UNIVERSITY
OF OSLO



IN4310 Adversarial Examples

Marius Aasan and Martine Hjelkrem Tan

mariuaas@uio.no matan@uio.no



Key Learning Goals

Robustness in computer vision models.

Adversarial robustness vs. classic robustness.

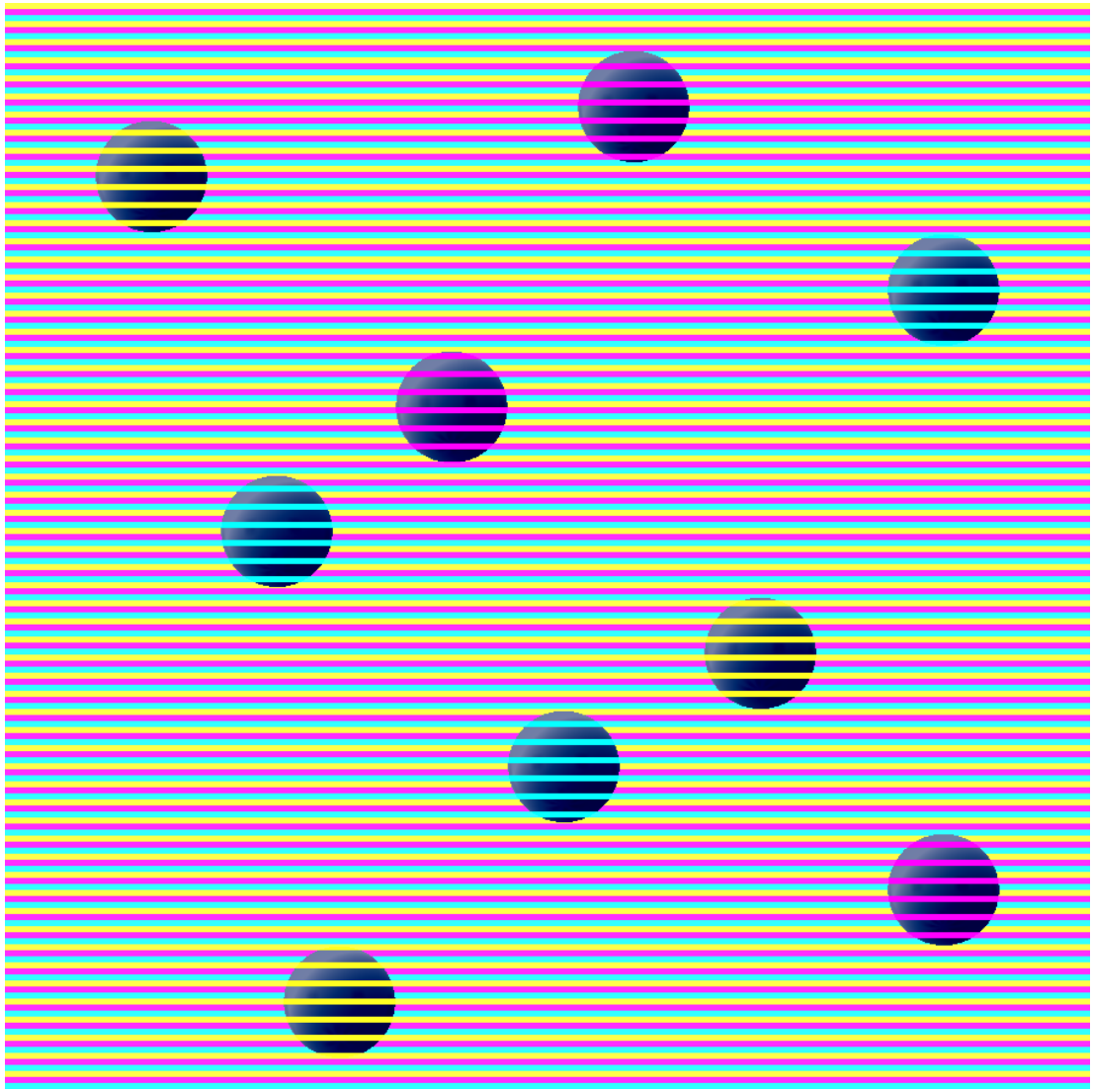
Different strategies for adversarial attacks.

Some defence strategies for adversarial attacks.

Regularization.

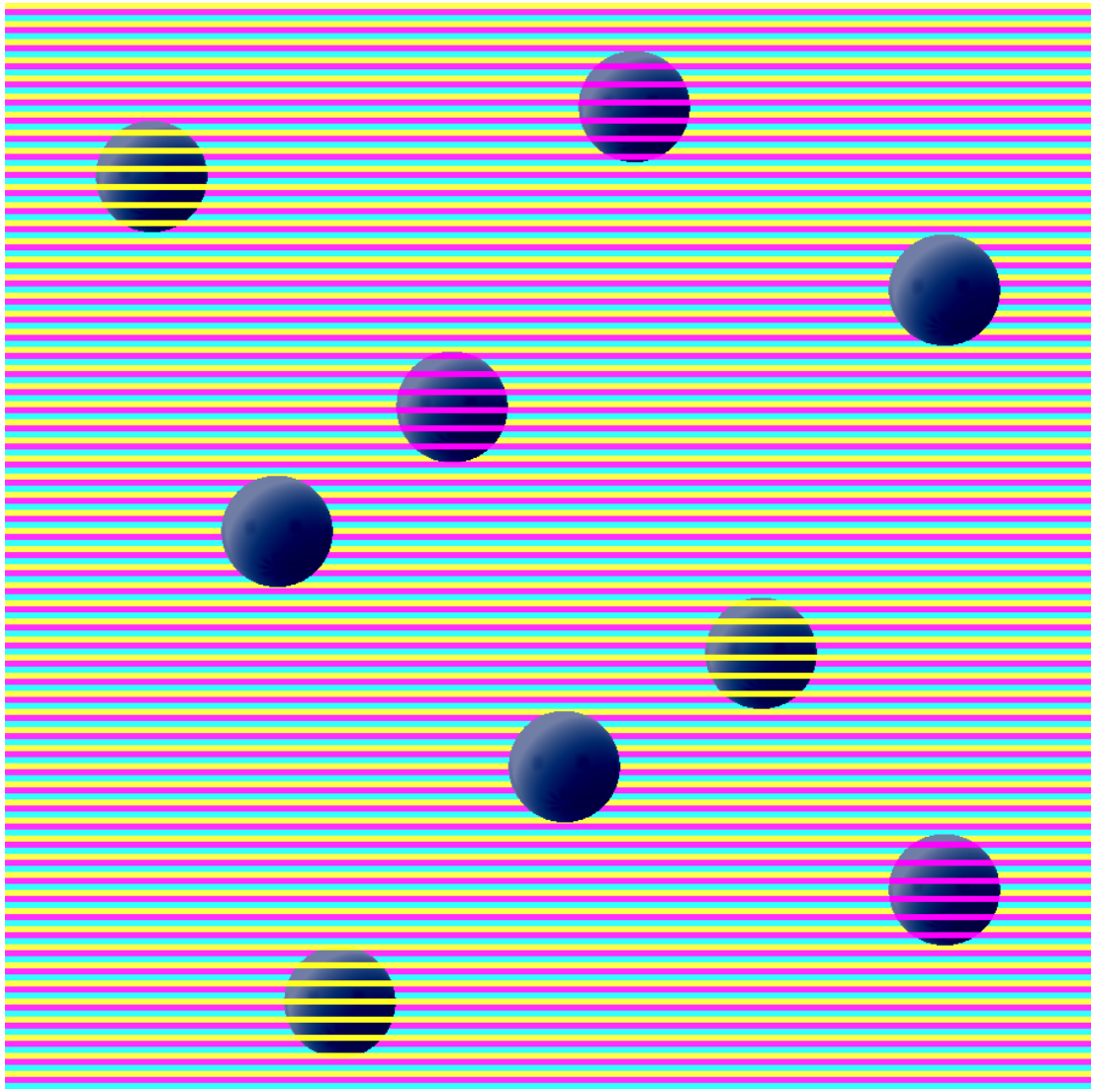
Example: Optical illusion

We are easily fooled...



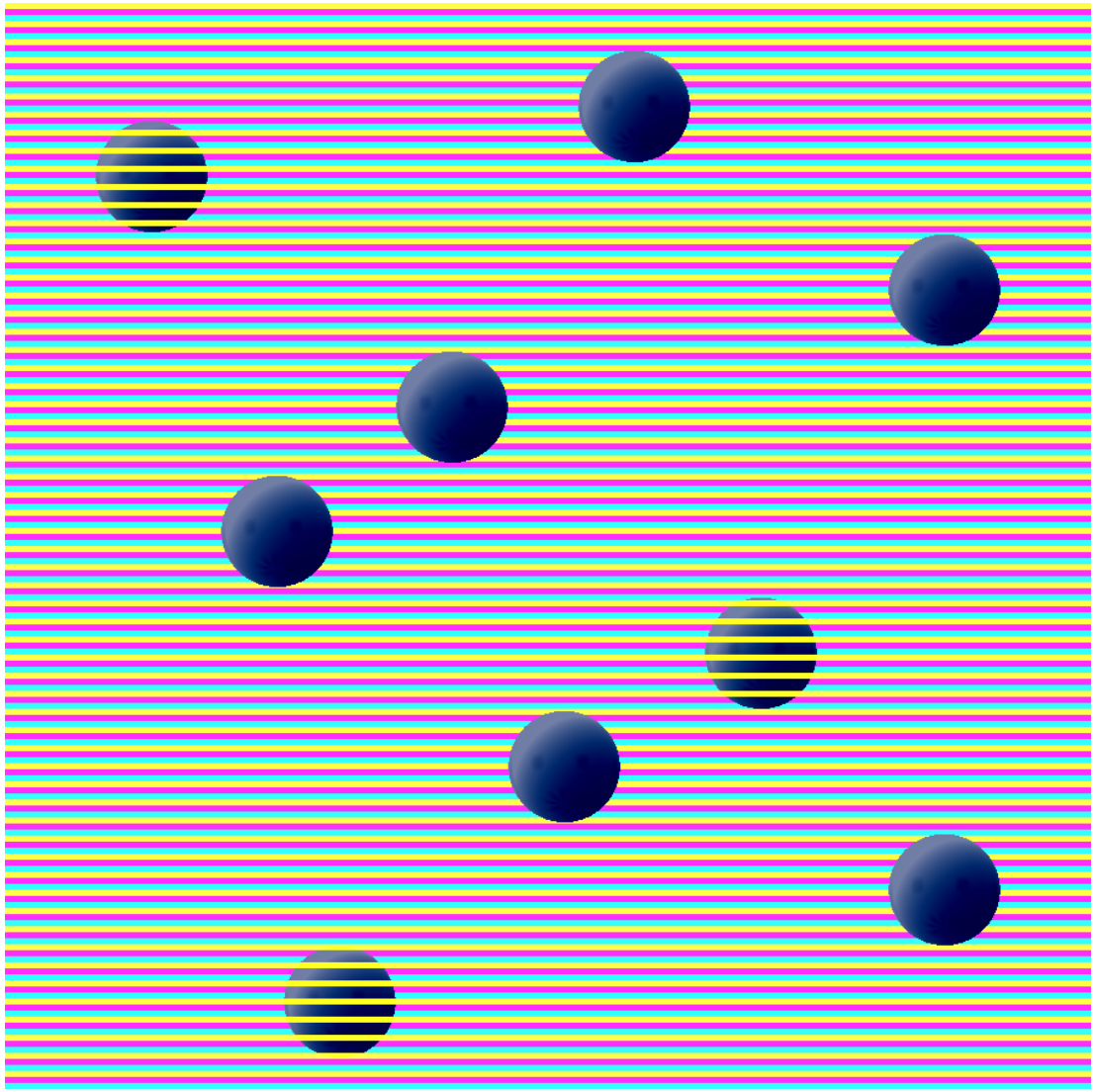
Example: Optical illusion

We are easily fooled...



