Adversarial Attacks

Lecture #11 - IN3310/4310

Marius Aasan, Martine Tan - 13.04.2023

Introduction v2.0

These guys again?

- **Marius Aasan**
  - PhD Image Processing - DSB@IFI + SFI Visual Intelligence
  - Fields: Self Supervised Learning, Vision Transformers and Hierarchical GNNs
- **Martine Tan**
  - PhD Image Processing - DSB@IFI
  - Fields: Self Supervised Learning, Probabilistic Learning, Hierarchical Modelling
- Lecturing two units in IN3310/4310.





Introduction

After this lecture you will be able to:

1. Explain the concept of robustness and analyse this property in computer vision models.
2. Contrast and compare adversarial robustness to classic robustness.
3. Identify and enumerate different forms of adversarial attacks.
4. Explain and implement common versions of adversarial attacks.
5. Justify and apply defence strategies for adversarial attacks.
6. Reflect on the effectiveness of classical regularisation in context of adversarial perturbations.

Plan for Today

Part 1.

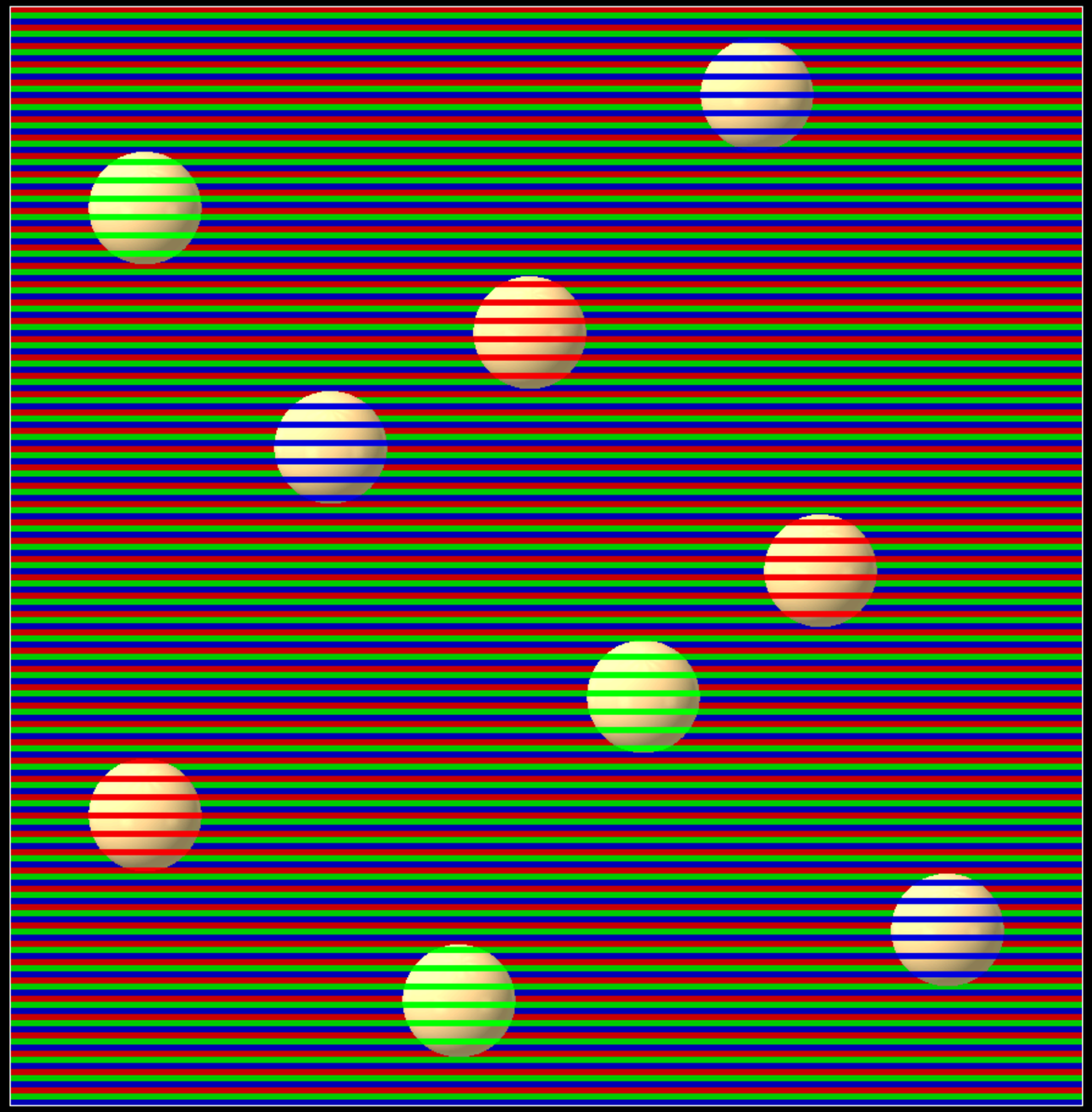
- **How do we fool humans?**
- **How do we fool computers?**
- **Taxonomy of attacks:**
  - White box / black box
  - Targeted / untargeted
- **White box attacks:**
  - Iterative gradient.
  - Projected gradient descent
  - Fast gradient sign method
- **Black box attacks**
- **Break (15 minutes)**

Part 2.

- **Decision boundaries**
- **The Manifold hypothesis**
- **Ill-posed / Ill-conditioned problems**
- **Classic Regularisation**
- **Specific Defense strategies:**
  - Adversarial training
  - Preprocessing / Gradient masking
  - Ensemble defences
  - Regularisation
- **Summary and architectural particularities**
- **Final Quiz (10 minutes)**

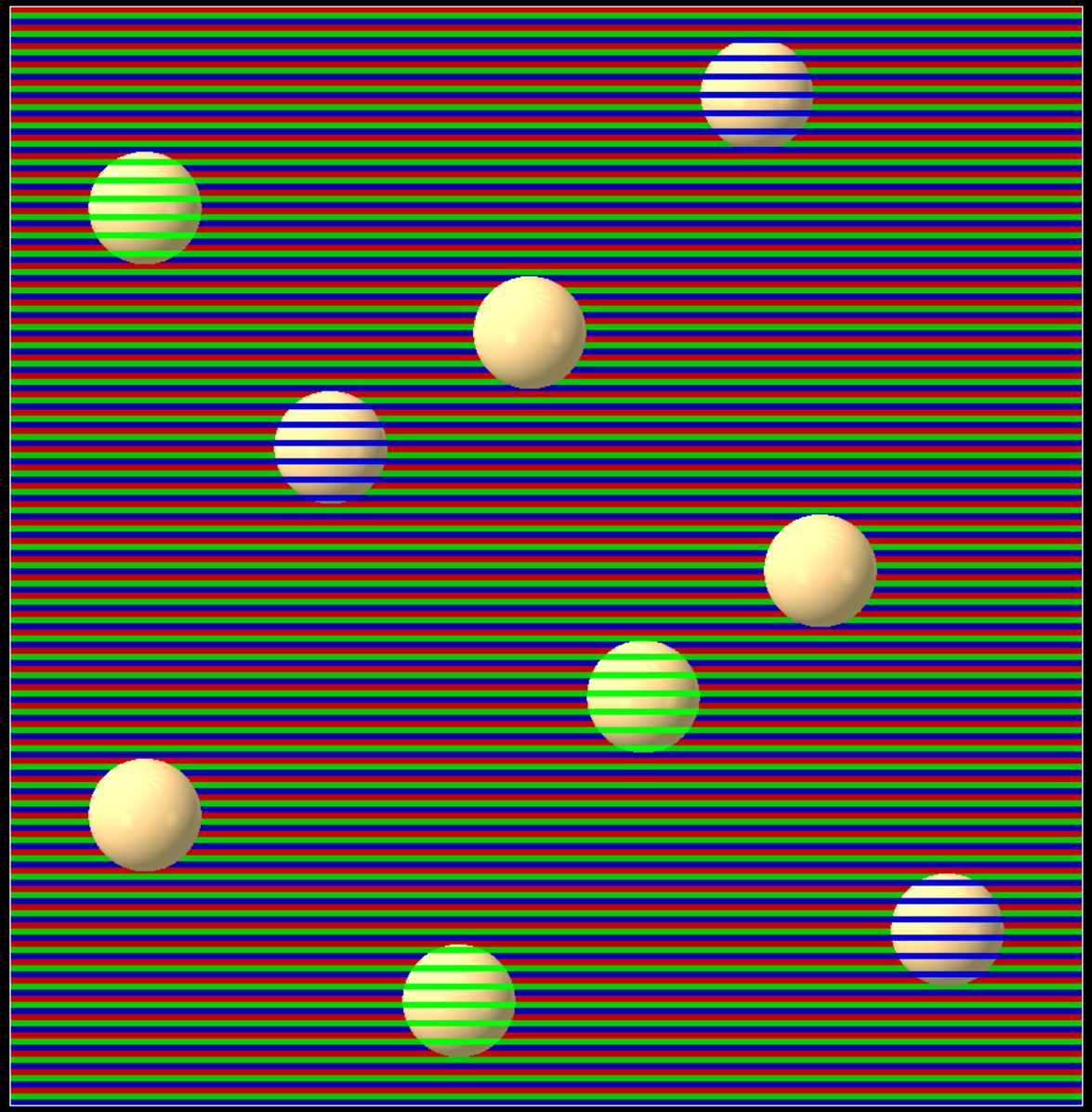
Example: Optical illusion

How to fool humans?



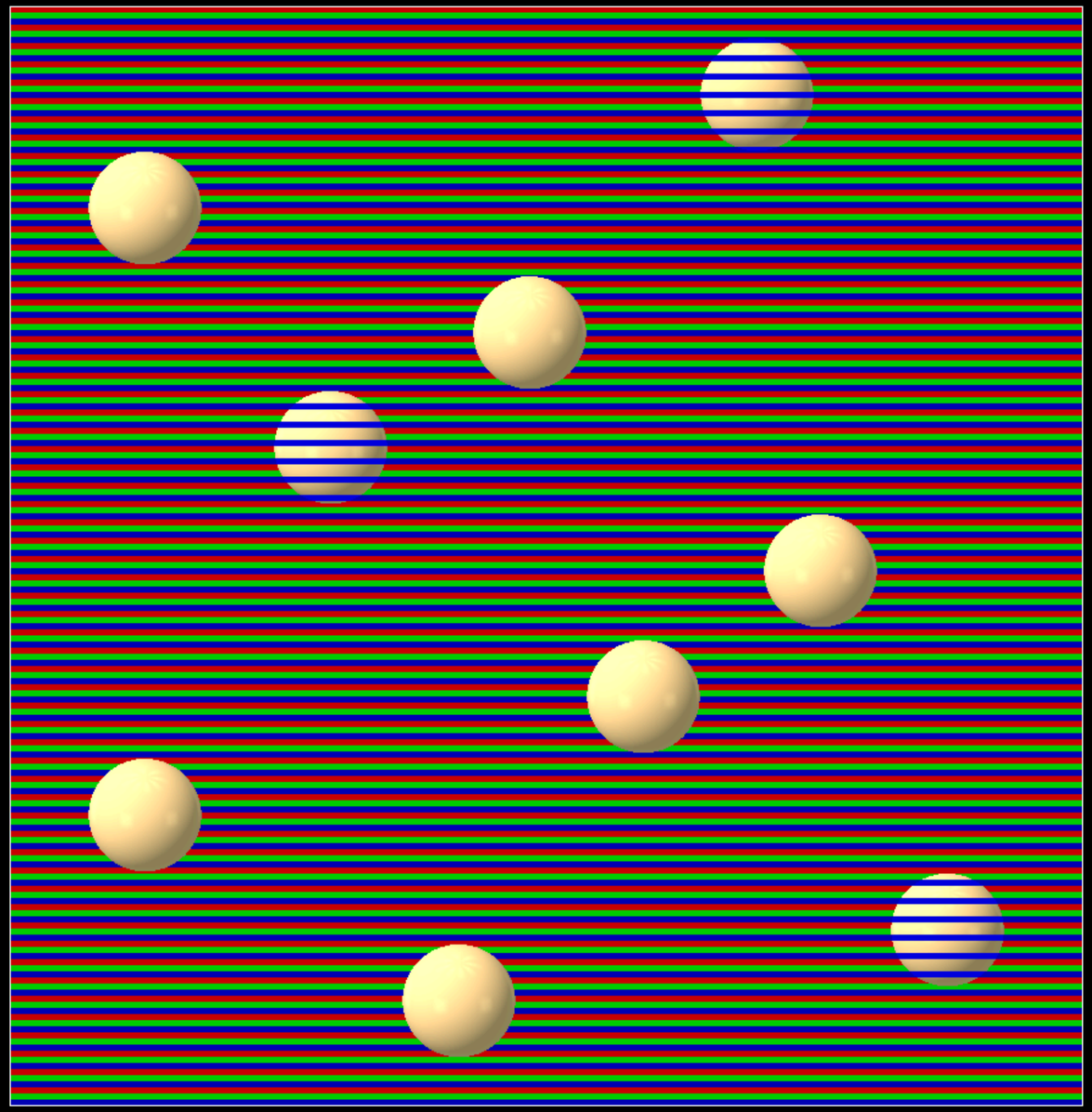
Example: Optical illusion

How to fool humans?



