I}_j} D_i W_i^T, \quad j = 2, \ldots, i-1$

$\gamma_j := \gamma_j D_i W_i^T, \quad j = 1, \ldots, i-1$

$\hat{\nu}_1 := \hat{\nu}_1 D_i W_i^T$

*// compute bounds*

$\psi_i := x^T \hat{\nu}_1 + \sum_{j=1}^{i} \gamma_j$

$\ell_{i+1} := \psi_i - \epsilon \|\hat{\nu}_1\|_{1,:} + \sum_{j=2}^{i} \sum_{i' \in \mathcal{I}_i} \ell_{j,i'} [-\nu_{j,i'}]_+$

$u_{i+1} := \psi_i + \epsilon \|\hat{\nu}_1\|_{1,:} - \sum_{j=2}^{i} \sum_{i' \in \mathcal{I}_i} \ell_{j,i'} [\nu_{j,i'}]_+$

**end for**

**output:** bounds $\{\ell_i, u_i\}_{i=2}^{k}$

https://arxiv.org/pdf/1711.00851.pdf

# Provable defense: convex polytope pt. 3 (Wong et al. 17)

Improved robustness, but increased non-adversarial test error



*Table 1.* Error rates for various problems and attacks, and our robust bound for baseline and robust models.

| PROBLEM | ROBUST | $\epsilon$ | TEST ERROR | FGSM ERROR | PGD ERROR | ROBUST ERROR BOUND |
|---|---|---|---|---|---|---|
| MNIST | × | 0.1 | 1.07% | 50.01% | 81.68% | 100% |
| MNIST | √ | 0.1 | 1.80% | 3.93% | 4.11% | 5.82% |
| FASHION-MNIST | × | 0.1 | 9.36% | 77.98% | 81.85% | 100% |
| FASHION-MNIST | √ | 0.1 | 21.73% | 31.25% | 31.63% | 34.53% |
| HAR | × | 0.05 | 4.95% | 60.57% | 63.82% | 81.56% |
| HAR | √ | 0.05 | 7.80% | 21.49% | 21.52% | 21.90% |
| SVHN | × | 0.01 | 16.01% | 62.21% | 83.43% | 100% |
| SVHN | √ | 0.01 | 20.38% | 33.28% | 33.74% | 40.67% |

https://arxiv.org/pdf/1711.00851.pdf

# Robustness via sparsification (Gopalakrishnan et al. 18)

Removing small values improves Lipshitz constant



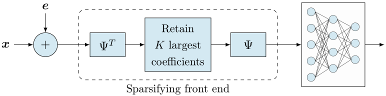



Figure 1: Sparsifying front end defense: For a basis in which the input is sparse, the input is projected onto the subspace spanned by the $K$ largest basis coefficients. This attenuates the impact of the attack by $K/N$, where $N$ is the input dimension.

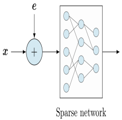



Figure 2: Network sparsity defense: Imposing sparsity *within* the neural network attenuates the worst-case growth of the attack as it flows up the network.

**Theorem 2.** *Consider an $\ell_\infty$-constrained input perturbation $\boldsymbol{e}_0 = \boldsymbol{e}$, with $\|\boldsymbol{e}\|_\infty \leq \epsilon$. Suppose that we impose $\ell_1$ constraints on the weights at each layer as follows:*

$$\|\boldsymbol{w}_{ij}\|_1 \leq \gamma_j \quad \forall\ i$$

*Then the effect of the perturbation is $\ell_\infty$-bounded at each layer:*

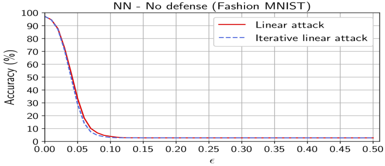
$$\|\boldsymbol{e}_j\|_\infty \leq \epsilon \prod_{l=1}^{j} \gamma_l \tag{2}$$

https://arxiv.org/abs/1810.10625

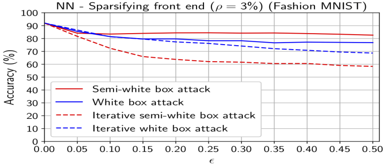
# Robustness via sparsification (Gopalakrishnan et al. 18)

More gradual decrease in accuracy as perturbation energy increases

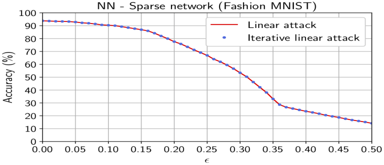
UNIVERSITY OF OXFORD
Mathematical Institute




(a) No defense




(b) With sparsifying front end




(c) With network sparsity

Figure 6: Fashion-MNIST: Binary classification accuracies as a function of $\epsilon$

https://arxiv.org/abs/1810.10625