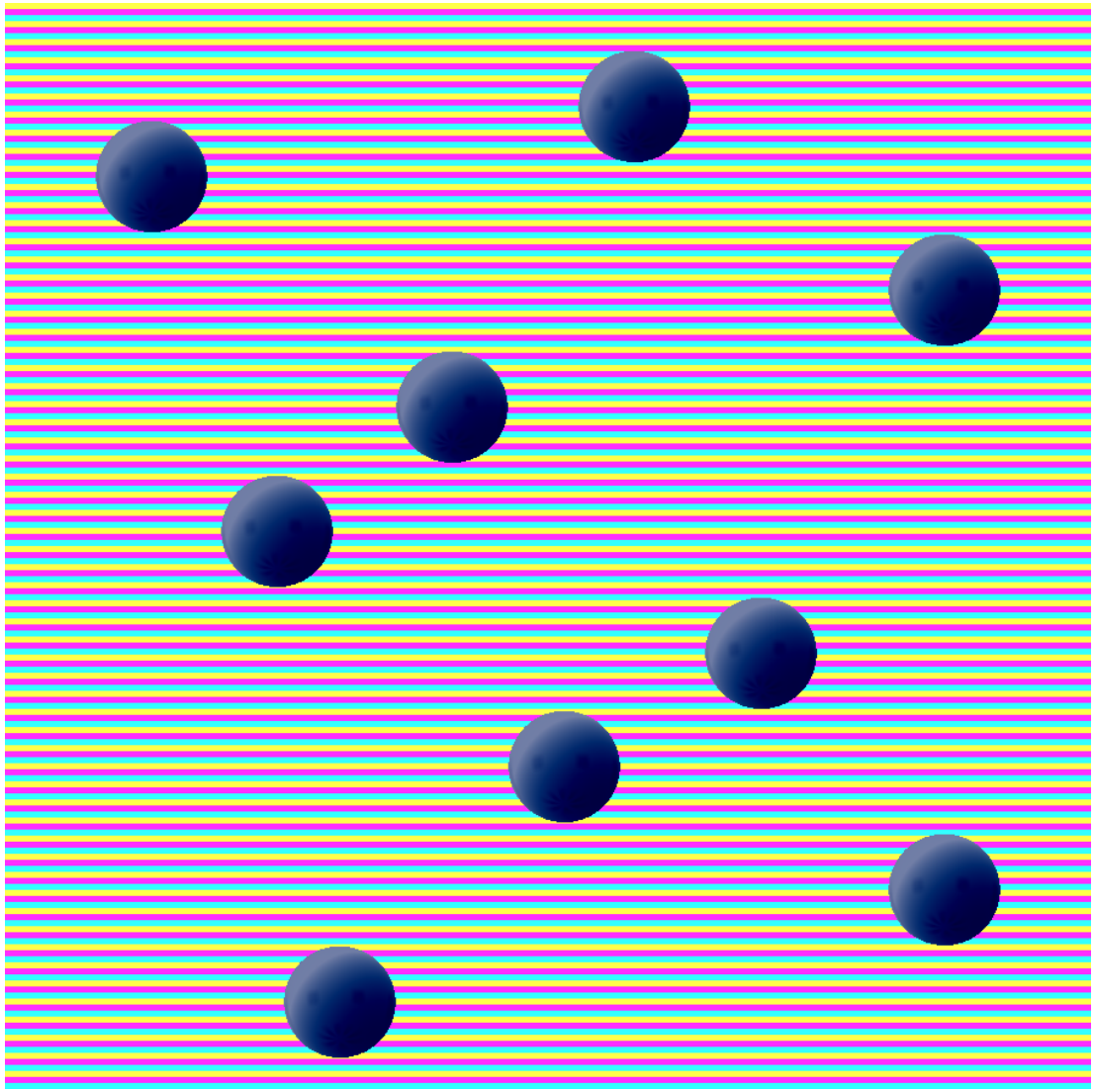
Example: Optical illusion

We are easily fooled...



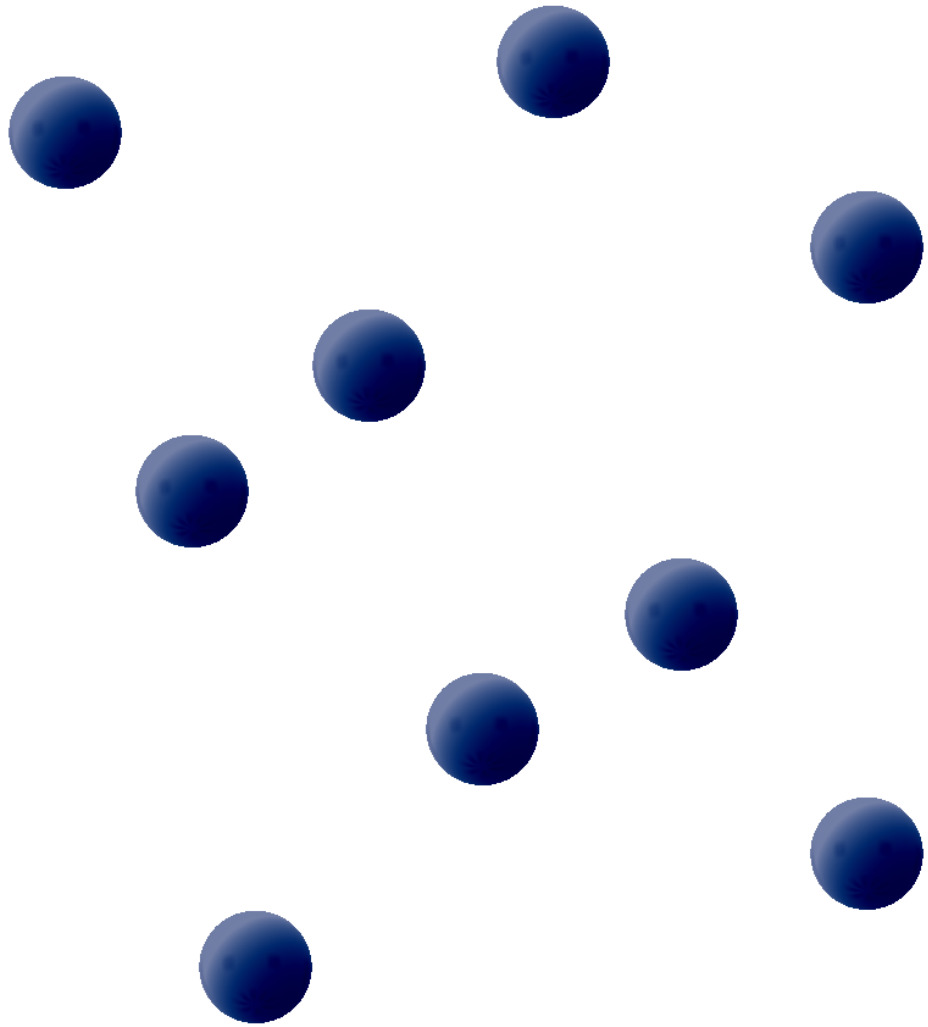
Example: Optical illusion

We are easily fooled...



Example: Optical illusion

We are easily fooled...



Reflection time

How do we fool cognitive systems?

- Example: Eykholt et al. used stickers to fool a deep learning classifier of road signs.
  - Would this fool humans?
- **Reflect:** What are the ethical implications of being able to manipulate AI decisions?



Graffiti
(a "natural" occurrence)

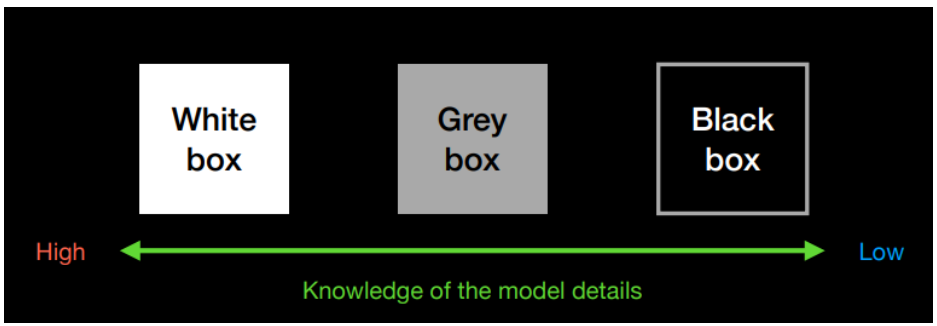


Physical perturbation
(stickers)

White box vs. black box attacks

How to trick neural networks

- **White box attacks:** The attacker has access to the model's parameters.
- **Black box attacks:** The attacker does *not* have access to the model's parameters, only its outputs.
- *We will mostly focus on white box attacks in this lecture.*



Targeted vs. untargeted attacks

How to trick neural networks

- Adversarial image: An image that has been slightly modified with the goal of tricking the classifier.
- **Untargeted attacks:** Trick the network to misclassify the adversarial image.
- **Targeted attacks:** Trick the network to classify a sample x into a fixed class which is different from the true class

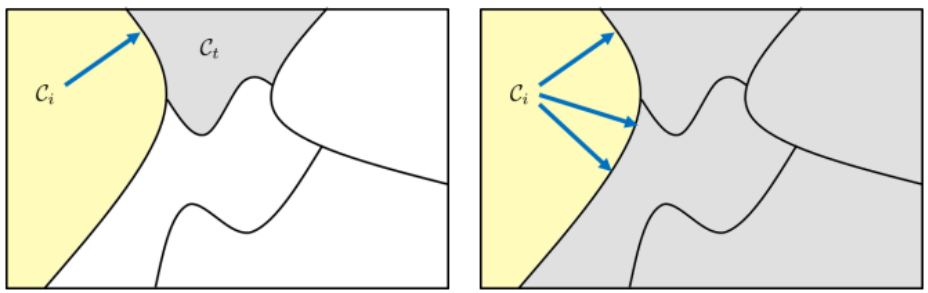
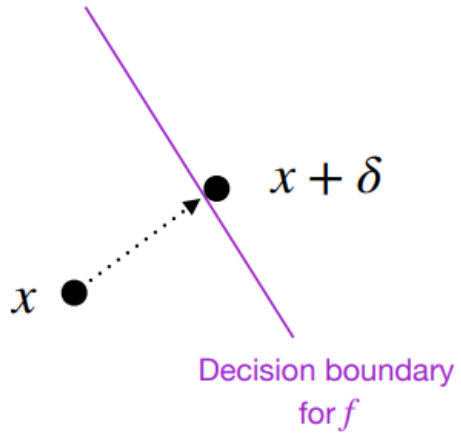


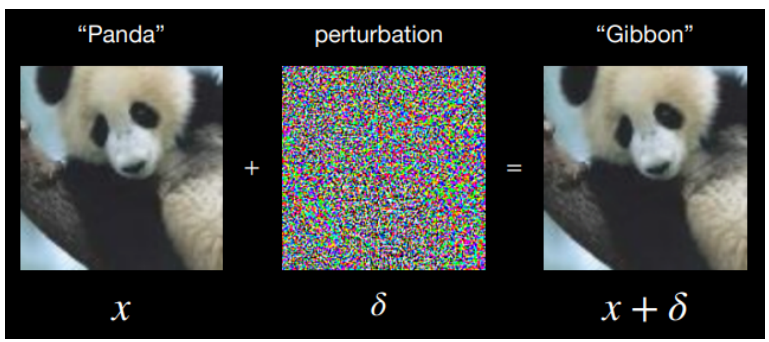
Image credit:
https://engineering.purdue.edu/ChanGroup/ECE595/files/chapter3.pdf

Finding adversarial examples

We need to be sneaky

- Suppose we have a classifier f, e.g. a pre-trained neural network
- Want to modify an input sample x by some small perturbation δ such that $x + \delta$ is very similar to x, but is still misclassified by f
- How can we find δ?
- **Idea:** Use gradient methods to iteratively perturb the input x until the model changes its prediction
- If we want an **untargeted** attack, it is sufficient that f changes its prediction. If we want a **targeted attack**, f needs to misclassify $x + \delta$ as a *specific* class.





Iterative gradient descent/ascent

White box attack

- We are given a trained model f for classification.
- Want a perturbation δ such that $x + \delta$ is misclassified, i.e. $\delta = \max_\delta L(c, f(x+\delta))$ where c is the original prediction.
- Two approaches: *ascent away from the original class (this slide)* and *descent towards the least probable class (next slide)*

Ascent away from original class

- Find original class prediction: $c^0 = \arg\max_c f(x)_c$
- Initialize $\delta_0 = 0$.
- Iterate

$$\delta_{n+1} = \delta_n + \eta \nabla_\delta L(c^0, f(x+\delta_n))$$

  where η is some small stepsize.
- Stop when prediction changes from original class i.e. $\arg\max_c f(x+\delta_n)_c \neq \arg\max_c f(x)_c$

NB: Let L be the loss of the logit outputs, not the softmax probabilities.

Question: Why is this important?