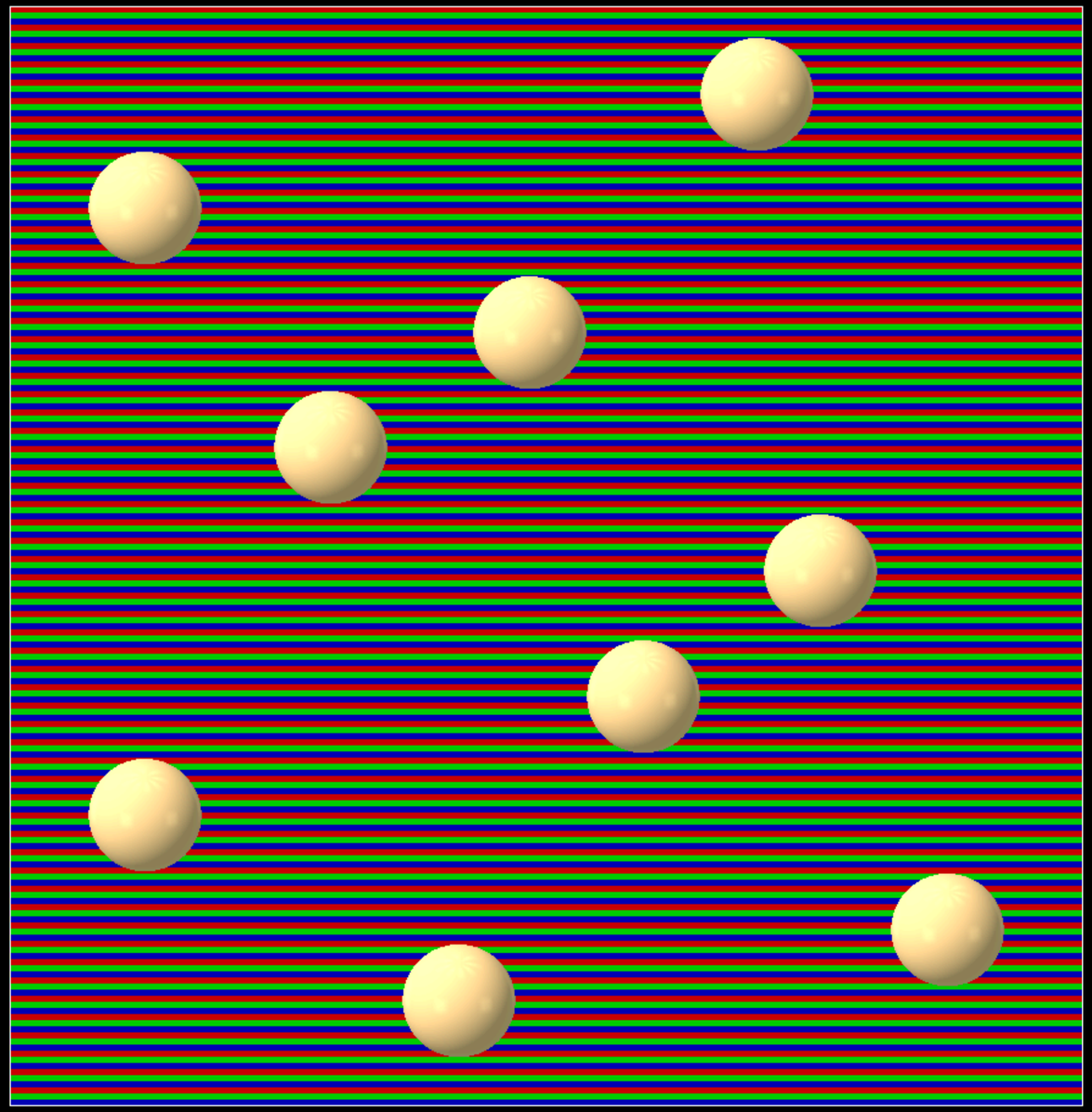
Example: Optical illusion

How to fool humans?



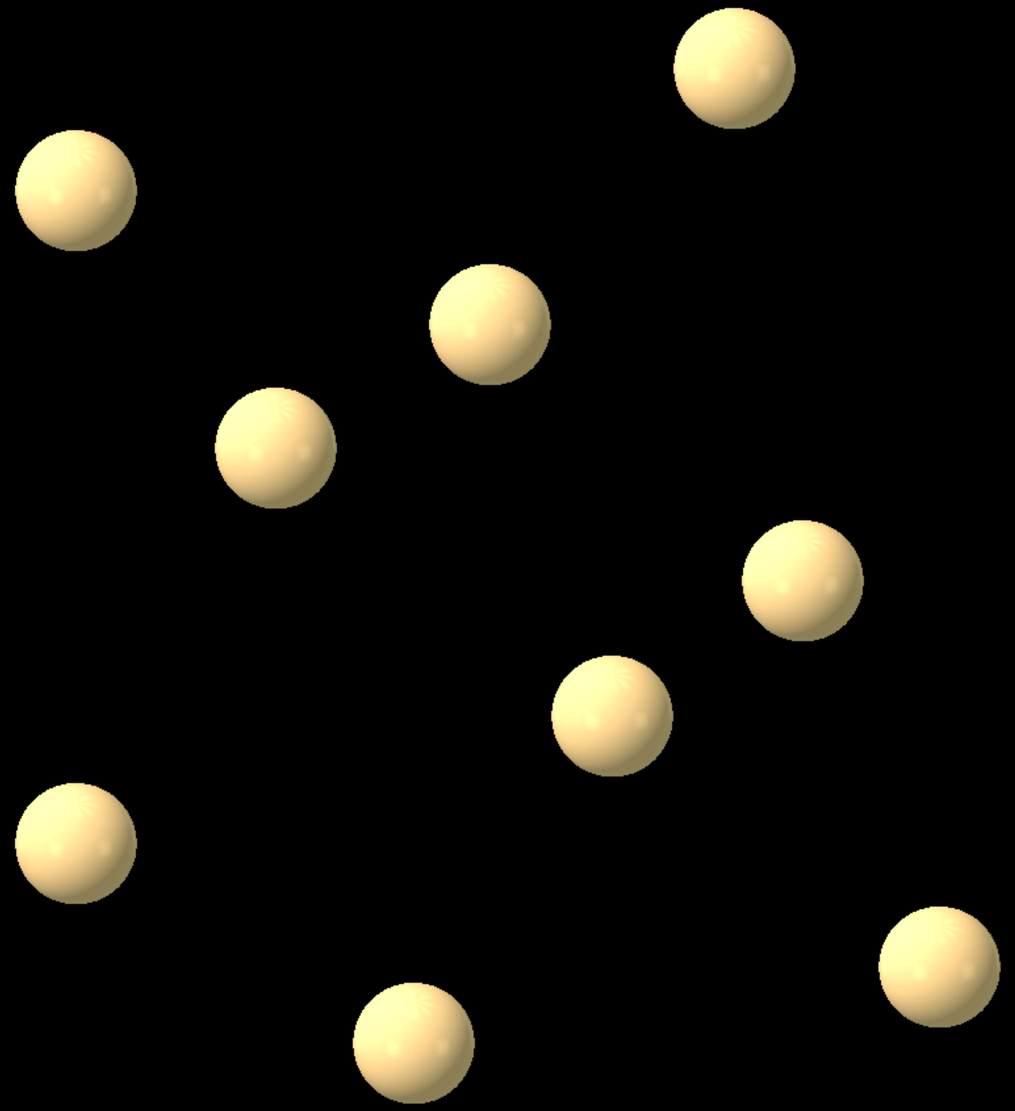
Example: Optical illusion

How to fool humans?



Example: Optical illusion

How to fool humans?



Reflection time

How do we fool cognitive systems?

- Another example: Eykholt et al. used stickers to fool a deep learning classifier of road signs
  - This would not fool humans!
- How do we fool cognitive systems?
- **Reflect:** What are the ethical implications of being able to manipulate AI decisions?



Grafitti (a “natural” occurance)

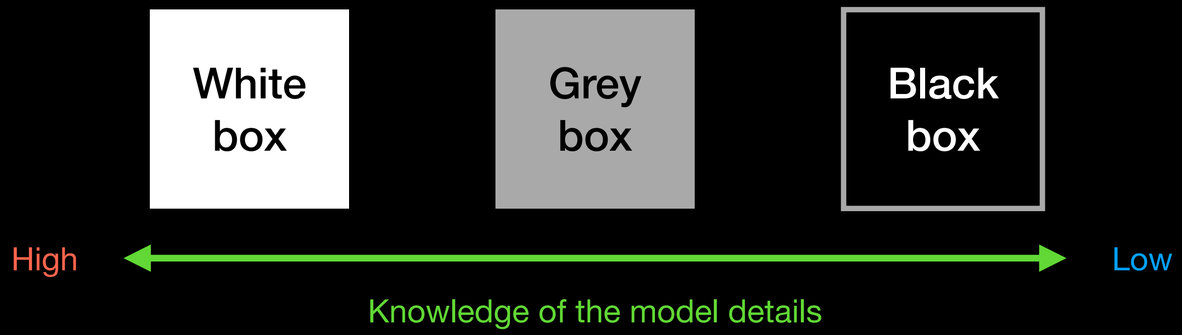


Physical perturbation (stickers)

White box vs. black box attacks

How to trick neural networks

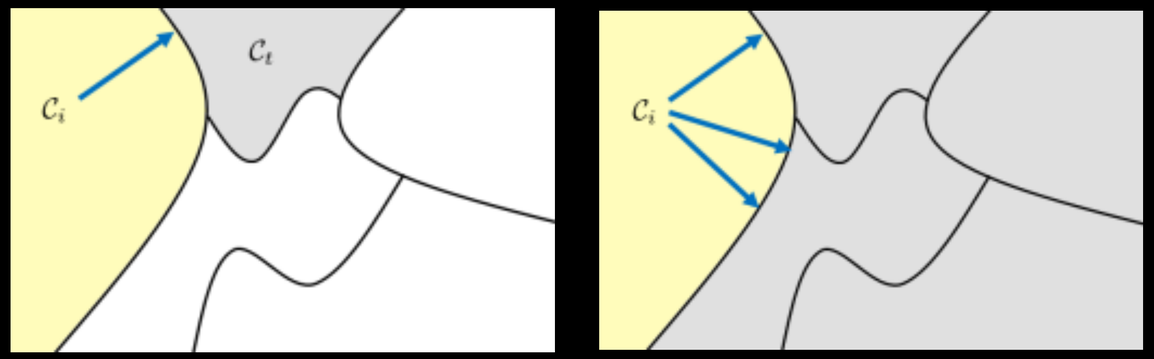
- **White box attacks:**
  The attacker has access to the model's parameters
- **Black box attacks:**
  The attacker does not have access to the model's parameters, only its outputs.
- *We will mostly focus on white box attacks in this lecture*



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



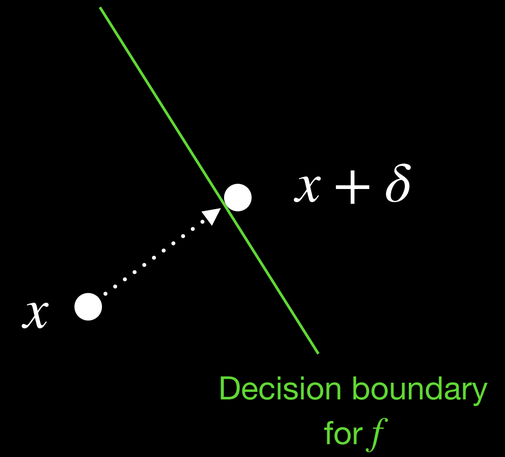
Targeted attack

Untargeted attack

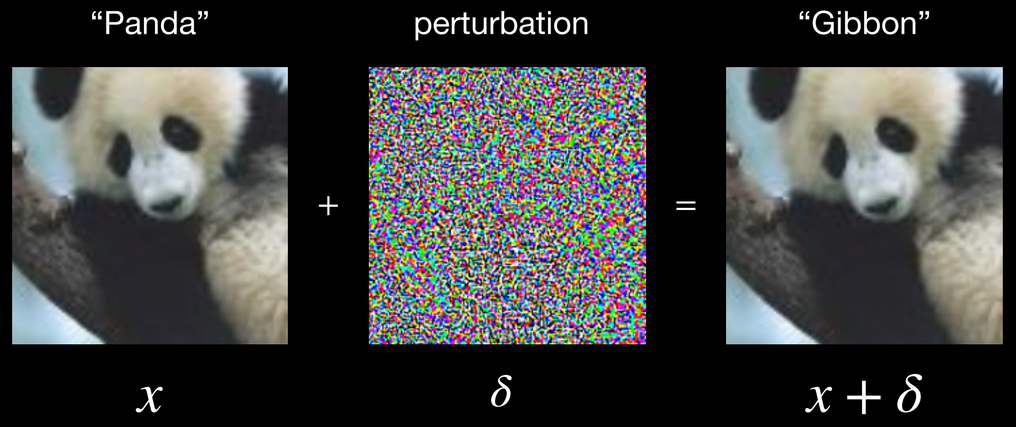
Image credit: https://engineering.purdue.edu/ChanGroup/ECE595/files/chapter3.pdf

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 f 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

NB: This is a busy slide, but we'll go through it slowly

- 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: *Gradient ascent away from original class* and *Gradient descent towards the least probable class*
- Optional: Can combine the two approaches $\delta_{n+1} = \delta_n + \eta\big(\nabla_\delta L(c^0, f(x+\delta_n)) - \nabla_\delta L(c^*, f(x+\delta_n))\big)$
- **Targeted version:** Use the gradient ascent method and set $c^* = c$, where c is the target class
- NB: Let $L(c, f(x))$ be the loss of the logit outputs, not the softmax probabilities from the last layer in the network. **Question:** Why?