Iterative gradient descent/ascent

White box attack

- Two approaches: *ascent away from the original class (last slide)* and *descent towards the least probable class (this slide)*
- Can optionally combine approaches

$$\delta_{n+1} = \delta_n + \eta[\nabla_\delta L(c^0, f_\delta) - \nabla_\delta L(c^*, f_\delta)]$$

- **Targeted version:** Use the gradient descent method and set $c^* = c$, where c is the target class

NB: Abbreviated notation: $f(x+\delta) = f_\delta$

Descent towards least probable class

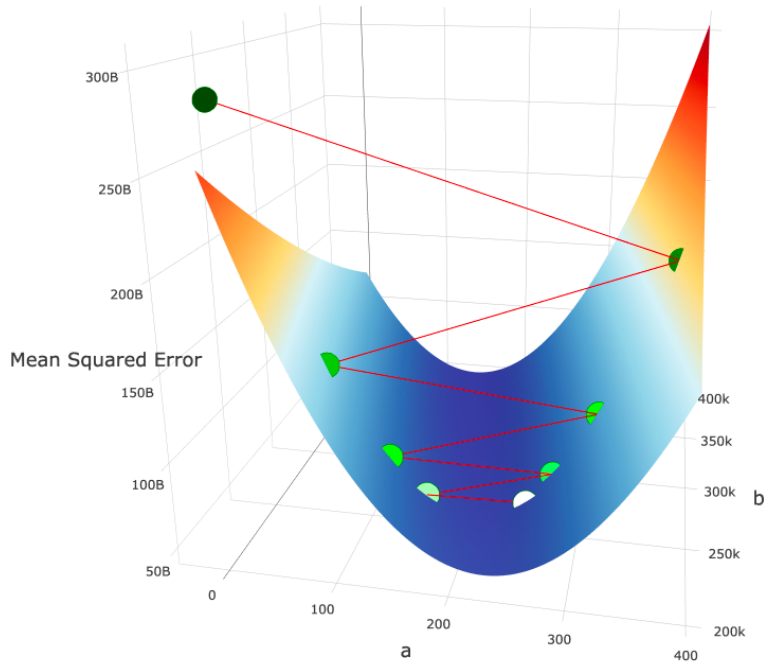
- Find least probable class: $c^* = \arg\min_c f(x)_c$
- Initialize $\delta_0 = 0$.
- Iterate

$$\delta_{n+1} = \delta_n - \eta \nabla_\delta L(c^*, f(x+\delta_n))$$

- Stop when prediction changes from original class i.e. $\arg\max_c f(x+\delta_n)_c \neq \arg\max_c f(x)_c$

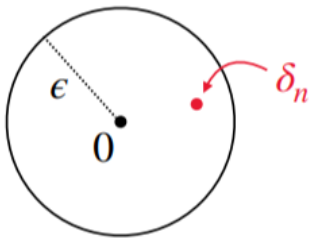
Iterative Gradient Methods

- **Problem**: Ensuring that perturbed image is not too different from original.
- *Need constraint* $\delta_n \ll 1$.
- **Quick Fix**: Set stepsize $\eta \ll 1$.
- **Question**: Is this a good solution?



Projected Gradient Methods

- **Idea:** Constrain the norm of the perturbations.
- $\hat{\delta} = \max_\delta L(c, f(x+\delta))$ s.t. $\|\hat{\delta}\| \leq \epsilon$.
- **Approach**: Use iterative gradient descent/ascent, but after each iteration we project δ_{n+1} into the ϵ-ball around the origin $B_\epsilon(0)$.
- The projection π_ϵ depends on the choice of **norm**. Common choices: $\ell_0, \ell_1, \ell_\infty$.
- As before, can use *ascent away from the original class* and/or *descent towards the least probable class*.



ϵ-ball $B_\epsilon(0)$ around 0

Projected Gradient Ascent

- Find original class prediction: $c^0 = \arg\min_c f(x)_c$
- Initialize $\delta_0 = 0$. Iterate until prediction changes from original class

$$\tilde{\delta}_{n+1} = \delta_n + \eta \nabla_\delta L(c^0, f_{\delta_n})$$
$$\delta_{n+1} = \pi_\epsilon[\tilde{\delta}_{n+1}]$$

Projecting perturbations: Selecting a norm

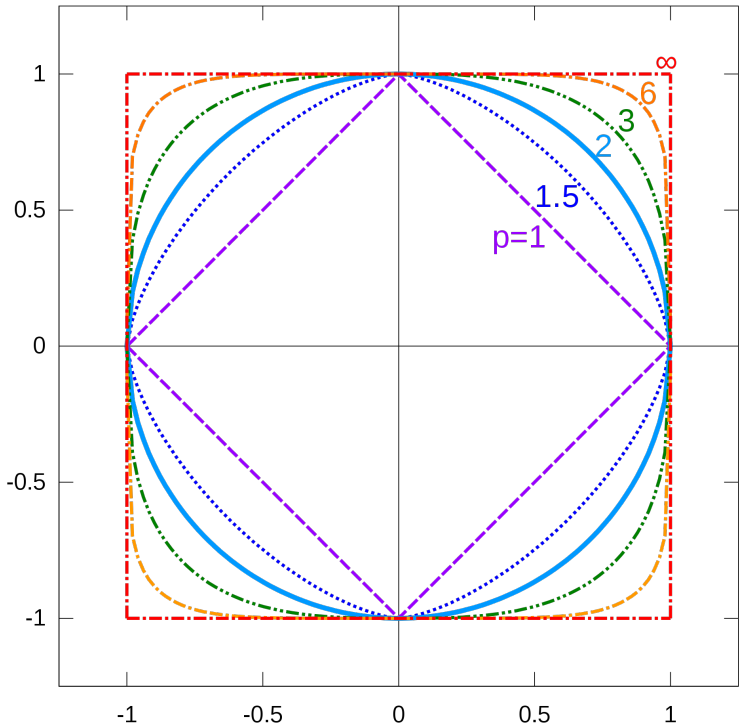
Recall; the ℓ_p norms are given by

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

For a given ℓ_p norm, we define

$$\pi_\epsilon(\hat{\delta}) = \begin{cases} \hat{\delta} \cdot \epsilon / \|\hat{\delta}\|_p, & \text{if } \|\hat{\delta}\|_p > \epsilon; \\ \hat{\delta} & \text{otherwise;} \end{cases}$$

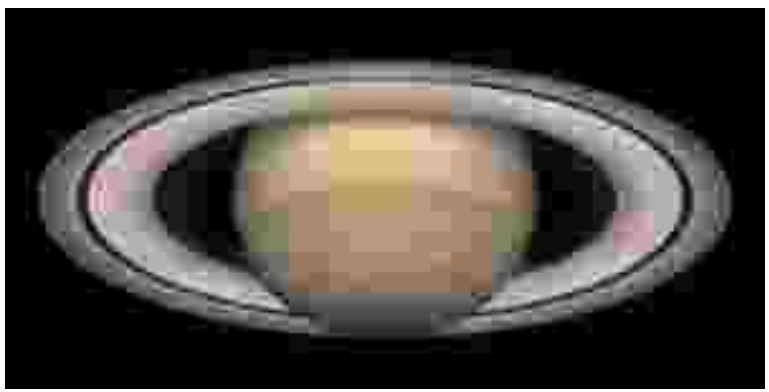
- Generally, we work with ℓ_2 (Euclidean).
- **Cheaper**: $\|x\|_1 = \sum_i |x_i|$.
- **Cheapest**: $\|x\|_\infty = \max_i |x_i|$.



Question: How does choice of norm change the behaviour of the attack?

Common Problems with Attacks

- **Invalid bounds**: $x \in [0, 1]^{C \times H \times W}$.
  Not necessarily the case for $x + \delta$.
- **Quantization**: 8-bit images stored with each channel $x_i \in \{0, 1, \ldots, 255\}$.
  Attacks may be less effective after quantization.
- **Compression**: Many datasets stored with lossy JPEG compression.
  Compression artifacts can destroy a carefully selected perturbation.



Murphy's Law: *"Anything that can go wrong, will go wrong"*.

Fast gradient sign

Problem: Finding perturbations with iterative gradient ascent/descent for many inputs can be expensive.

Idea: Only update δ once, using only the sign of the gradient.

Quick way to create many adv. examples, but each has a lower chance of fooling the model.

Fast gradient sign

- $\delta = \eta \cdot \text{sign}(\nabla_x L(c^0, f(x))$
  Adversarial image is then $\bar{x} = x + \delta$

- This simply adds a perturbation of magnitude in the direction of the gradient to each pixel value of the image
- Requires no iterations
  $\Rightarrow$ is much faster!

Iterative gradient sign

Iterative version of fast gradient sign.

NB: Both fast gradient sign and iterative gradient sign can be make into targeted attack methods.

Question: How would you do this?

Iterative gradient sign

- Find original class prediction: $c^0 = \arg\max_c f(x)_c$
- Initialize $\delta_0 = 0$.
- Iterate

$$\tilde{\delta}_{n+1} = \delta_n + \eta \cdot \text{sign}\big(\nabla_\delta L(c^0, f(x+\delta_n))\big)$$
$$\delta_{n+1} = \pi_\epsilon(\tilde{\delta}_{n+1})$$

  where η is some small stepsize.

- Stop when prediction changes from original class

Universal adversarial attacks

One perturbation to fool them all...

- So far, the adversarial perturbations δ are computed for one image x only
- Have to find one δ for every x
- **Universal adversarial attacks:** Find one δ that has a high change of causing misclassifications on *all* the input samples

Question: What are the consequences of this result?

Reflect: What does this imply about the loss landscape of neural networks?

Universal adversarial perturbation (simplified)

1. Initialize $\delta = 0$
2. For each sample $x_i \in X$:

   If $\arg\max_c f(x_i + \delta) = \arg\max_c f(x_i)$:

   (i) $\tilde{\delta}_i \leftarrow \arg\min ||v||_2$ s.t.

   $$\arg\max_c f(x_i + \delta + v) \neq \arg\max_c f(x_i)$$

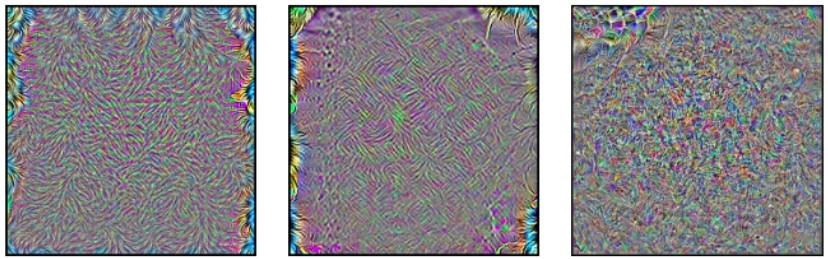
   (ii) $\delta \leftarrow \pi_\epsilon(\delta + \tilde{\delta}_i)$
3. Repeat step 2 until convergence

Remark: Check out this paper: https://arxiv.org/pdf/1610.08401.pdf

Universal adversarial attacks

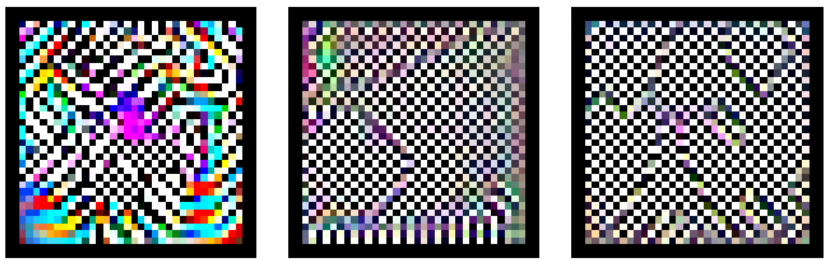
More results

- Hayes & Danezis (2018) introduced *Universal Adversarial Networks (UAN)* that generates Universal Adversarial Perturbations (UAPs) capable of fooling a target classifier
- Fig. 3: Examples of UAPs for different networks
- Success rates between 70% - 90%, depending on model



(a) VGG-19 (b) ResNet-152 (c) Inception-V3

Fig. 2: UAPs generated by a UAN for ImageNet.



(a) VGG-19 (b) ResNet-101 (c) DenseNet

Fig. 3: UAPs generated by a UAN for CIFAR-10.

https://arxiv.org/pdf/1708.05207.pdf

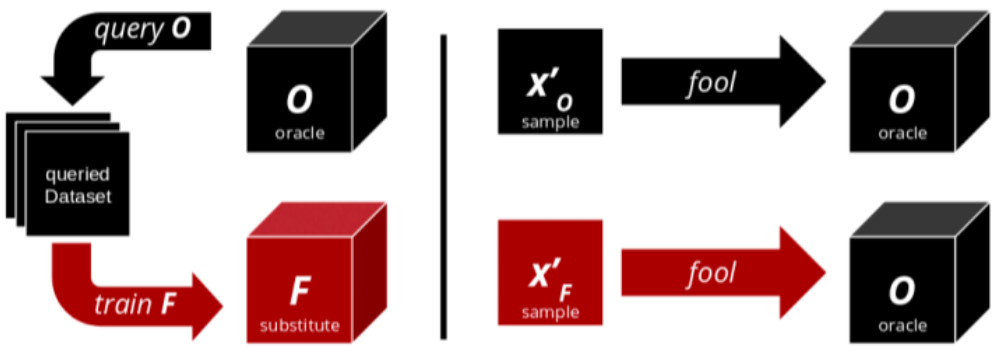
Black box attacks

A brief mention

We only have access to the output $\hat{y} = f(x)$ of the model. This means we can't compute gradients!