Gradient ascent away from original class

- Find original class prediction: $c^0 = \text{argmax}_c\, f(x)_c$
- Initialize $\delta_0 = \mathbf{0}$.
  Then iterate until prediction changes from original class c^0
  $\delta_{n+1} = \delta_n + \eta \nabla_\delta L(c^0, f(x+\delta_n))$
  where η is some small stepsize

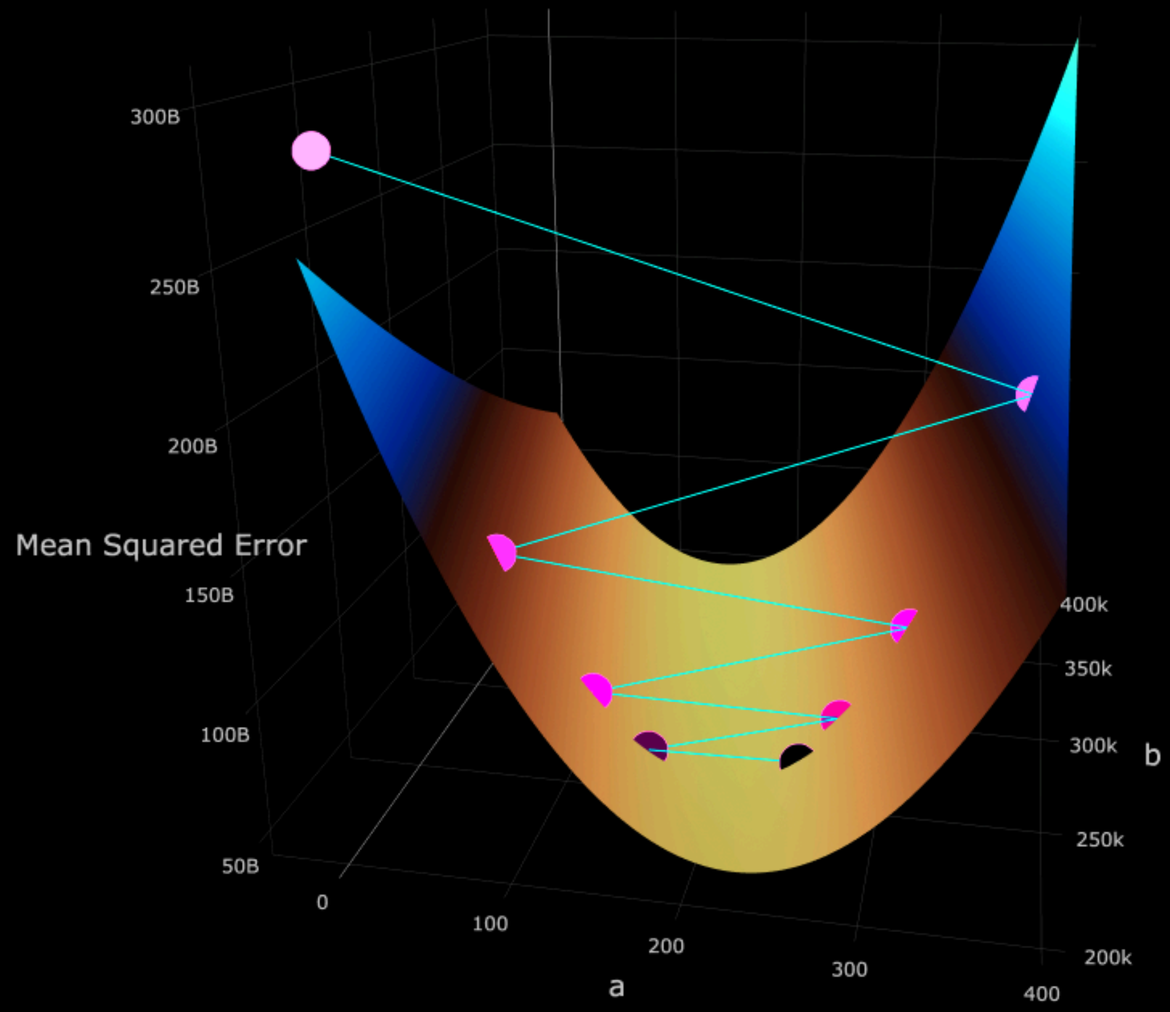
Gradient descent towards least probable class

- Find least probable class: $c^* = \text{argmin}_c\, f(x)_c$
- Initialize $\delta_0 = \mathbf{0}$.
  Then iterate until prediction changes from the original class
  $\delta_{n+1} = \delta_n - \eta \nabla_\delta L(c^*, f(x+\delta_n))$

Remark: Both methods stop iterating when $\text{argmax}_c\, f_c(x+\delta_n) \neq \text{argmax}_c\, f(x)$

Iterative gradient
White Box Attack

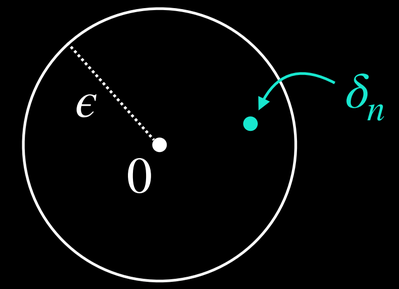
- Problem: How do we ensure the adversarial image is not wildly different from the original image?
- We want the perturbations δ_{n+1} to be "small"
- Quick fix: Use stepsize η small enough so the difference between x and final adversarial image $x + \delta_{n+1}$ is barely visible.
- **Question: Is this a good solution?**



Projected gradient method
Bounded by an epsilon ball

NB: Usually called projected gradient descent (PGD) in literature. We formulate is as gradient ascent here.

- Problem with the previous method: The perturbations δ are not bounded
- Idea: Put a constraint on the perturbations! Now we want $\delta = \max_\delta L(c, f(x+\delta))$ s.t. $\|\delta\| \leq \epsilon$ for some small $\epsilon > 0$
- Approach: Use iterative gradient descent/ascent, but after each iteration we project δ_{n+1} onto the ϵ-ball around 0, i.e. $B_\epsilon(0)$
- This projection π_ϵ depends on our choice of norm $\|\cdot\|$ $\pi_\epsilon(\tilde{\delta}_{n+1}) = \pi_{B_\epsilon(0),\,\|\cdot\|}(\tilde{\delta}_{n+1}) = \inf_{\delta \in B_\epsilon(0)} \|\tilde{\delta}_{n+1} - \delta\|$
- Common to use ℓ_2 or ℓ_∞ norm. Details on next slide.
- As before, we can combine gradient descent away from the original class and gradient ascent towards some other class $c^* \neq c^0$



ϵ-ball $B_\epsilon(0)$ around 0

Projected gradient ascent

- Find original class prediction: $c^0 = \text{argmax}_c\, f(x)_c$
- Then iterate until prediction changes from original class

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

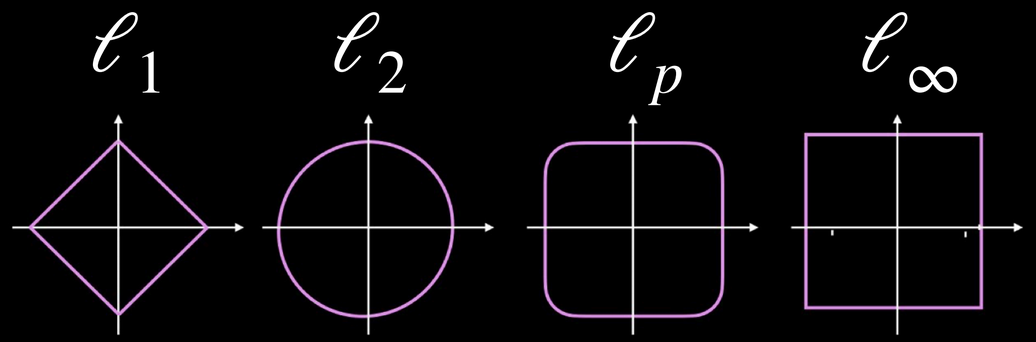
Projected gradient method

With the the ℓ_2 norm and ℓ_∞ norm

- For the ℓ_p norm:

  With $\|\cdot\|_p$ the projection is

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



- We generally work with the ℓ_2 (Euclidean) norm.

- For the ℓ_∞ norm:

  Recall $\|x\|_\infty = \max_n |x_n|$. The projection $\pi_\epsilon(\tilde{\delta})$ can be computed by $\pi_\epsilon(\tilde{\delta})_i = \min\{\max\{\tilde{\delta}_i, -\epsilon\}, \epsilon\}$ for each pixel value i

Use `clip(delta, -eps, eps)`

Common problems

Anything that can go wrong will go wrong

A. Pixel values after `ToTensor()` should be in $[0,1]$. This may not be the case for our adversarial sample $x + \delta$

B. PIL expects inputs as integers in $\{0,...,255\}$. After gradient updates, the pixels values are floats. The rounding of floats to uint8 before saving an image may change the predicted class and destroy our adversarial sample!

C. In general, the image saving process may induce changes in the pixel values due to lossy compression. Example: saving as jpg. Instead, save images as png without lossy compression.

Out of bounds problem

Quantization problem

Compression problem

Fast gradient sign and iterative gradient sign

Only use the sign of the gradient in the updates

Fast gradient sign

- $\delta = \eta \cdot \text{sign}\big(\nabla_x L(c^0, f(x))\big)$

  Adversarial image is then $\bar{x} = x + \delta$

- This simply adds a perturbation of magnitude $\eta > 0$ in the direction of the gradient to each pixel value of the image

- Requires no iterations - is much faster!

Both fast gradient sign and iterative gradient sign can be made into targeted attack methods.

Question: How would you do this?

Iterative gradient sign

- Find original class prediction:
  $c^0 = \text{argmax}_c\, f(x)_c$

- Initialize $\delta_0 = \mathbf{0}$.

- Then iterate until prediction changes from original class

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

Remark: This version of iterative gradient sign uses gradient descent away from the original class c^0.