Two approaches:

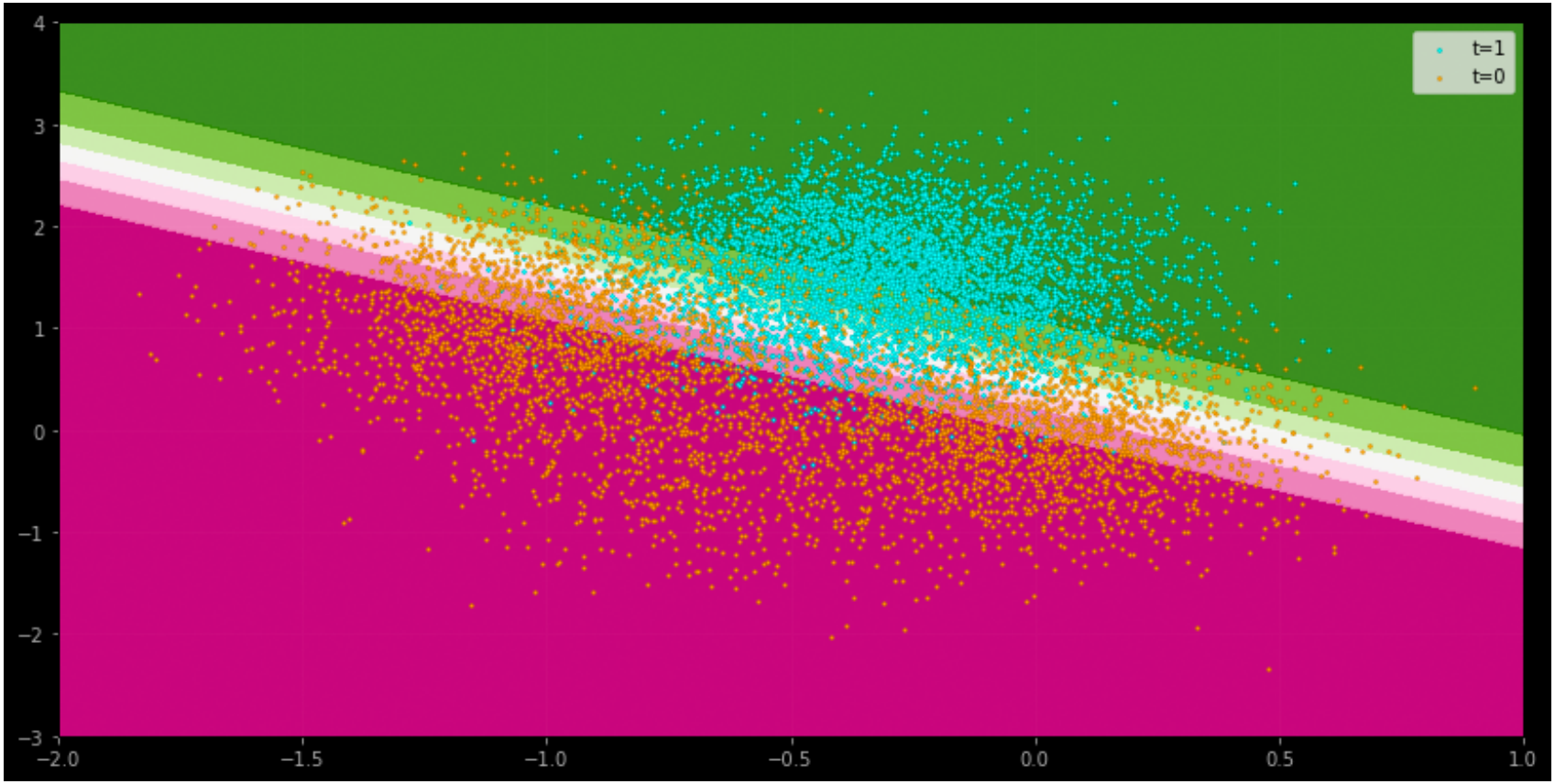
1) **Surrogate attacks:** Train a surrogate model $\hat{f} \approx f$ and attack the surrogate with a white box method. Assume the attack on $\hat{f}$ generalizes to f.



2) **Boundary attacks:** Starts with a random image and iteratively refines the adversarial example by moving it closer to the decision boundary.
   - **Importantly:** We want to end up on the wrong side of the boundary!
   - Without gradient, we instead use a hill-climbing approach.

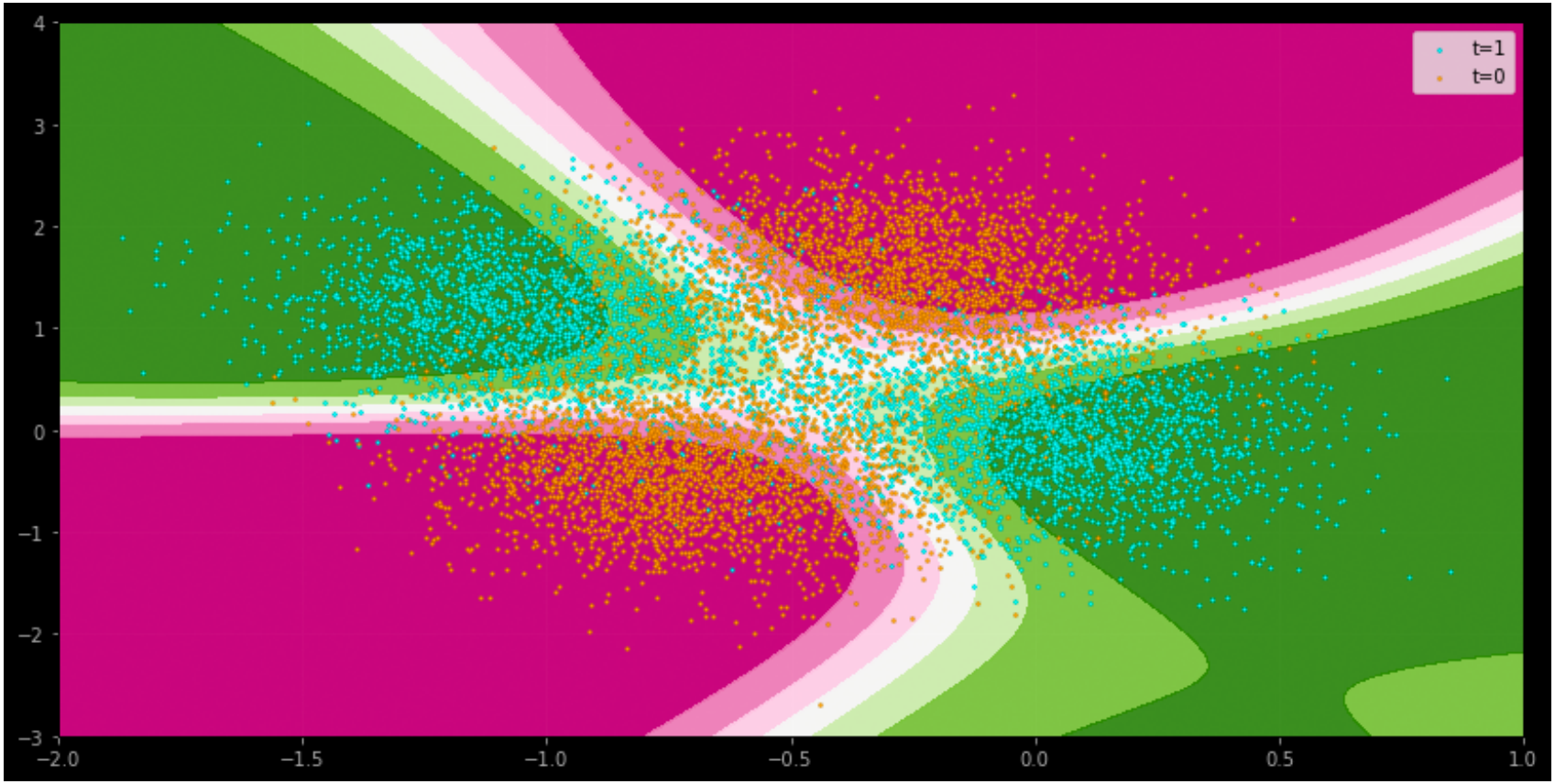
Decision Boundaries

Example: Logistic Regression, Linear



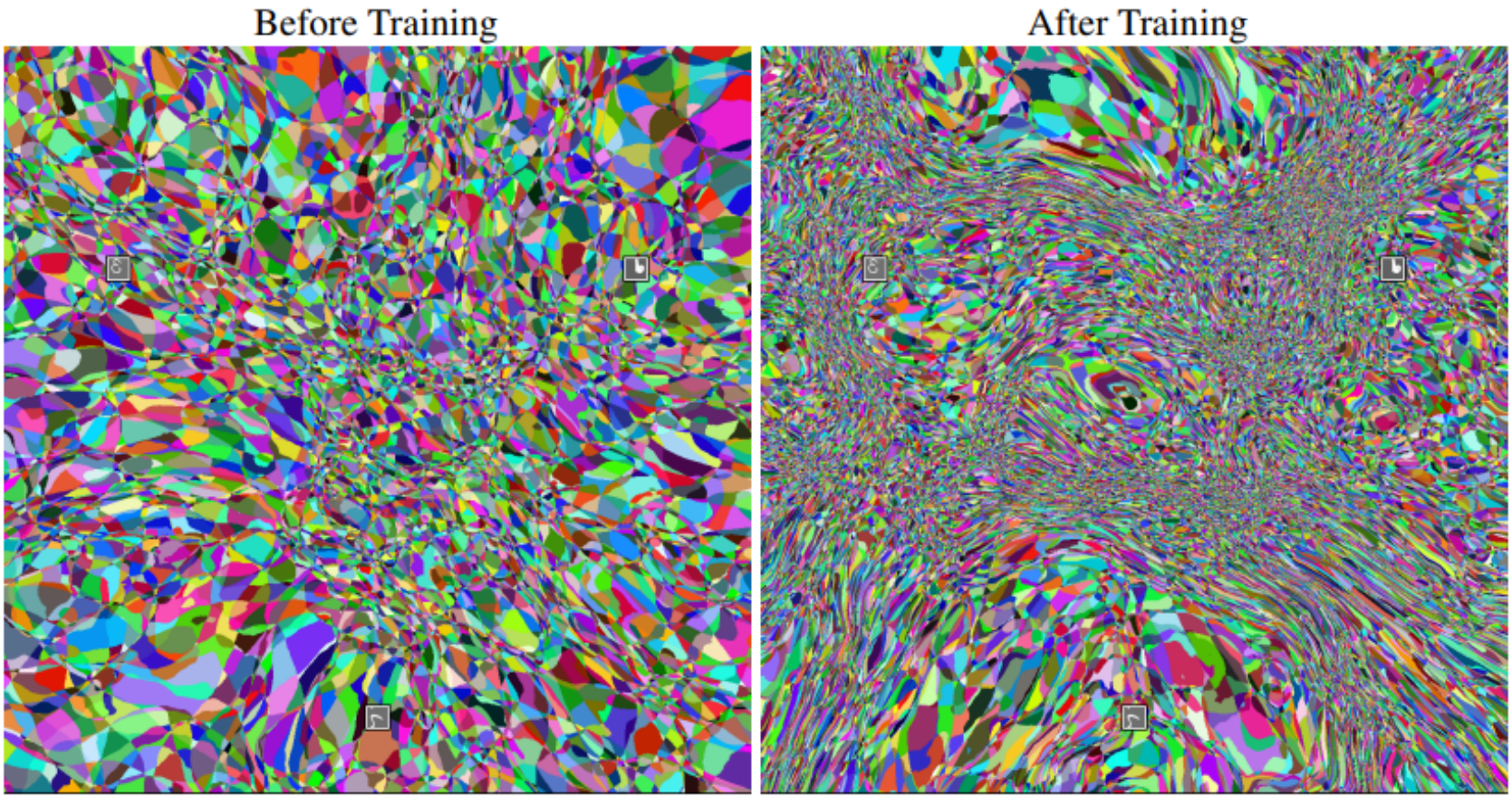
Decision Boundaries

Example: Multi-Layer Perceptron, Nonlinear



Decision Boundaries

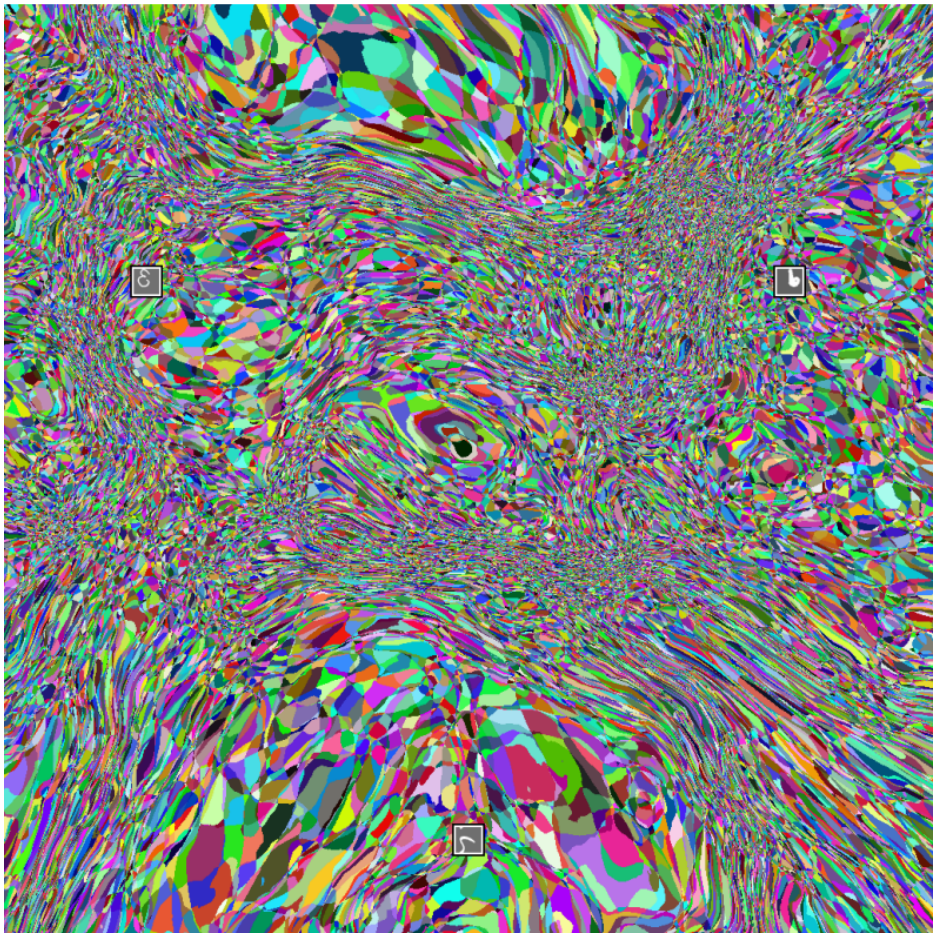
Example: Deep Neural Network



Complex decision boundaries

Why does this matter?

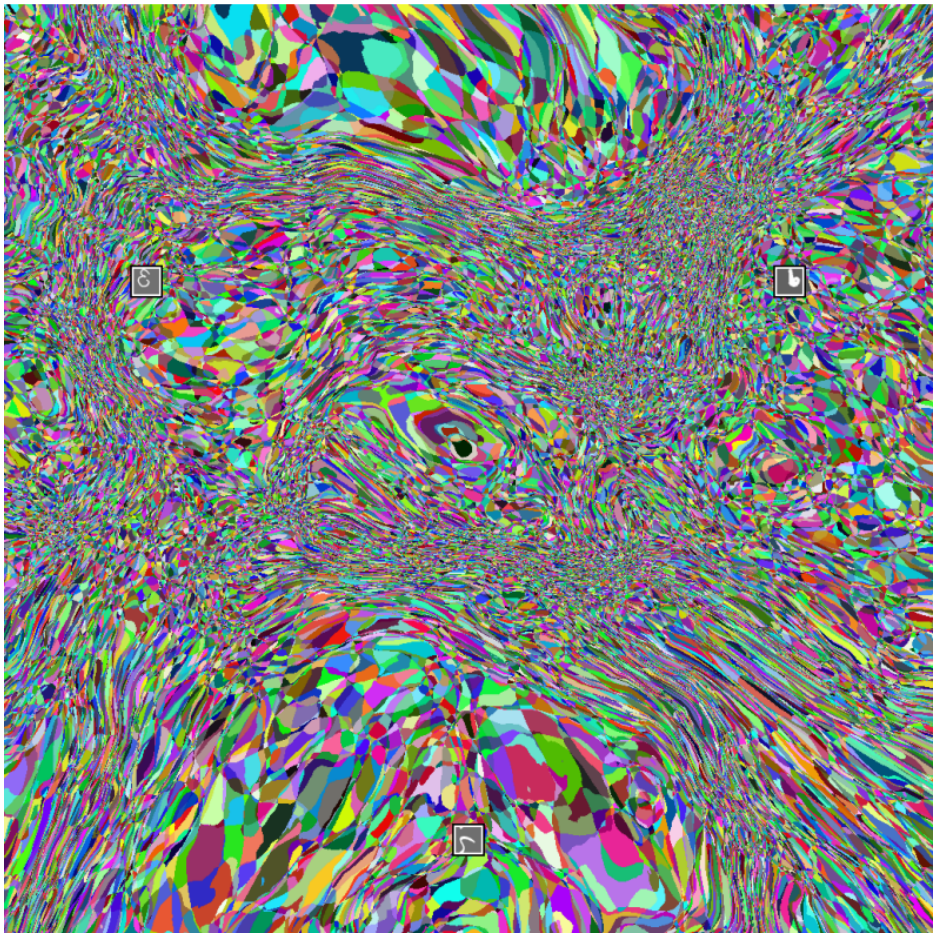
- **Classical view:** The samples are **dense** in the input space
- In statistical learning settings, we operate in a **much lower dimension** than in deep learning
- We cannot rely on such assumptions for sufficiently high dimensional data



Complex decision boundaries

Some intuition

- Even with huge amounts of data, some regions and dimensions of the image space will be sparse
- In these sparse regions, the decision boundaries can be more or less **arbitrary**
- Result: Perturbing x just a tiny bit can lead to very different outputs



Well-posedness

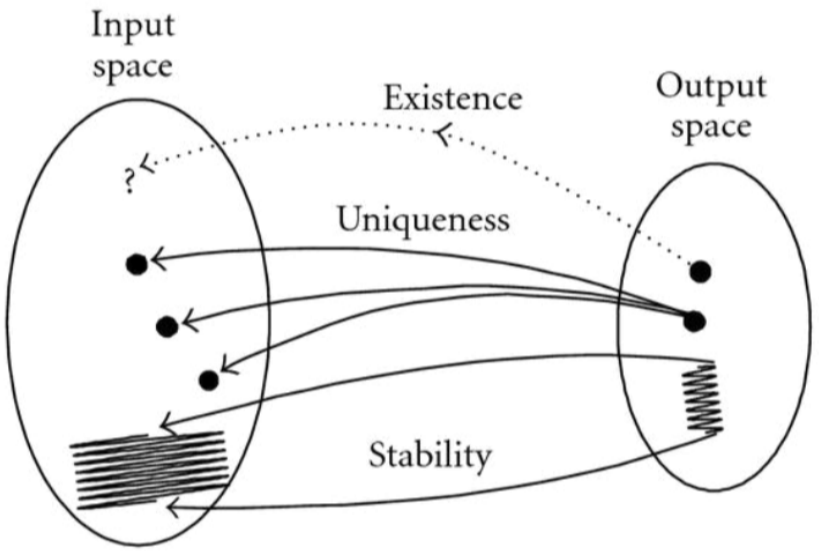
Motivating stability

Well-posedness is a term used to describe problems that are "nice" / "solvable"

A problem is **well-posed** if:

- A solution **exists**
- The solution is **unique**
- The solution changes **continuously** with the input.

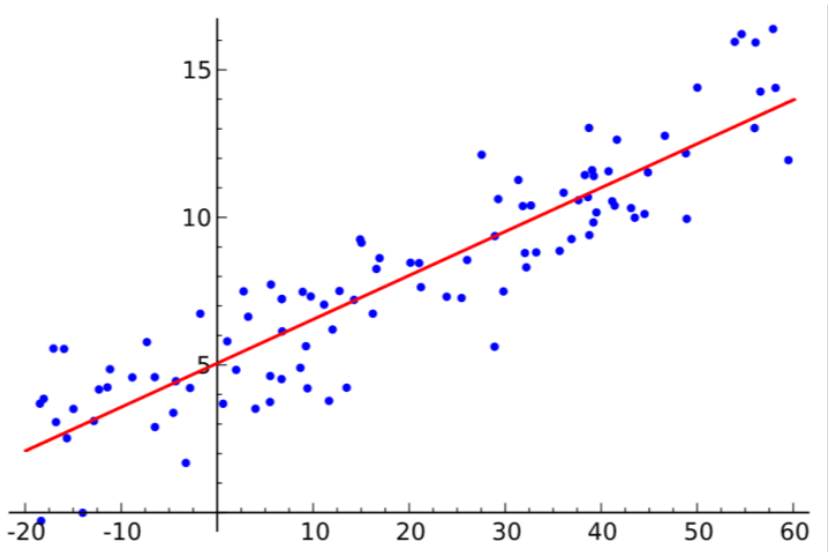
If any of these criteria do not hold, the problem is **ill-posed**



Well-posedness

Example: Are linear projections well-posed?

- Consider a simple linear model: $f(x_i) = Wx_i + b = y_i$ with $W \in \mathbb{R}^{m \times n}$
- If $m < n$ the model is ill-posed since multiple x_i can map to the same y_i. However, this is not a problem for classification or regression tasks.
- Importantly: f satisfies existance and stability (continous)
- **Convolutions** are special cases of linear maps $\Rightarrow$ They also have these properties.



Ill-conditioning

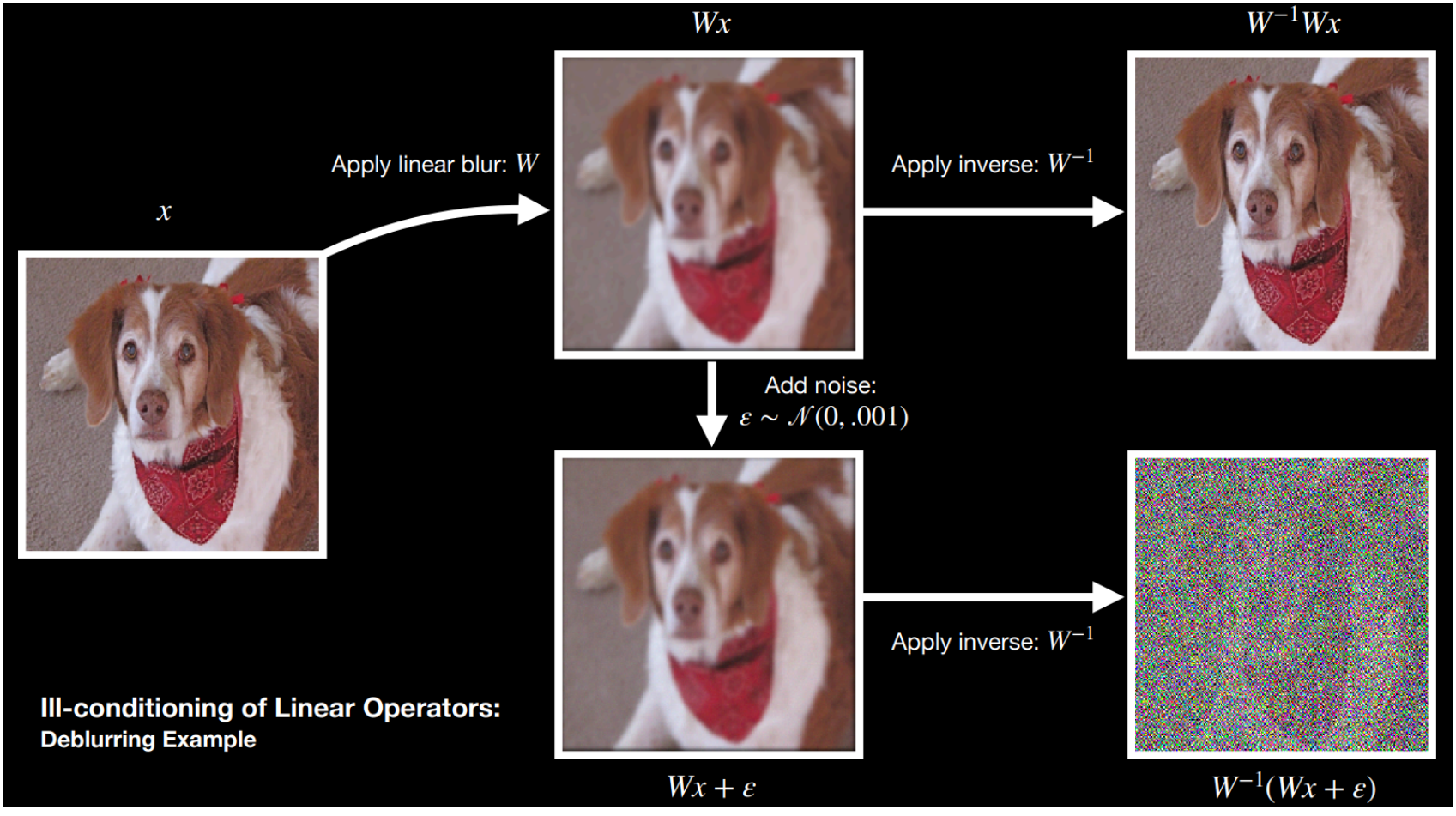
Stable, but badly behaved

- Even if continuous mappings are technically stable, they can exhibit bad behaviour
- Small changes in inputs can lead to very large changes in outputs
  $\Rightarrow$ **"Ill-conditioning"!**



Lipschitz continuity:
f is Lipschitz continuous if $||f(x) - f(x')||_2 \leq k||x - x'||_2$ for all inputs x, x', and we call k the Lipschitz constant of f.