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 a new adversarial perturbation for every input sample
- Universal adversarial attacks: Find one perturbation 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

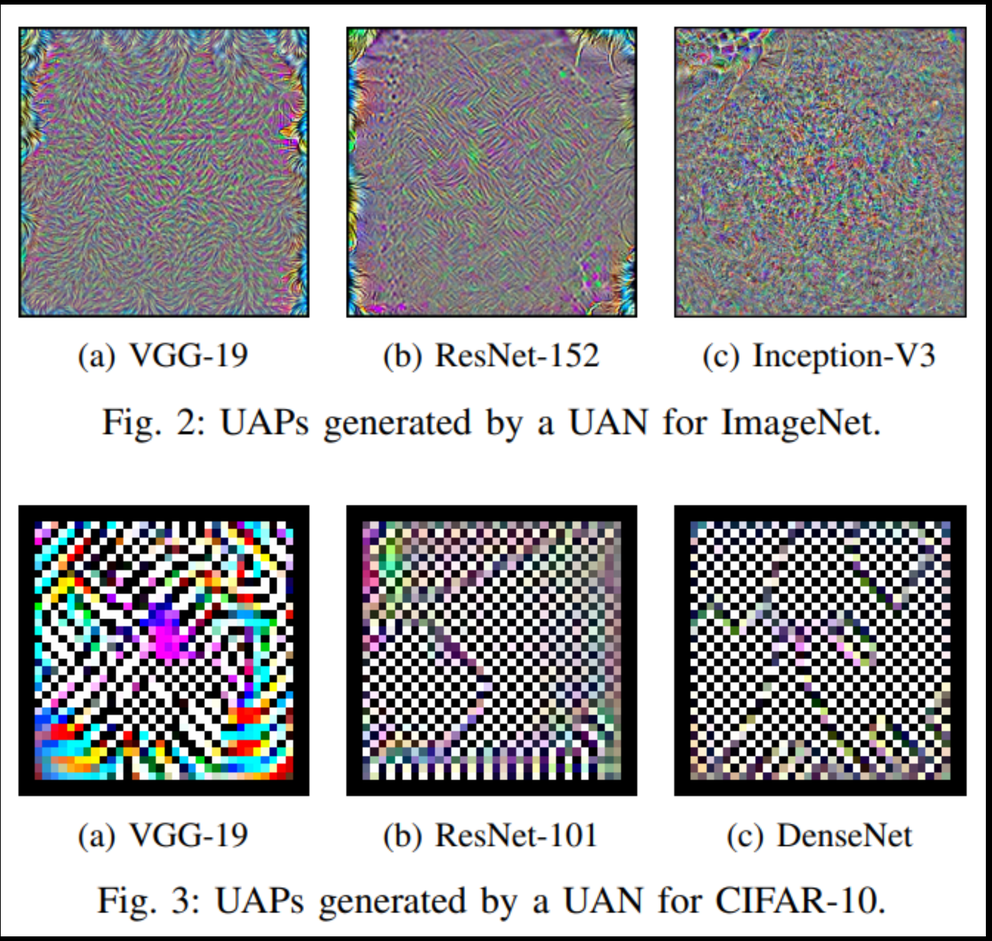
1. Initialize $\delta = \mathbf{0}$.
2. For each sample $x_i \in X$:
   - If $\text{argmax}_c\, f(x_i + \delta) = \text{argmax}_c\, f(x_i)$:
     - $\tilde{\delta}_i \leftarrow \text{argmin} \|v\|_2$ s.t. $\text{argmax}_c\, f(x_i + \delta + v) \neq \text{argmax}_c\, f(x_i)$
     - $\delta \leftarrow \pi_\epsilon(\delta + \tilde{\delta}_i)$
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
- Some examples of these UAPs capable of fooling 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

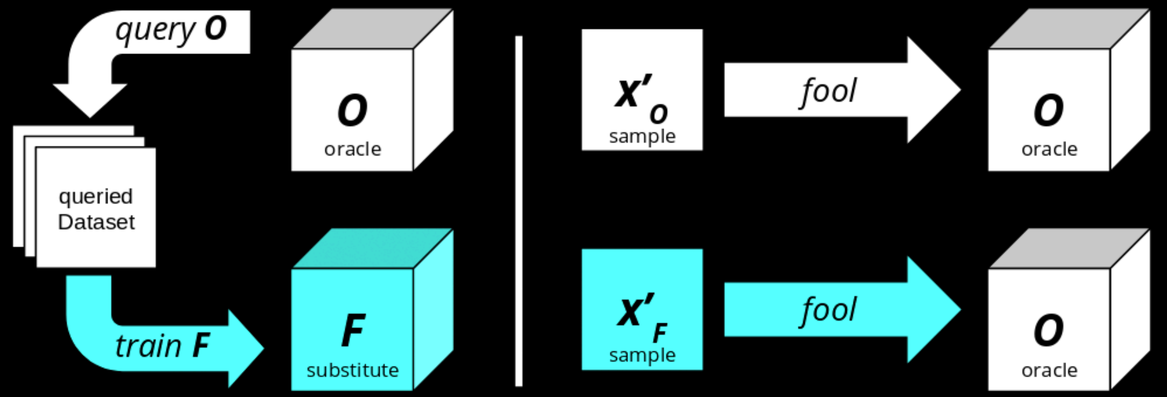
Black box attacks

A brief mention

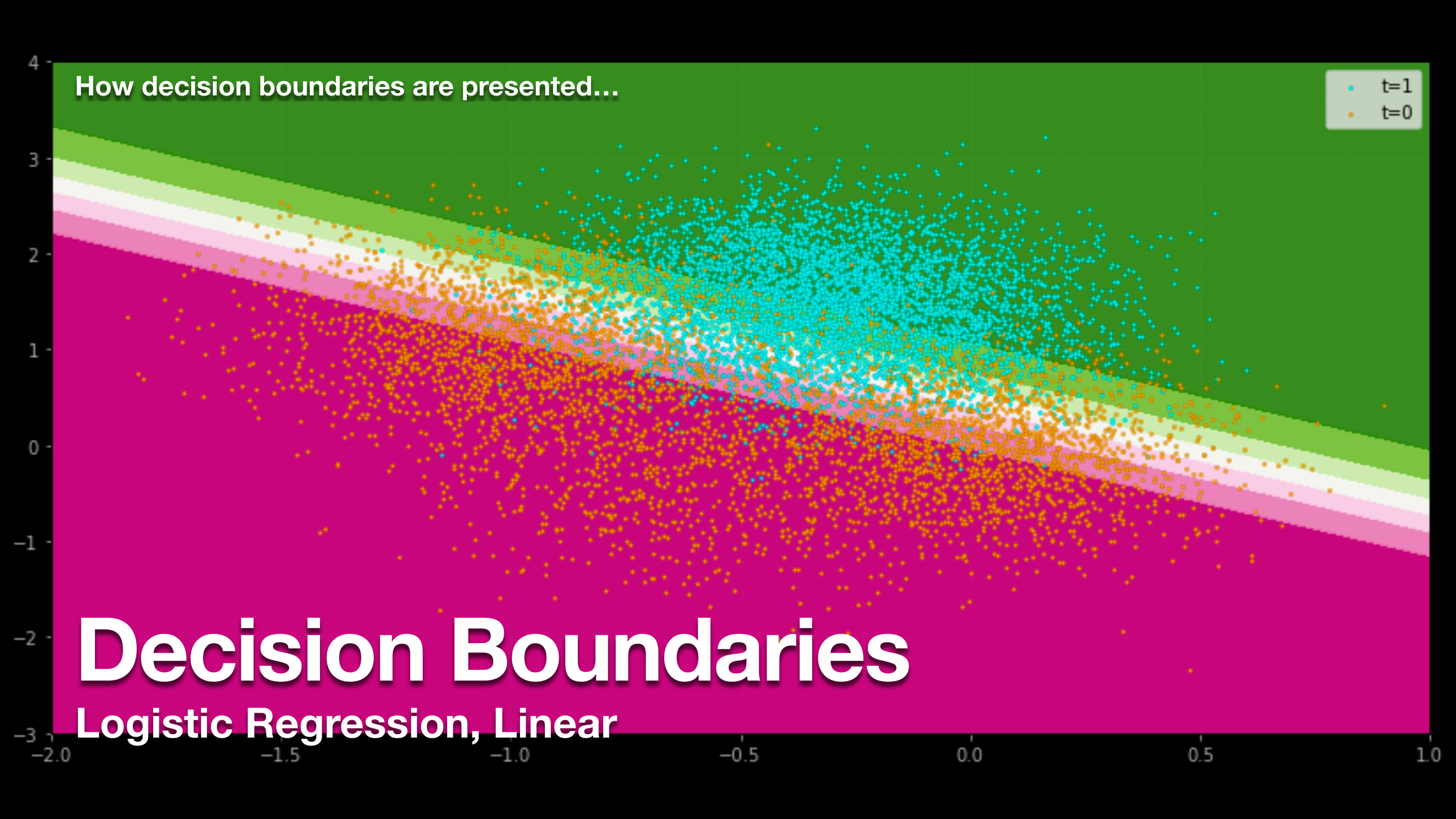
We only have access to the output $\hat{y} = f(x)$ of the model. This means we can't compute gradients!

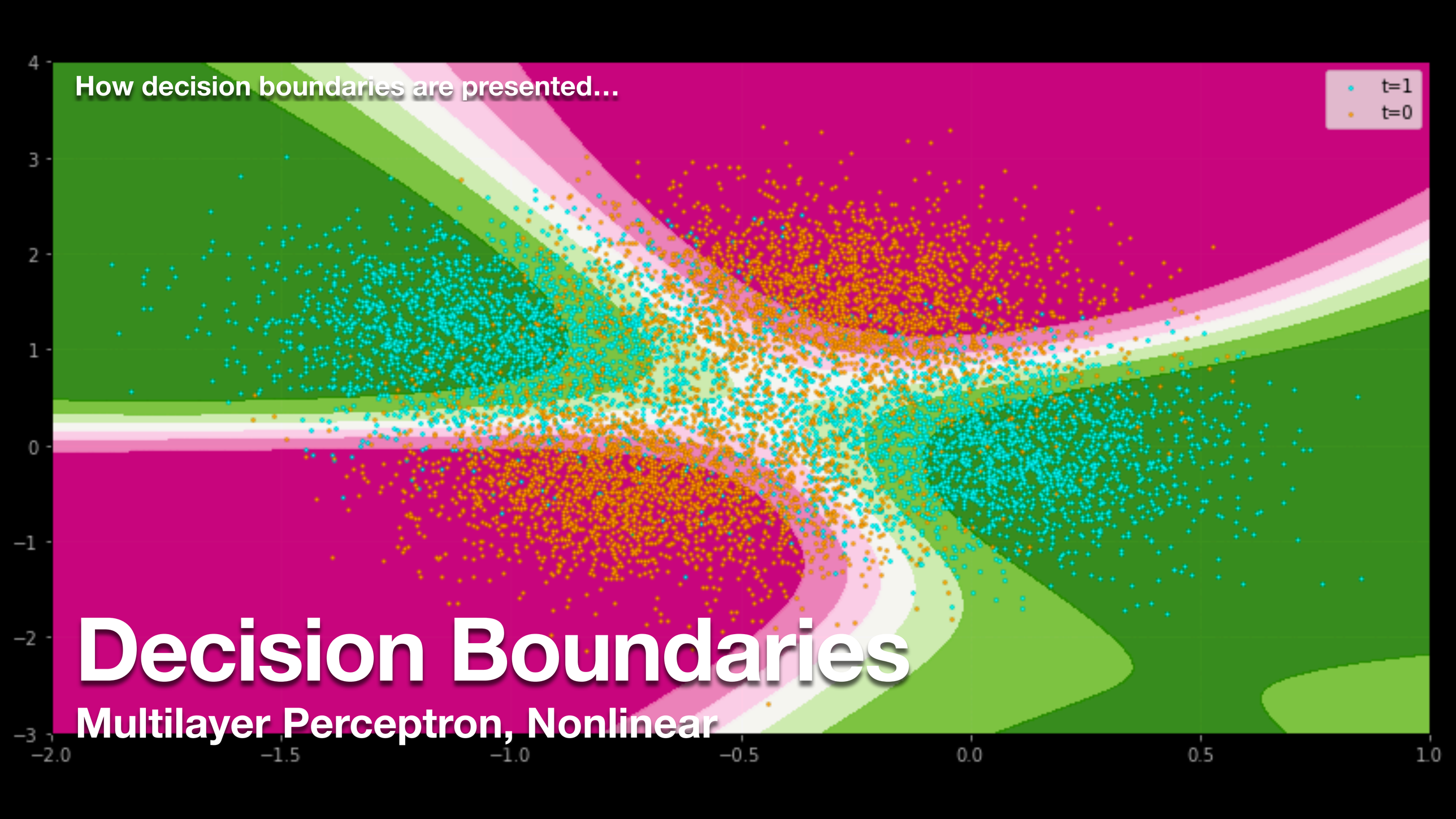
Two approaches:

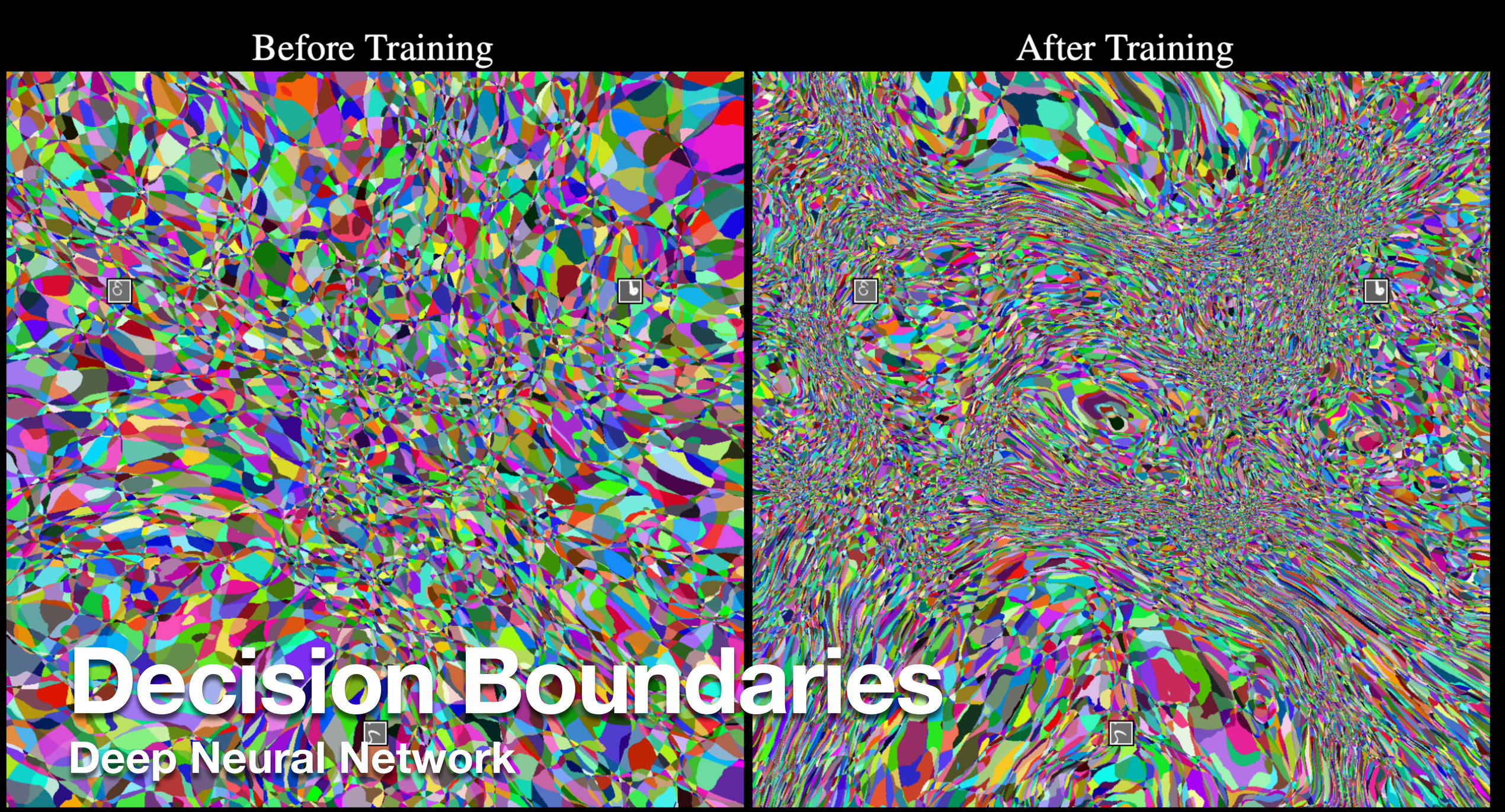
- Surrogate attacks: Train a surrogate model $\hat{f} \approx f$, and attack the surrogate with a white box method. Assume that the attack on $\hat{f}$ generalizes to f.
- Boundary attacks: starts with a random image and iteratively refines the adversarial example by moving it closer to the decision boundary.
  - **Importantly**: We want to end up on the wrong side of the boundary!
  - Without gradient, we instead use a hill-climbing approach.




Intermission
You look like you could use a snack!
Mr. Pops
We'll meet you in the lobby.


How decision boundaries are presented...
t=1
t=0
4
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
Decision Boundaries
Logistic Regression, Linear

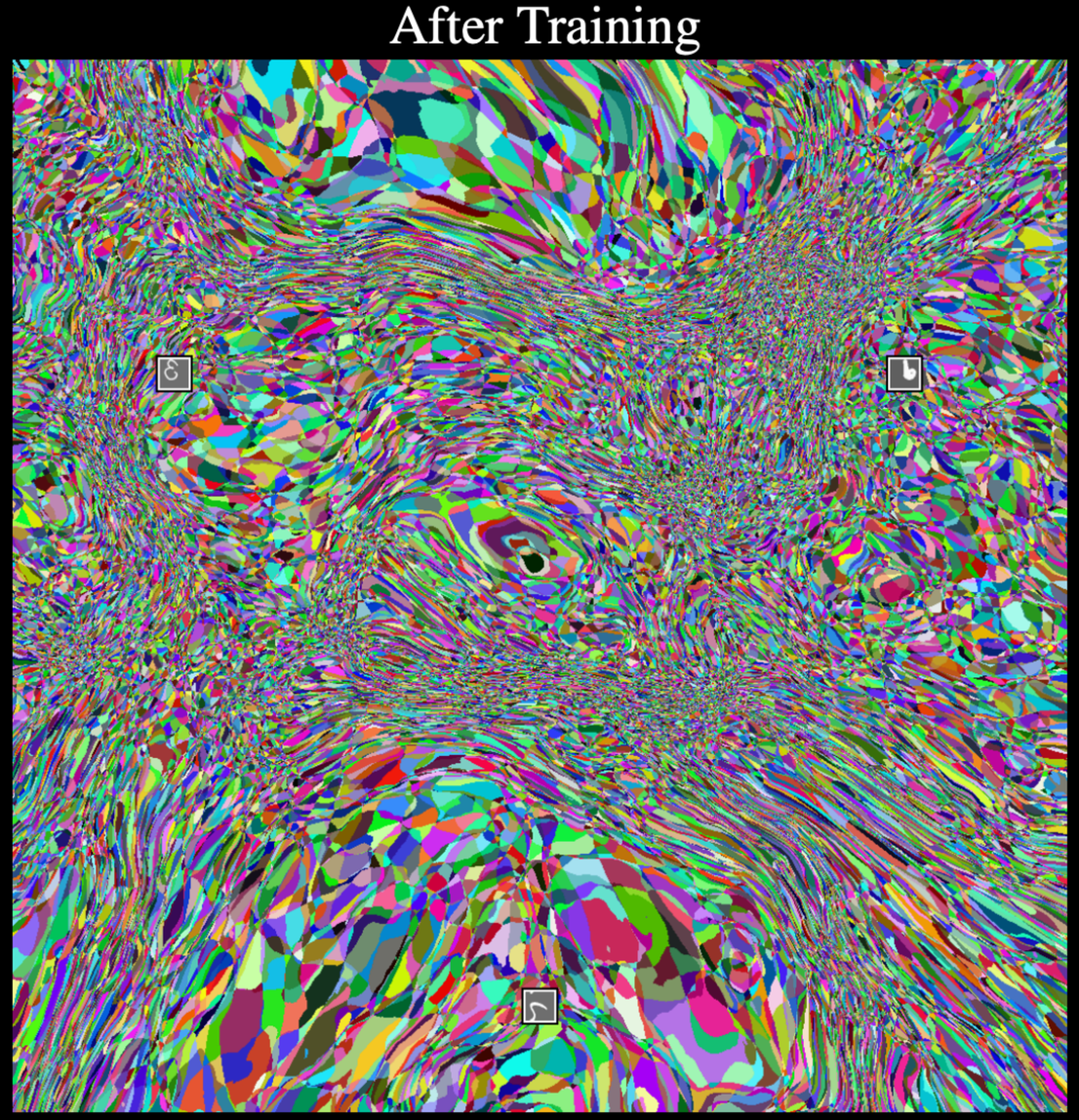

How decision boundaries are presented...
t=1
t=0
4
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
Decision Boundaries
Multilayer Perceptron, Nonlinear


Before Training
After Training
Decision Boundaries
Deep Neural Network

Why does this matter?

Complex decision boundary

- **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.

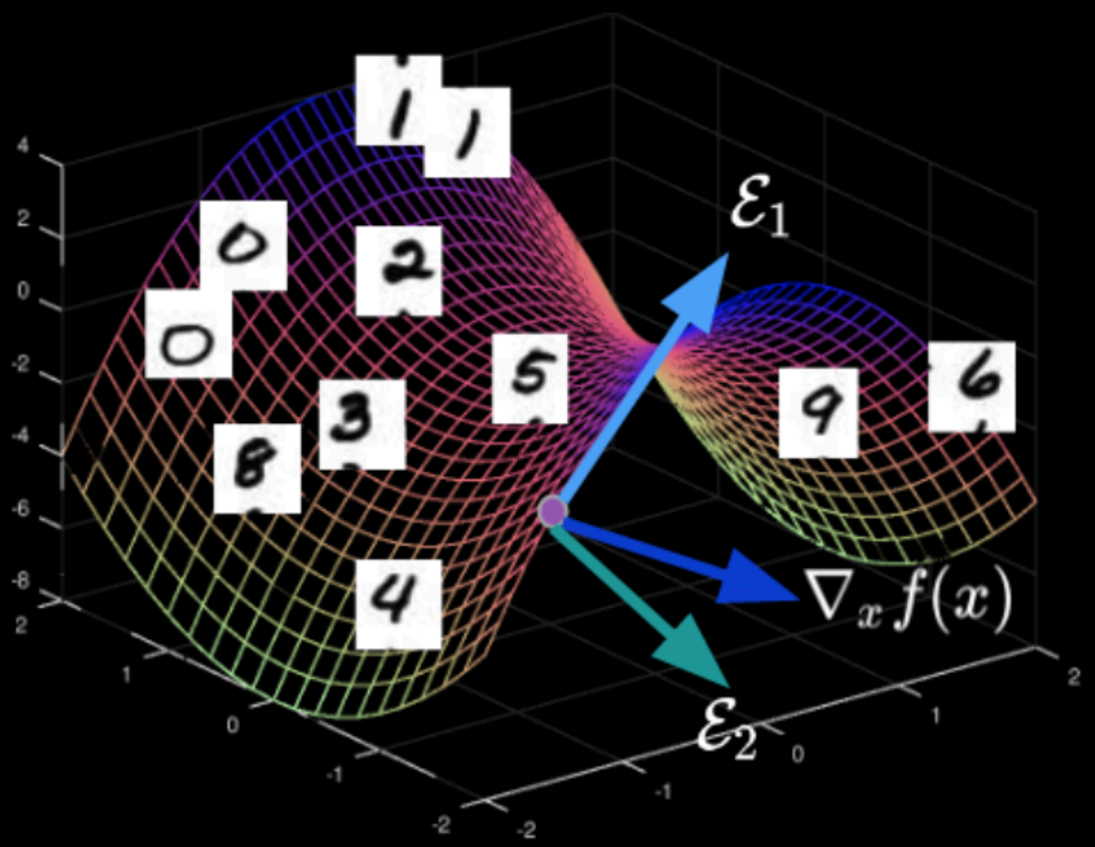


The Data Manifold

The Manifold Hypothesis

Assumption: High dimensional data has structure that lies close to some **lower dimensional manifold.**

- Num. possible RGB 256×256 images: $(256^3)^{256^2} = 1.75\ldots \times 10^{473479}$.*
- How many of these are realistic images of human faces?
- By the MH, the actual variations among different faces can be captured by a **much smaller set of dimensions.**