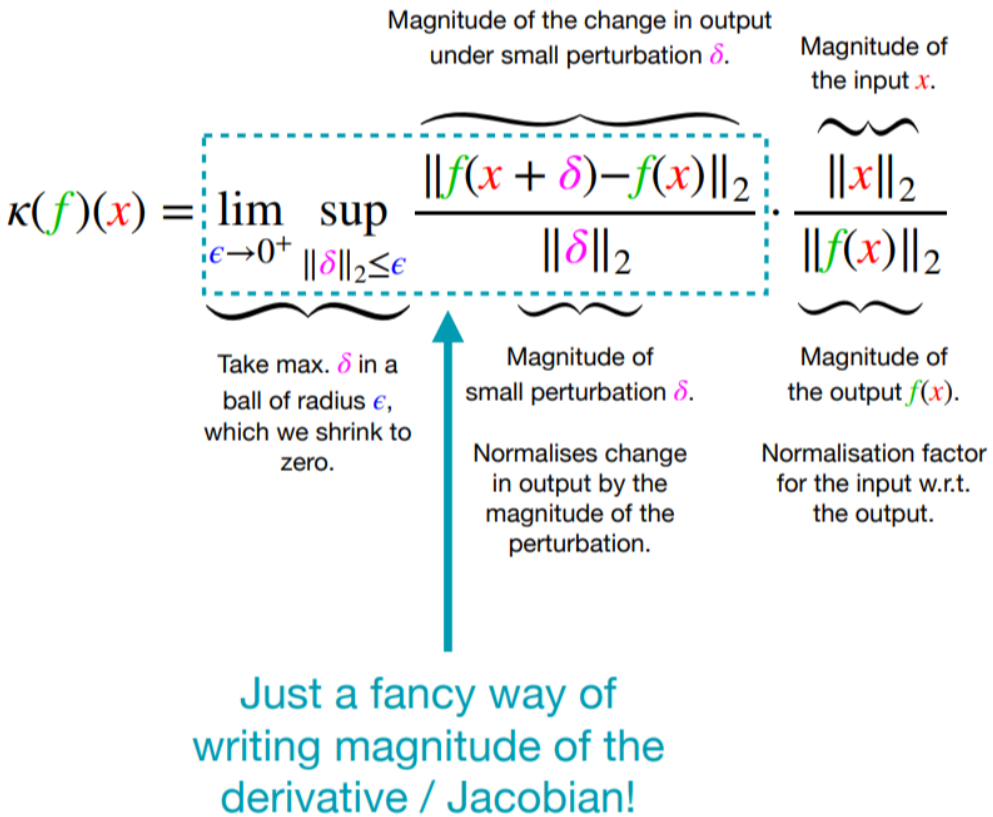
Ill-conditioning: Deblurring example



Condition numbers

Quantizing ill-conditioning

Relative condition number:



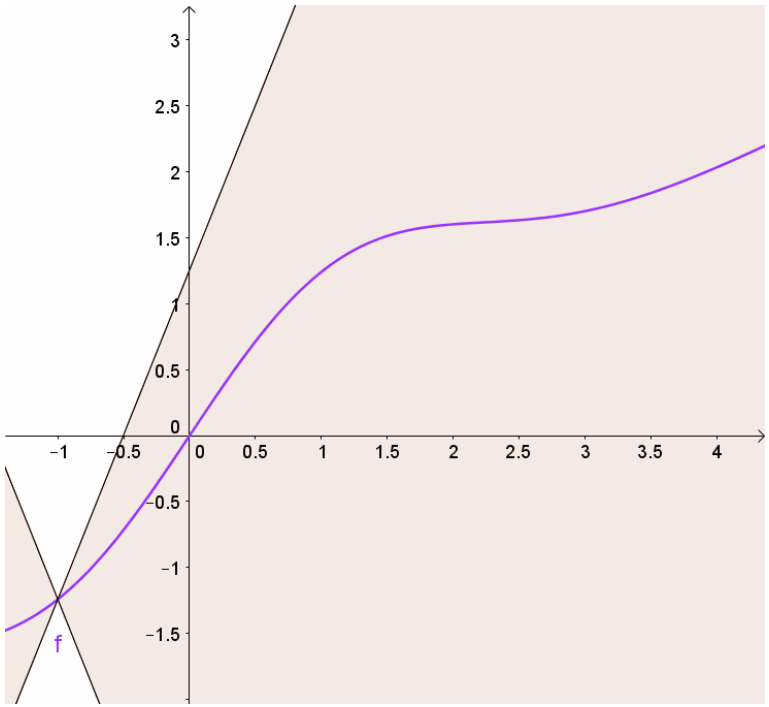
Tells us how sensitive a function is to a perturbation at a point x

Essentially: Normalize the magnitude of the derivative at x by the magnitude of the input w.r.t. the output.

Some intuition

Lipschitz? Condition number?

- **Lipschitz constant:** Maximal slope of a function. Sayns how "fast" the function can "jump".
- If our network has a smaller Lipschitz constant it is smoother and less sensitive to perturbations.
- **Relative condition number:** Bounded from above by k_f / k_{f-1} called the *absolute condition number*.



Matrix conditioning

Condition numbers for linear maps

For linear maps (matrices), the **absolute condition number*** is given by

$$\kappa(W) = ||W||_2||W^{-1}||_2 = \varsigma_1/\varsigma_n = k_W/k_{W^{-1}}$$

where ς_i denotes the ith singular value of W.

We can remedy ill-conditioned matrices by dampening the singular values

$\Rightarrow$ **Regularization!**



* For Euclidean spaces

ℓ_2 Regularization

For matrices

- Consider the **deblurring problem** from before: $Wx + \epsilon = y$. We want to find $\hat{x} \approx x$.
- If we assume $\epsilon \sim \mathcal{N}(0, \alpha^{-1} I)$, we can solve for

$$\min_x \frac{1}{2} \left[||Wx - y||_2^2 + \alpha ||x||_2^2 \right]$$

- Can solve for this using tricks from linear algebra, which shows that ℓ_2 regularization gives dampening of the singular values. Calculation $\rightarrow$
- **NB:** This calculation is **not** part of the curriculum (just for interested students)

Calculate the gradient of the objective w.r.t. x and set it to 0. Then

$$W^T(Wx - y) + \alpha x = 0$$
$$W^T W x + \alpha x = W^T y$$
$$(W^T W + \alpha I)x = W^T y$$
$$x = (W^T W + \alpha I)^{-1} W^T y$$

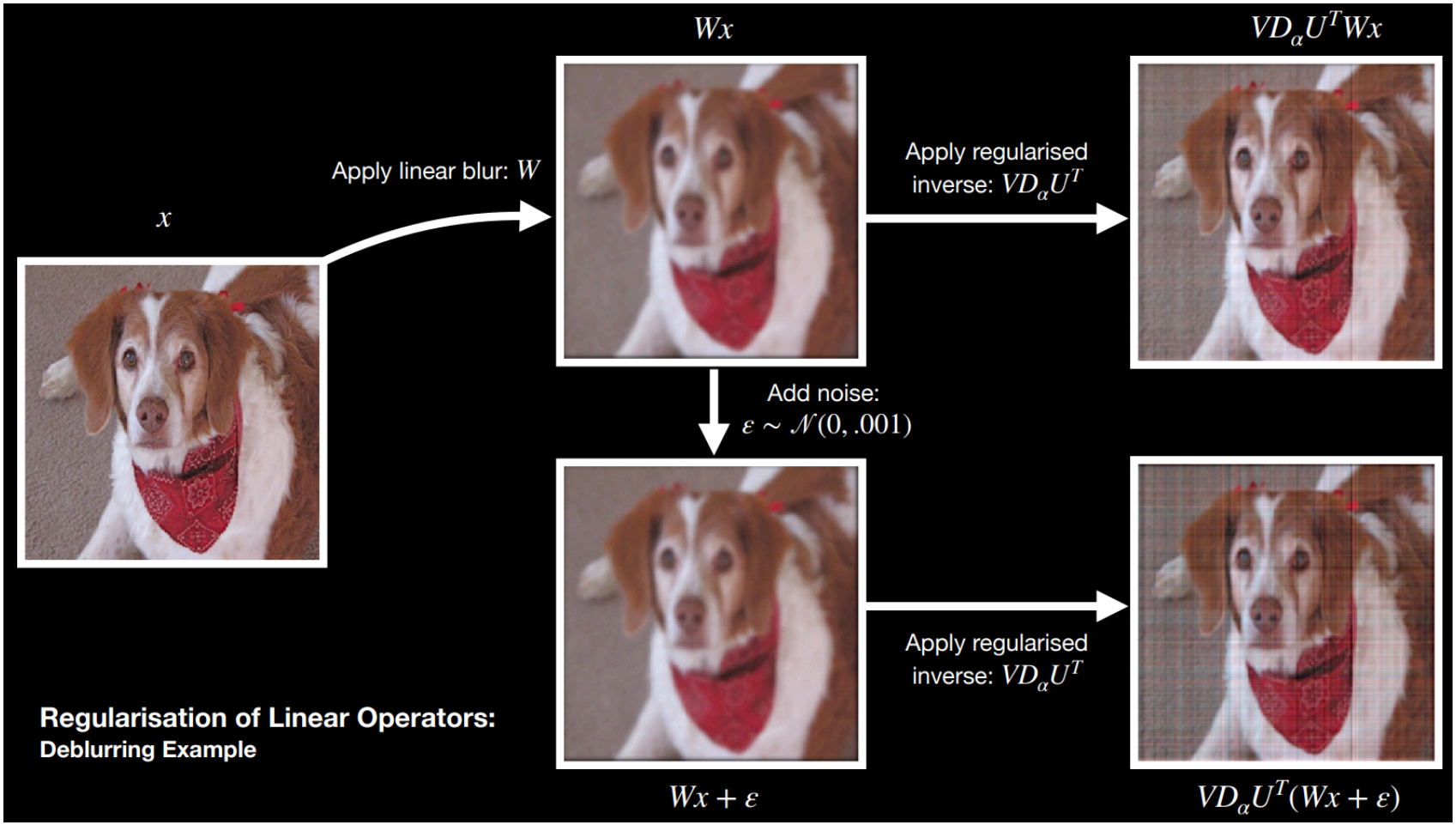
apply singular value decomposition:

$$x = (V \Sigma U^T U \Sigma V^T + \alpha I)^{-1} V \Sigma U^T y$$
$$x = (V \Sigma^2 V^T + \alpha V I V^T)^{-1} V \Sigma U^T y$$
$$x = (V [\Sigma^2 + \alpha I] V^T)^{-1} V \Sigma U^T y$$
$$x = V \Sigma (\Sigma^2 + \alpha I)^{-1} U^T y$$
$$x = V D_\alpha U^T y$$

where $D_\alpha = \text{diag}([\varsigma_i / (\varsigma_i^2 + \alpha)]_{i=1}^n)$

ℓ_2 Regularization

Deblurring example



ℓ_2 Regularization

For neural networks

- ℓ_2 regularization is straightforward in SGD
- Use a modified update rule:

$$W_{t+1} = (1 - \lambda)W_t - \eta \nabla_W \mathcal{L}(W_t)$$

- λ dampens the weight W_t by **weight decay**

Terms:

$W_t :=$ Current weight at step t

$\eta :=$ Learning rate

$\mathcal{L} :=$ Loss function

How to plug this into SGD:

Note that $\nabla_W \mathcal{L}_{\text{reg}}(W_t) = \nabla_W \mathcal{L}(W_t) + \alpha W_t$. Then

$$\begin{aligned} W_{t+1} &= W_t - \eta \nabla_W \mathcal{L}_{\text{reg}}(W_t) = W_t - \eta(\nabla_W \mathcal{L}(W_t) + \alpha W_t) \\ &= (1 - \eta\alpha)W_t - \eta \nabla_W \mathcal{L}(W_t) \end{aligned}$$

Setting $\lambda = \eta\alpha$ gives the weight decay update

ℓ_1 Regulatization

Also known as Lasso

- Another common method: ℓ_1 regularization (Lasso)
- Increases sparsity $\rightarrow$ setting many of the weights to zero
- Reduces the number of effective parameters in the model and prevents over-fitting
- **Con:** Requires non-convex optimization, so not as straightforward to implement for NNs as ℓ_2 regularization

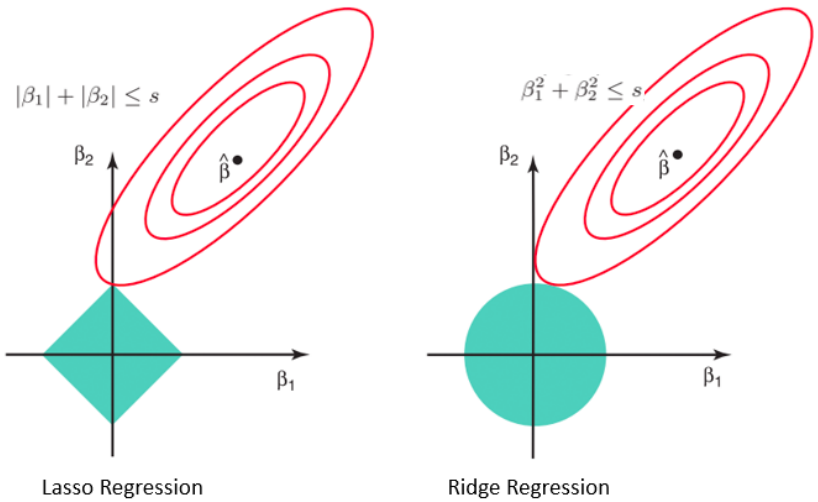
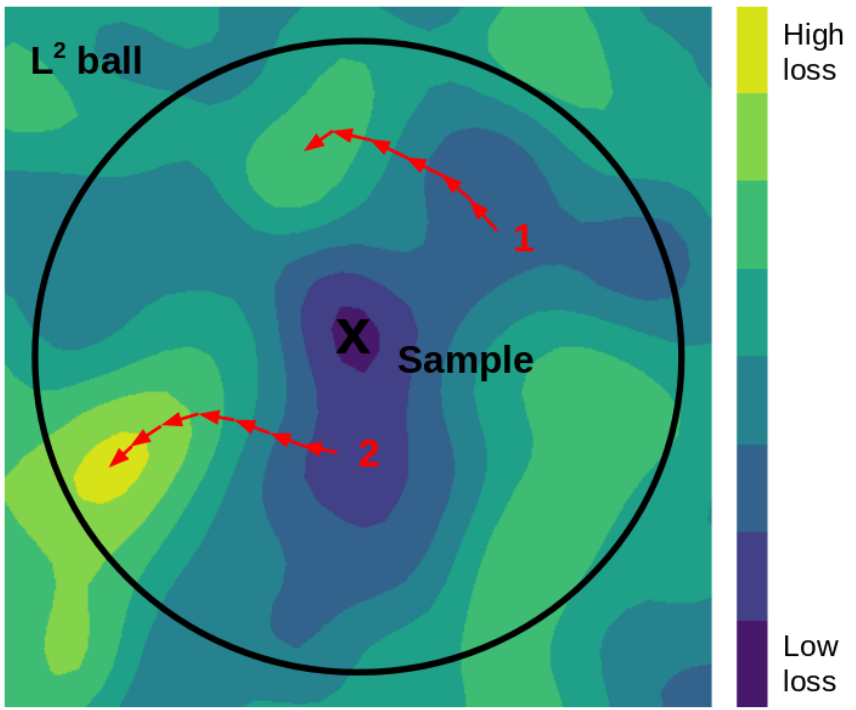


Image credit: Elements of Statistical Learning (Hastie, Tibshirani, Friedman, 2001)

Adversarial perturbations vs. stochastic noise

- **Stochastic perturbation:** Select random element in a ball centred on input according to prob.distribution.
- **Adversarial perturbation:** Select perturbation in the ball which maximally displaces the output.
- Adversatial attacks are **worst case** perturbations, often close to the perimeter of the ball

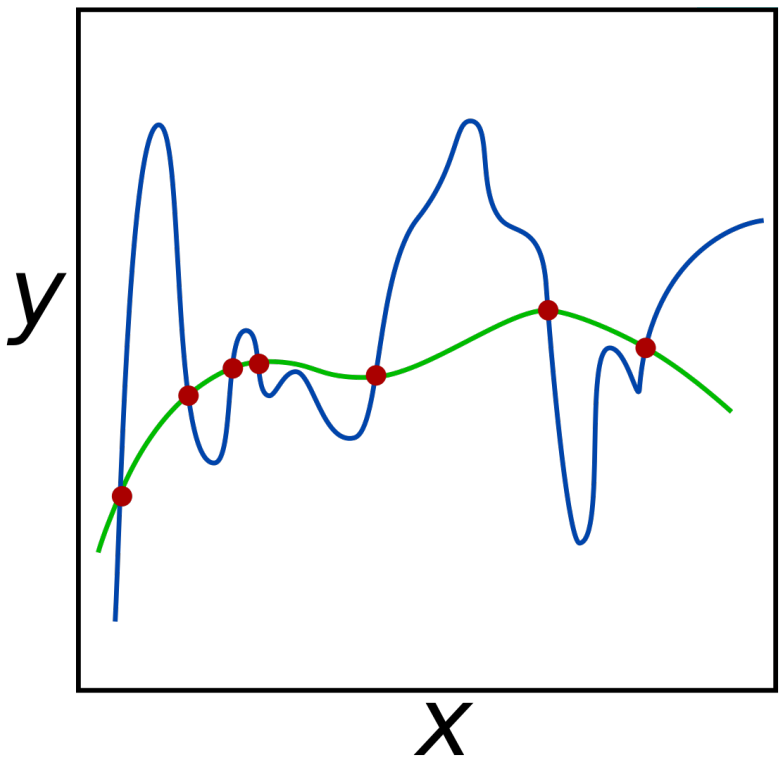


How do we defend against adversarial perturbations?

The first line of defence

Reducing condition numbers helps adversarial robustness. Some common tricks:

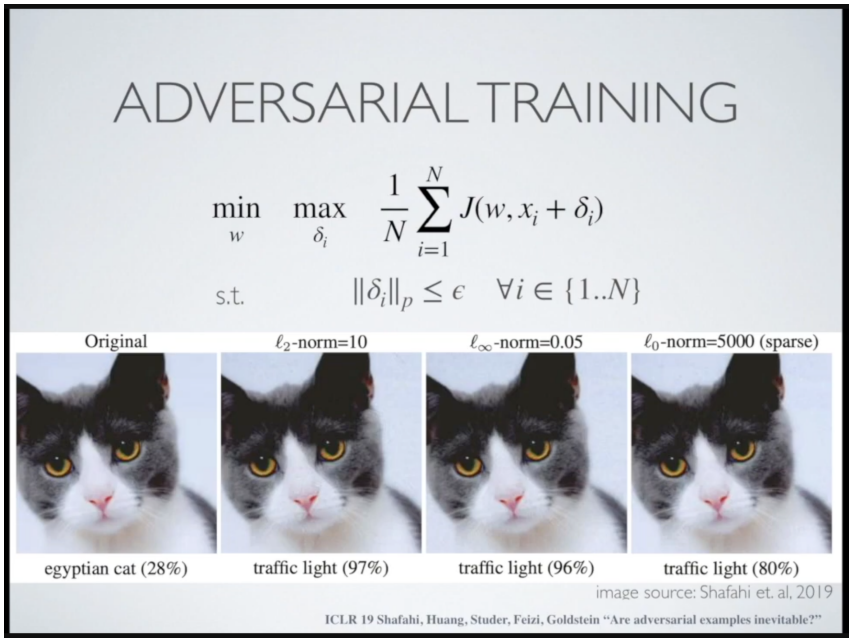
- **Weight decay and dropout:** Essential for general robustness
- **Data augmentation:** Robustness to transformations while augmenting dataset size
- **Label smoothing:** Smoothing the loss landscape
- Consider using **smooth activations** e.g. ELU, CELU, GELU



Alternate defences: Adversarial training

Simple idea: Train your network on universal adversarial perturbations.

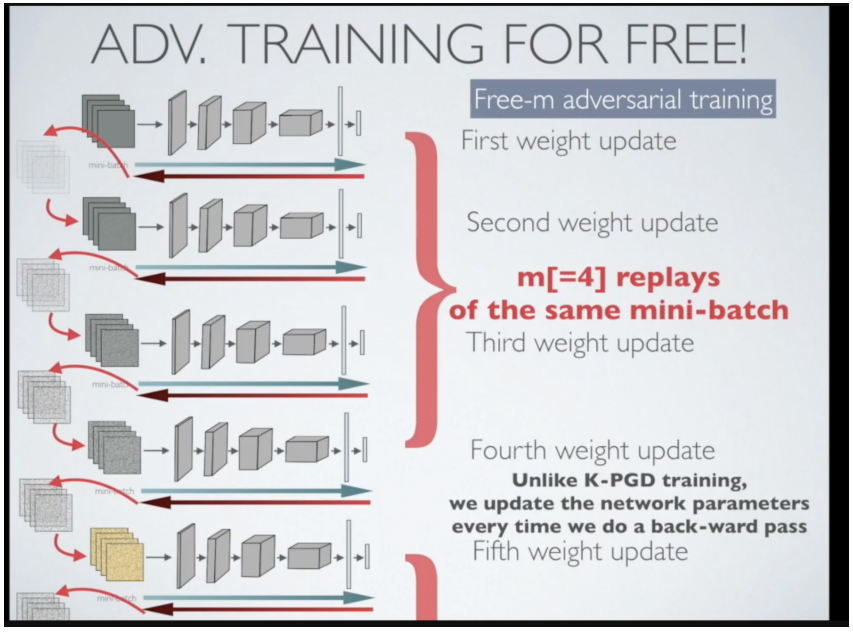
1. Find a universal adversarial perturbation δ with Projected Gradient Descent (PGD) using K steps without backward pass
2. Perform standard forward and backward pass of your network using δ



In practice: Is expensive and reduces predictive power

Alternate defences: Adversarial training (for free?)

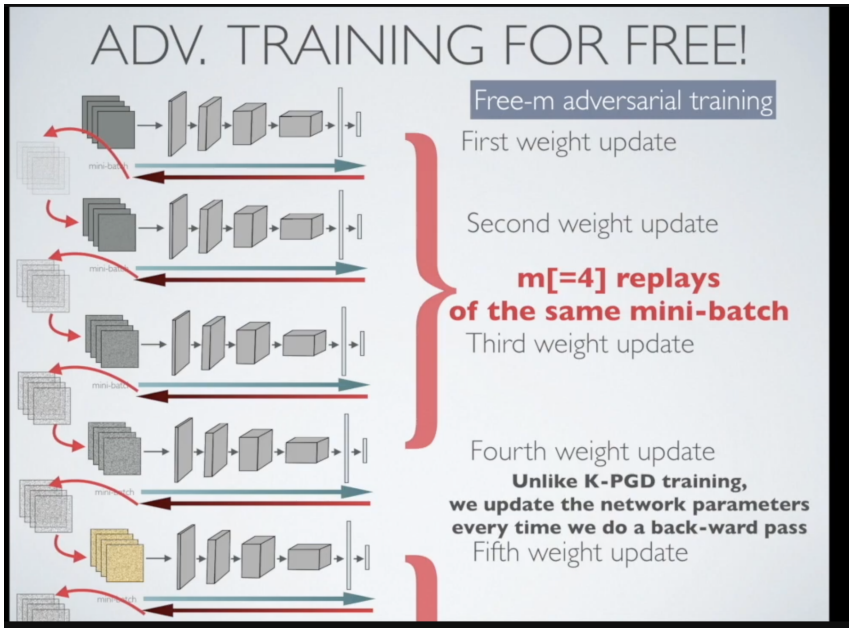
- Shafahi et al. (2019) proposes concurrent estimation by learning and updating a global perturbation, using m "replay steps" over the same batch
- **Pros:** Backward passes are now performed for every iteration $\Rightarrow$ More effective
- **Cons:** Potentially harms the stochasticity of the gradients and reduces predictive power. *Certainly not "free"!*



Adversarial training for free!