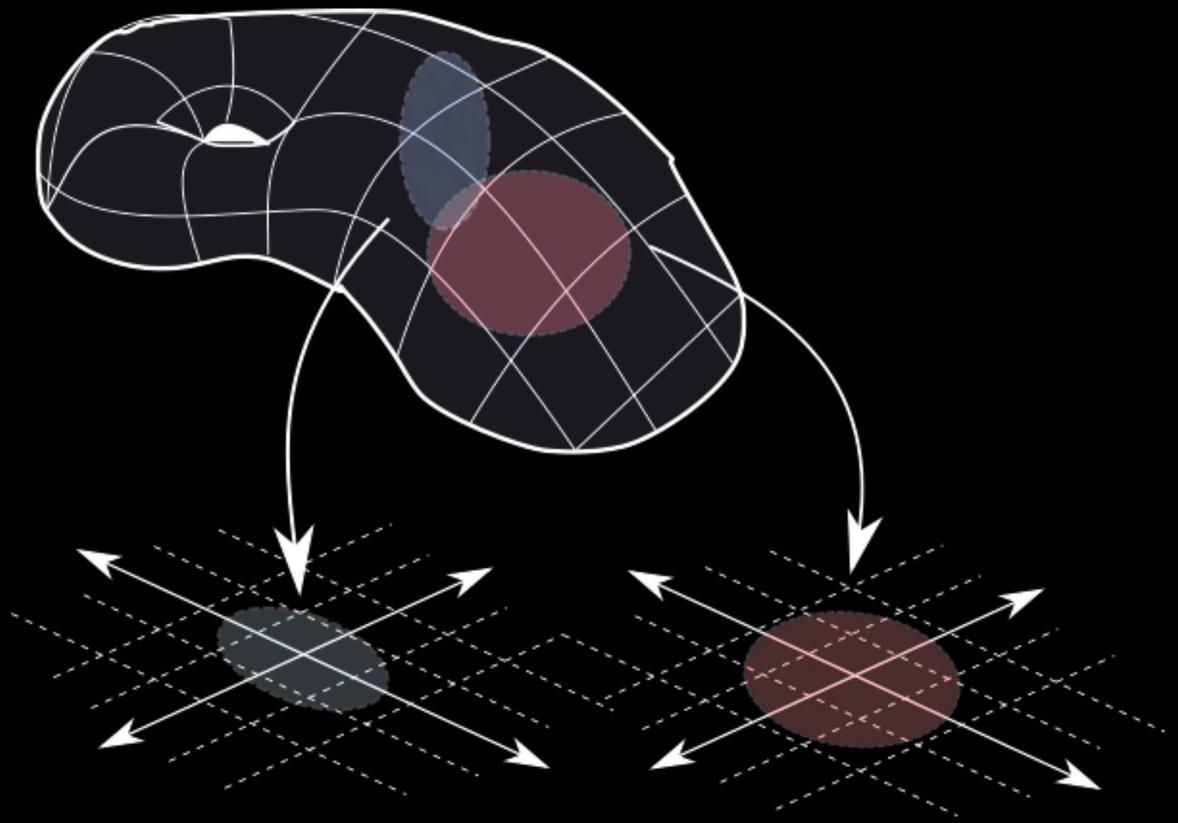
$k-$ dimensional manifold in $\mathbb{R}^d$

* Up to this point in the universe, if every atom were a supercomputer generating 1 billion images per second, they could have created approx. 5.22×10^{100} images. If we boost this to 1 billion images per unit of Planck time, we get approx. 10^{153} images. We've barely gotten started generating all the possible images…

Quick Digression

What is a manifold?

- A manifold is a smooth, continuous surface existing in a higher dimensional space.
- Consider a sheet of paper that you hold in your hand. If you twist, bend, or even crumple the paper, it is still a 2D object embedded in a 3D space*.
- Similarly, we can think of a piece of string as a 1D manifold in 3D space*.
- **Counterexample**: Unit cube in $\mathbb{R}^3$ (Why?)

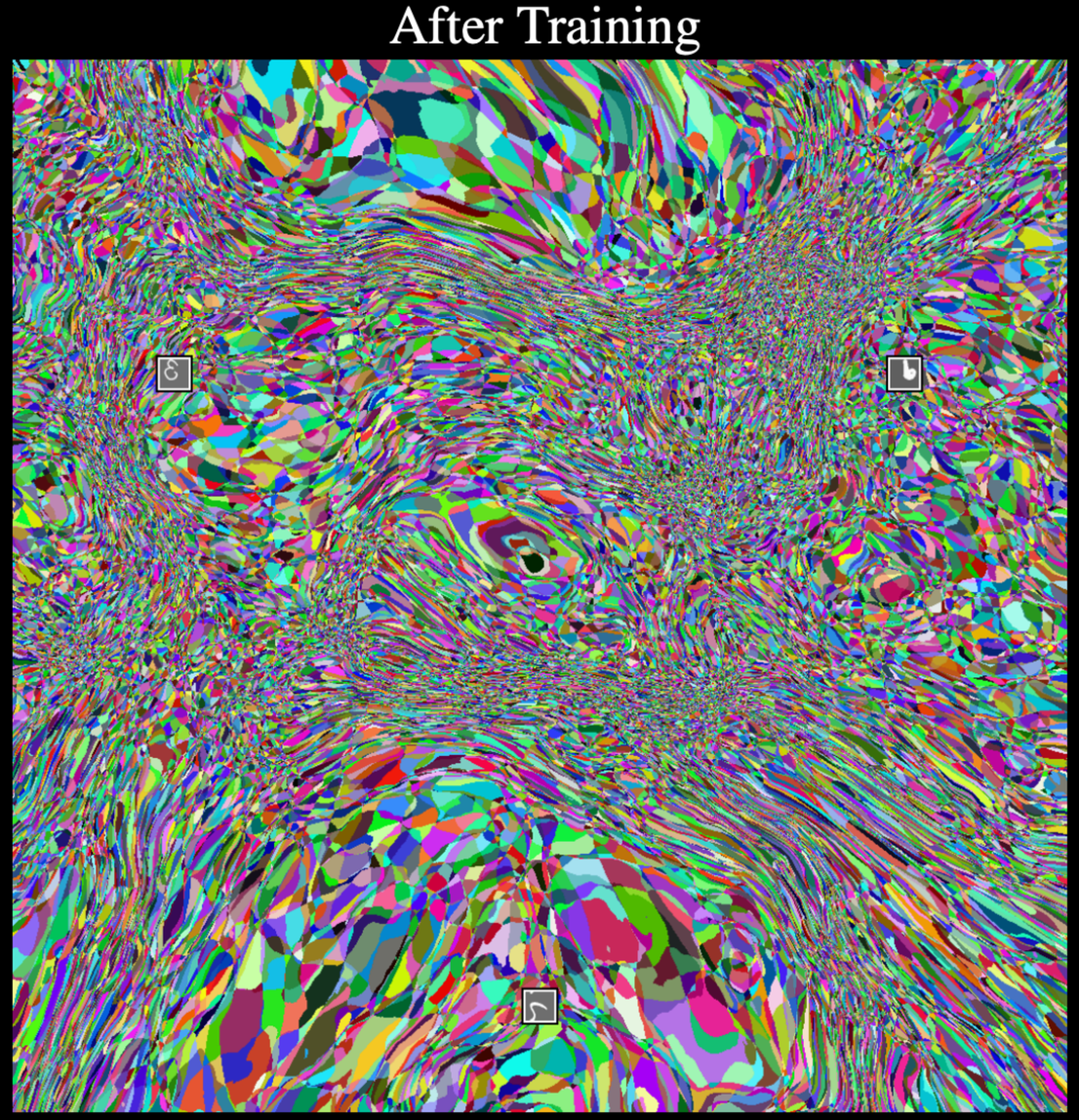


* **NOTE**: Both the sheet and the string are actually 3D objects with volume, but these are just illustrative examples

Some Intuition

Complex decision boundaries

- 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**.
- Thus, by moving just a tiny bit away from a training sample could lead to very different outputs.



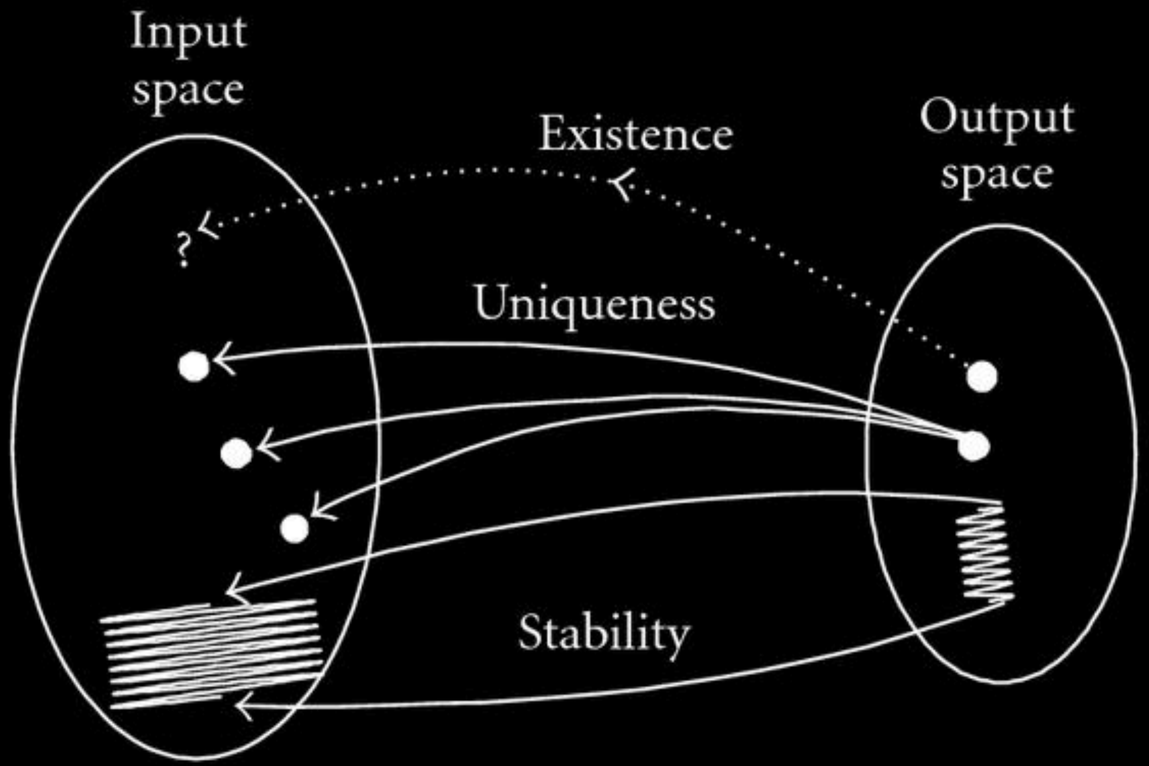
Well-Posedness
Motivating Stability

Well-posedness is a term used to describe problems that are "*nice*" or "*solvable*".

We say a problem is **well-posed** if:

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

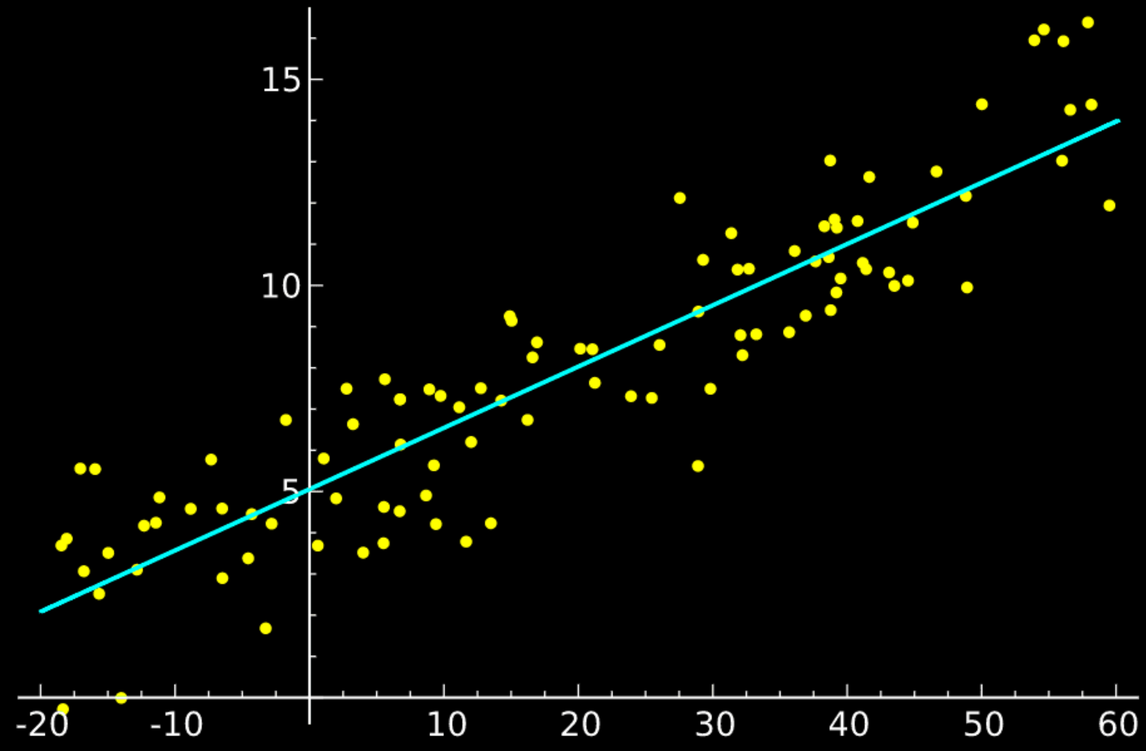
If one or more of these criteria do not hold, we say a problem is **ill-posed**.



Linear Projections

Well posed?

- Consider a simple linear model: $f(x_i) = Wx_i + b = y_i$, with $W \in \mathbb{R}^{m \times n}$.
- For $m < n$, the model is ill-posed, since multiple x_i can map to same y_i.
- This is however **not a problem** for classification or regression tasks.
- Importantly: f satisfies existence, and stability (continuous).
- **Convolutions** share this property, they are special cases of linear maps.



Ill-Conditioning

Stable, but badly behaved!

- Even if continuous mappings are technically stable, they can exhibit some **bad behaviour**.
- This means that they can have small changes in inputs which can lead to very large changes in outputs.

Any continuous f that satisfies

$$\|f(x) - f(y)\|_2 \leq k\|x - y\|_2$$

for all x, y, is *Lipschitz continuous**, and we call k the *Lipschitz constant* of f.

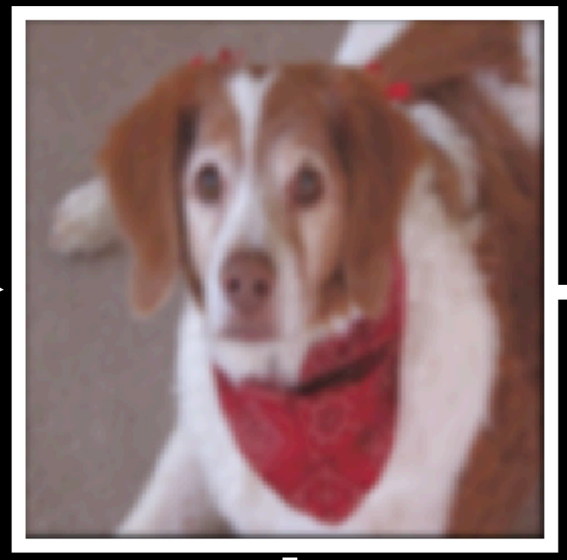


* With respect to Euclidean spaces, $\|\cdot\|_2$



x

Apply linear blur: W



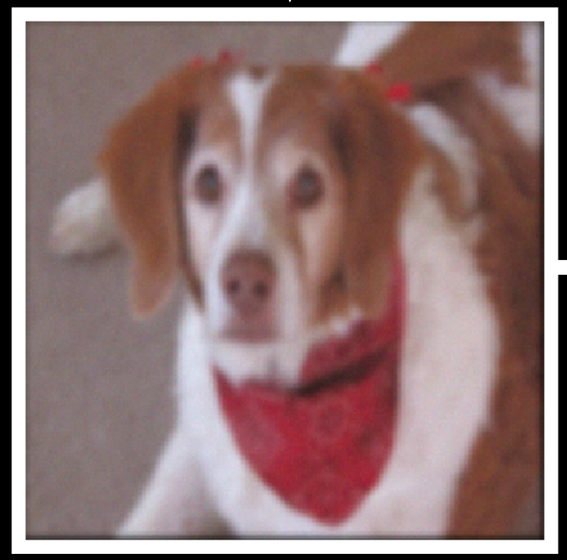
Wx

Apply inverse: W^{-1}



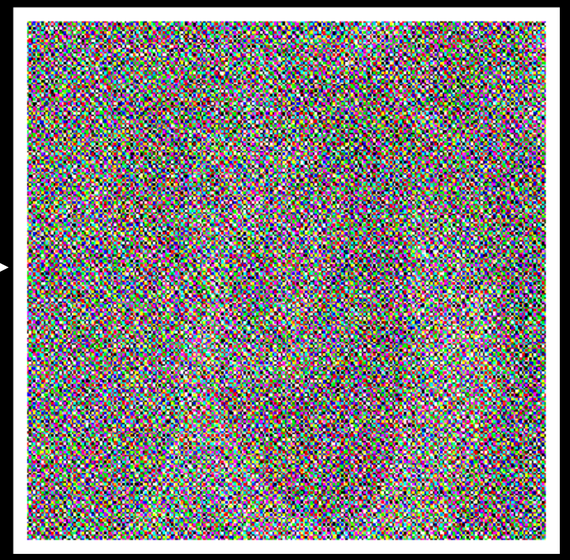
$W^{-1}Wx$

Add noise:
$\varepsilon \sim \mathcal{N}(0, .001)$



$Wx + \varepsilon$

Apply inverse: W^{-1}



$W^{-1}(Wx + \varepsilon)$

Ill-conditioning of Linear Operators:
Deblurring Example

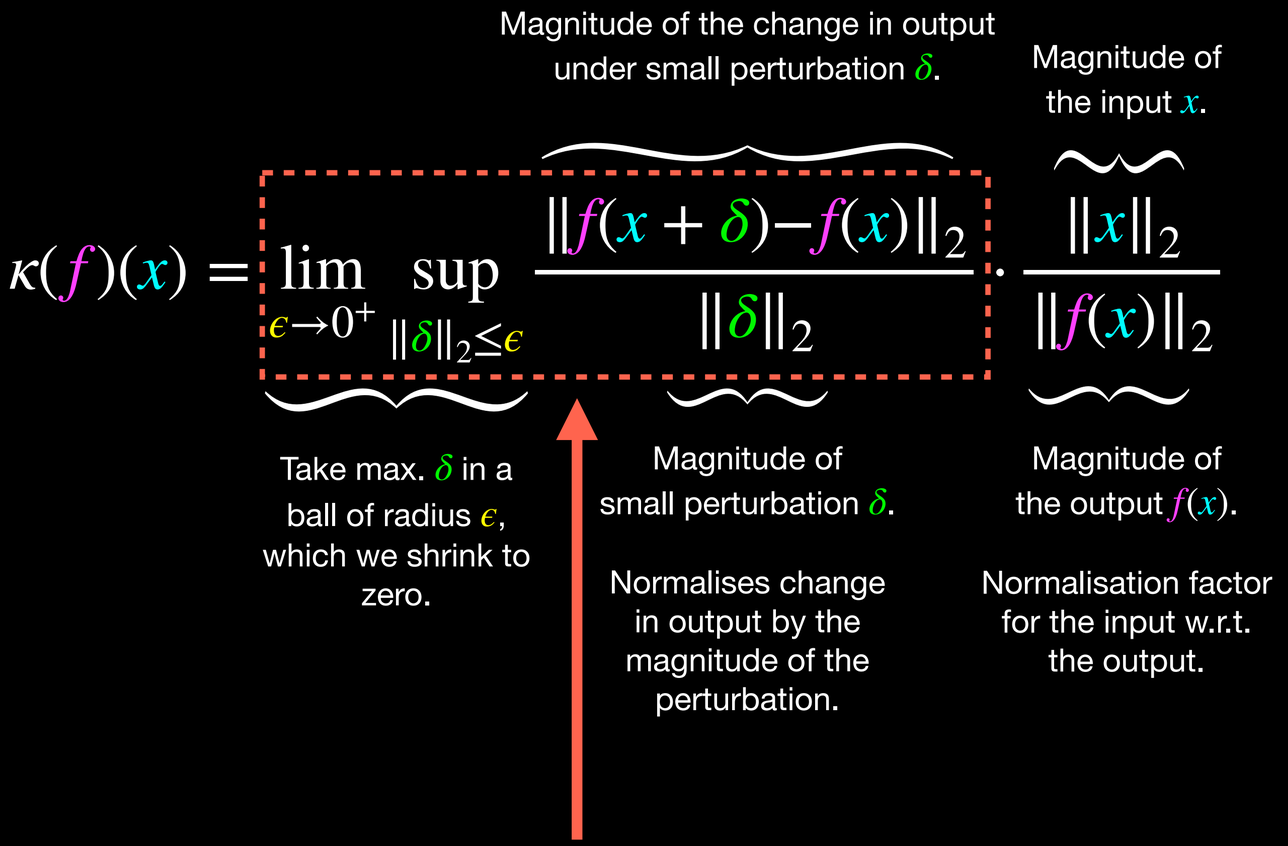
Condition Numbers

Quantising Ill-conditioning

Relative condition number:

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

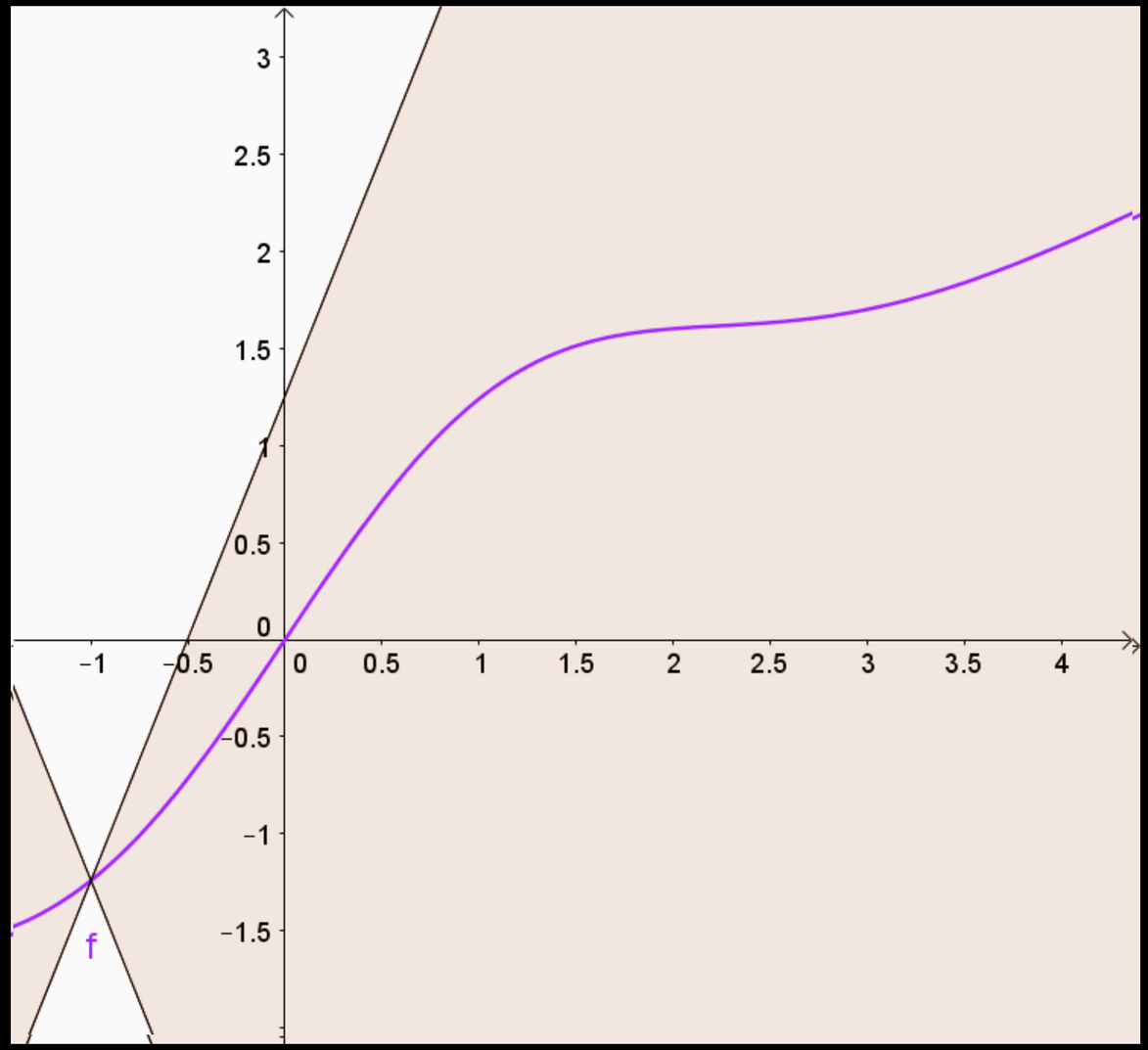
Essentially: normalise the magnitude of the derivative at x by the magnitude of the input w.r.t. the output.



Some More Intuition

Lipschitz? Condition number?

- **Lipschitz constant**: Maximal slope of a function. Says how “fast” the function can “jump”.
- **Rel. Cond. Num.:** Bounded from above by k_f / k_{f-1}, called the **absolute condition number.**
- If our network has a smaller Lip.const., it is **smoother**, and **less sensitive to perturbation**.