Alternate defences: Adversarial training (for free?)

- Shafahi et al. (2019) proposes concurrent estimation by learning and updating a global perturbation, using m "replay steps" over the same batch
- **Pros:** Backward passes are now performed for every iteration $\Rightarrow$ More effective
- **Cons:** Potentially harms the stochasticity of the gradients and reduces predictive power. *Certainly not "free"!*



Adversarial training for ~~free!~~

cheaper?

Alternate defences: Quantization in preprocessing

Idea: Reduce damaging noise components in preprocessing.

- **Quantization:** Reducing the bit representation of the image from 8-bit to 4-bit
- **DCT/JPEG Compression:** Simple and effective, encode limited number of significant frequencies



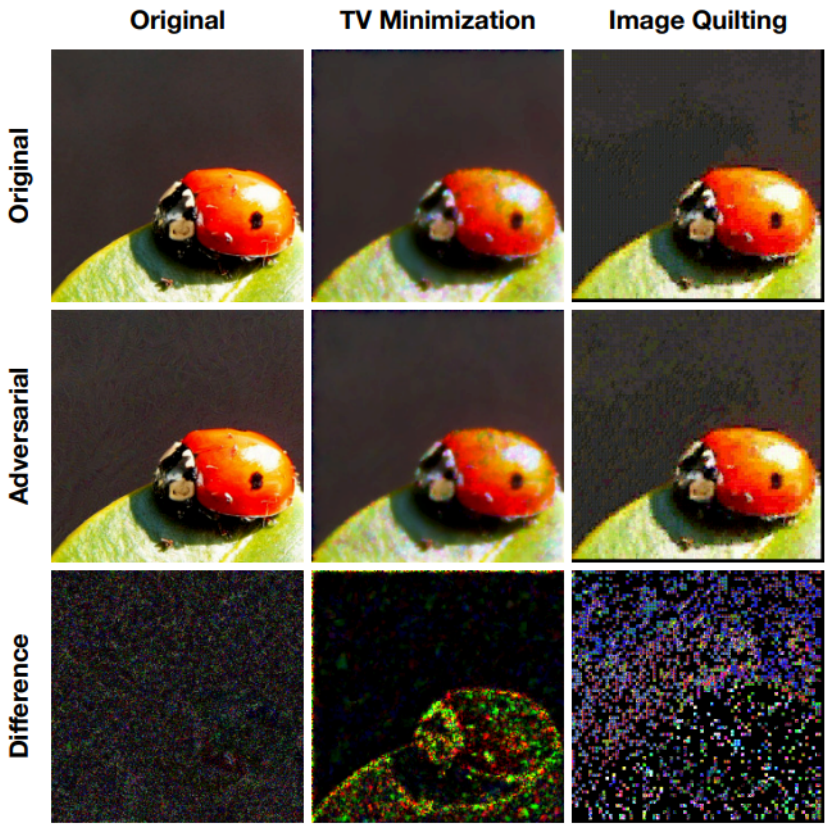
Utilizing common problems in attacks as defence!

Alternate defences: Denoising in preprocessing

Guo et al. (2018) proposes two alternate methods:

- **Total variation denoising:** Penalize normed difference between adjacent pixels.
- **Quilting:** Learn a codebook of clean patches, and project image patches to closest clean patch

Cons: While effective, all these methods sacrifice image quality.



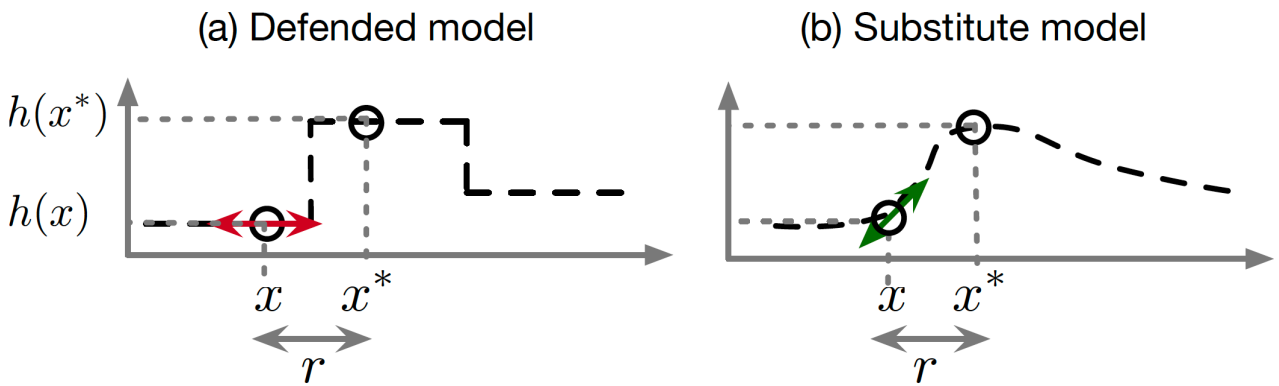
Alternate defences: General gradient masking

Preprocessing defences prevents low magnitude noise components from effectively propagating to the input.

We sometimes refer to such techniques as **gradient masking**.

Other gradient masking techniques:

- Non-differentiable layers
- Stochastic layers
- Defensive distillation: teacher-student networks with label smoothing.



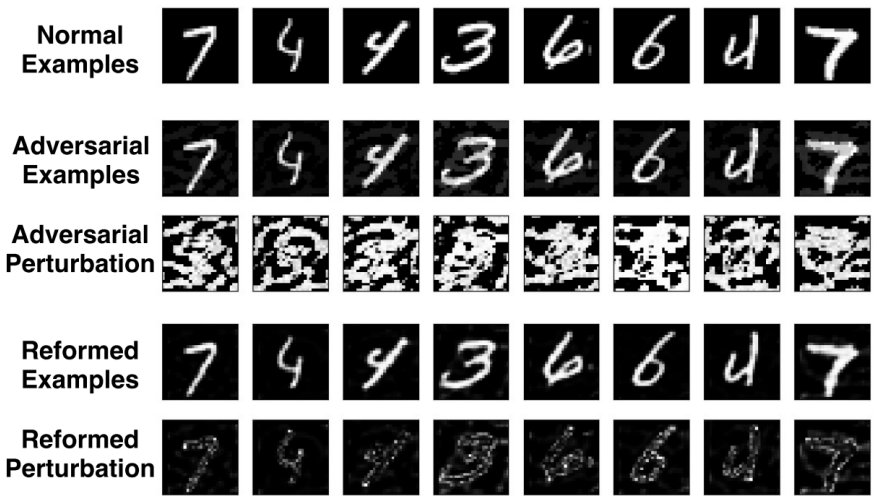
Alternate defences: Ensemble defence

Idea: It is harder to find attacks that apply to multiple models, especially if they are complimentary.

Leveraged in **ensemble defence**: Combined ensembles of models that work together.

MagNet: Model agnostic. Uses separate detection network to detect attacks and "reform" them before passing to final model.

Cons: Naturally very expensive



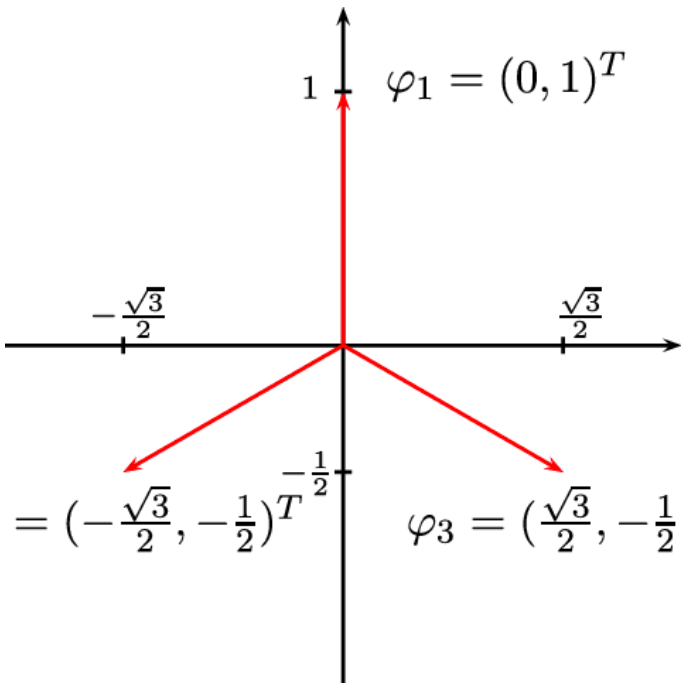
Alternate defences: Other regularization methods

Parseval Networks: Encourages weights to be approx. orthonormal by adding $\alpha||W^T W - I||_2$ to loss term.

Spectral Normalization: Different methods designed to normalise the Lipschitz const. of weights such that $k \approx 1$

TRADES: Minimise the distance (KL divergence) between predictions made from adversarial distribution and data distribution

Cons: Parseval reg. and Spectral norm. needs to be applied to each weight individually. TRADES need to estimate the distribution of adversarial outputs.



Alternate defences: Certified defences

Certified defences are regularisation methods that gives guaranteed bounds of robustness.

- **Randomized Smoothing:** Train network to output the mean of outputs of stochastically perturbed inputs
- **SmoothAdv:** Randomized smoothing, but using adversarial perturbations

Pro: Theoretically sound.

Con: Still expensive

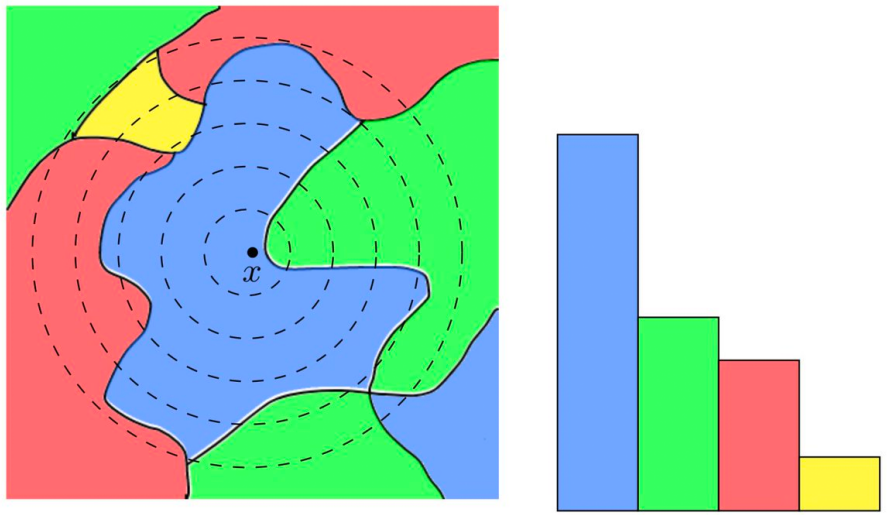
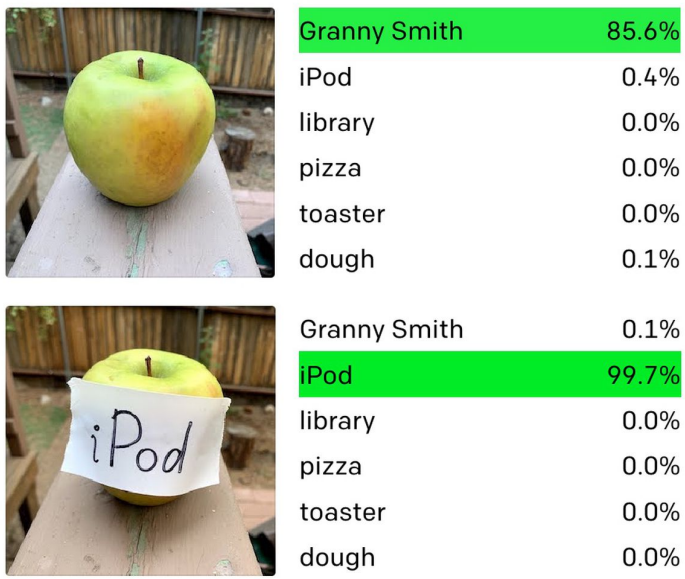


Figure 1. Illustration of randomized smoothing. **Left**: the decision regions of the base classifier f are drawn in different colors. The dotted lines are the level sets of the distribution $\mathcal{N}(x, \sigma^2 I)$. **Right**: the distribution $f(\mathcal{N}(x, \sigma^2 I))$. Here, $g(x)$ is "blue."

Adversarial defence strategies: Summary

- Classic regularization, quantization, and compression are cost effective, but sacrifices image quality
- More involved methods are expensive
- **Be aware:** few to no methods can withstand adaptive, persistent attacks
- You should try to have a clear understanding of the taxonomy of defences and how they work.



"Typographical attacks": https://openai.com/research/multimodal-neurons