Matrix Conditioning

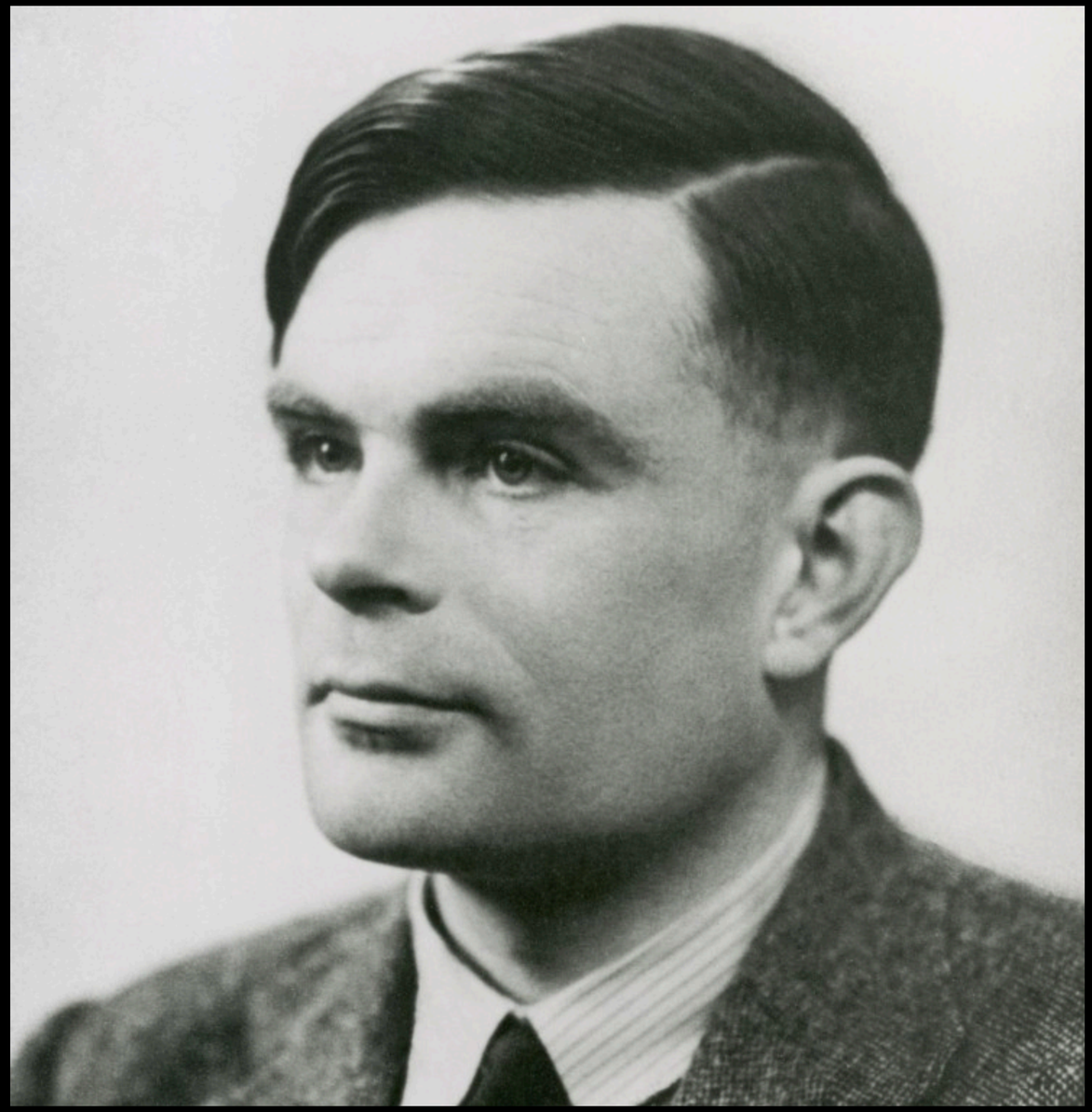
Condition numbers for Linear Maps

For linear maps (matrices), Alan Turing showed* that the abs. condition num. for Euclidean spaces 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**.



* Turing (1948): Rounding-Off Errors in Matrix Processes

ℓ_2 Regularisation
Fixing Bad Behaviour

- Consider the deblurring problem $Wx + \varepsilon = y$. We want to find $\hat{x}$.
- If we assume $\epsilon \sim \mathcal{N}(0, \alpha^{-1}I)$, we get

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

Taking the gradient w.r.t. x and setting this to zero:

$$\begin{aligned}
W^T(Wx - y) + \alpha x &= 0 \\
W^TWx + \alpha x &= W^Ty \\
(W^TW + \alpha I)x &= W^Ty \\
x &= (W^TW + \alpha I)^{-1}W^Ty \\
x &= (V\Sigma U^TU\Sigma V^T + \alpha I)^{-1}V\Sigma U^Ty \quad \text{(SVD)} \\
x &= (V\Sigma^2V^T + \alpha VIV^T)^{-1}V\Sigma U^Ty \\
x &= (V[\Sigma^2 + \alpha I]V^T)^{-1}V\Sigma U^Ty \\
x &= V\Sigma(\Sigma^2 + \alpha I)^{-1}U^Ty \\
x &= VD_\alpha U^Ty
\end{aligned}$$

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

ℓ_2 Regularisation
Fixing Bad Behaviour

- Consider the deblurring problem $Wx + \varepsilon = y$. We want to find $\hat{x}$.
- If we assume $\epsilon \sim \mathcal{N}(0, \alpha^{-1}I)$, we get

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

Taking the gradient w.r.t. x and setting this to zero:

$$\begin{aligned}
W^T(Wx - y) + \alpha x &= 0 \\
W^TWx + \alpha x &= W^Ty \\
(W^TW + \alpha I)x &= W^Ty \\
x &= (W^TW + \alpha I)^{-1}W^Ty \\
x &= (V\Sigma U^TU\Sigma V^T + \alpha I)^{-1}V\Sigma U^Ty \quad \text{(SVD)} \\
x &= (V\Sigma^2V^T + \alpha VIV^T)^{-1}V\Sigma U^Ty \\
x &= (V[\Sigma^2 + \alpha I]V^T)^{-1}V\Sigma U^Ty \\
x &= V\Sigma(\Sigma^2 + \alpha I)^{-1}U^Ty \\
x &= VD_\alpha U^Ty
\end{aligned}$$

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

Dampening of singular values!!!

ℓ_2 Regularisation
Fixing Bad Behaviour

- Consider the deblurring problem $Wx + \varepsilon = y$. We want to find $\hat{x}$.
- If we assume $\epsilon \sim \mathcal{N}(0, \alpha^{-1}I)$, we get

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

Not Exam Material!

Taking the gradient w.r.t. x and setting this to zero:

$$\begin{aligned}
W^T(Wx - y) + \alpha x &= 0 \\
W^TWx + \alpha x &= W^Ty \\
(W^TW + \alpha I)x &= W^Ty \\
x &= (W^TW + \alpha I)^{-1}W^Ty \\
x &= (V\Sigma U^TU\Sigma V^T + \alpha I)^{-1}V\Sigma U^Ty \quad \text{(SVD)} \\
x &= (V\Sigma^2V^T + \alpha VIV^T)^{-1}V\Sigma U^Ty \\
x &= (V[\Sigma^2 + \alpha I]V^T)^{-1}V\Sigma U^Ty \\
x &= V\Sigma(\Sigma^2 + \alpha I)^{-1}U^Ty \\
x &= VD_\alpha U^Ty
\end{aligned}$$

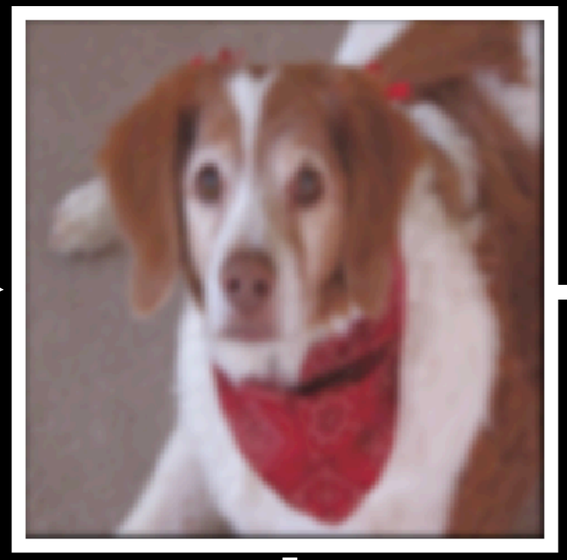
Dampening of singular values!!!

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



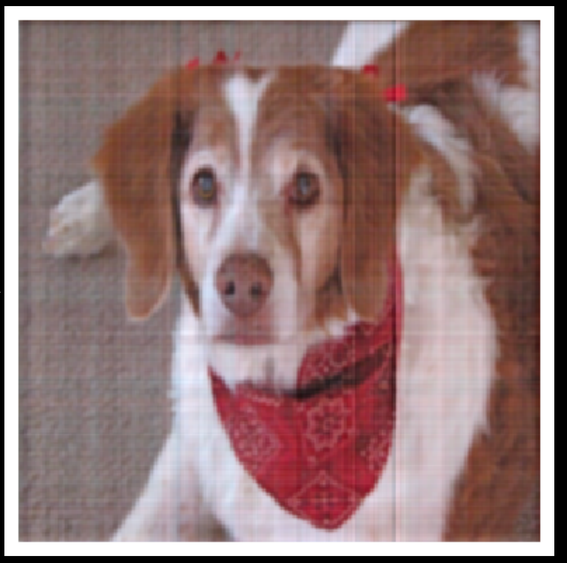
x

Apply linear blur: W



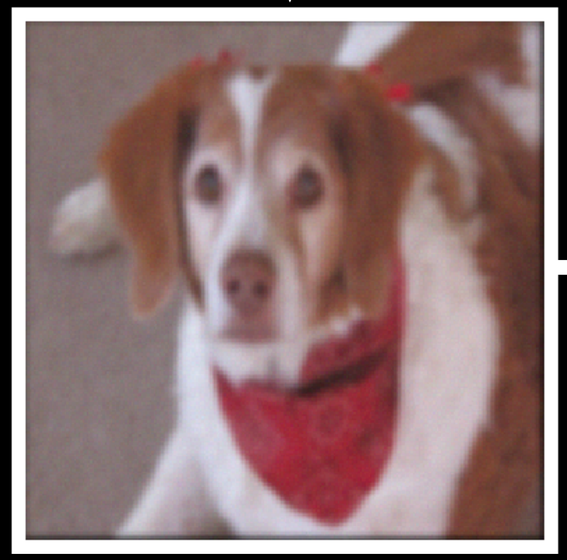
Wx

Apply regularised inverse: $VD_\alpha U^T$



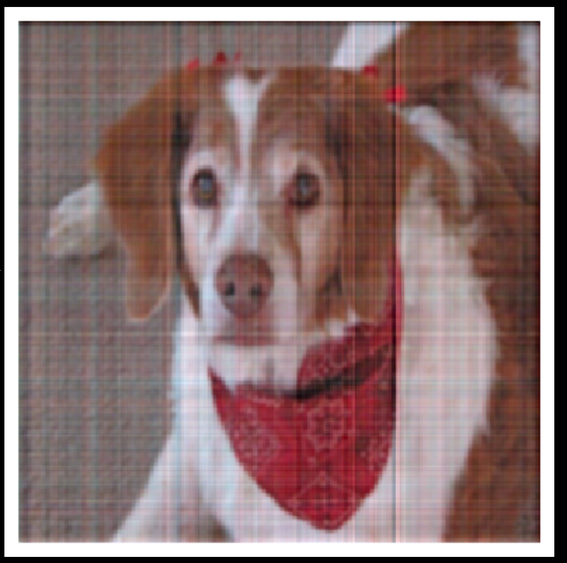
$VD_\alpha U^T Wx$

Add noise: $\varepsilon \sim \mathcal{N}(0, .001)$



$Wx + \varepsilon$

Apply regularised inverse: $VD_\alpha U^T$



$VD_\alpha U^T(Wx + \varepsilon)$

Regularisation of Linear Operators:
Deblurring Example

ℓ_2 Regularisation
How do we do this in NNs?

- ℓ_2 regularisation is straightforward in SGD.
- Using modified update rule $w_{t+1} = (1 - \lambda)w_t - \eta \nabla_w \mathscr{L}(w_t)$
- The term λ dampens the weight by **weight decay**.
- To see the equivalence, we note $\nabla_w \mathscr{L}_{\text{reg}}(w_t) = \nabla_w \mathscr{L}(w_t) + \alpha w_t$

Refresher:
w_t := Current weight at step t
η := Learning rate
$\mathscr{L}$:= Loss function

Plugging into SGD:

$$\begin{aligned} w_{t+1} &= w_t - \eta \nabla_w \mathscr{L}_{\text{reg}}(w_t) \\ &= w_t - \eta\big(\nabla_w \mathscr{L}(w_t) + \alpha w_t\big) \\ &= (1 - \eta\alpha)w_t - \eta \nabla_w \mathscr{L}(w_t) \end{aligned}$$

Setting $\lambda = \eta\alpha$ obtains the weight decayed update.

ℓ_1 Regularisation
The Lasso method

- Another common method: ℓ_1 regularisation (Lasso).
- Increases sparsity → setting many of the weights to zero.
- This both reduces the number of effective parameters in the model and prevents overfitting.
- **Con:** Non-convex optimisation, not as straighforward as ℓ_2.

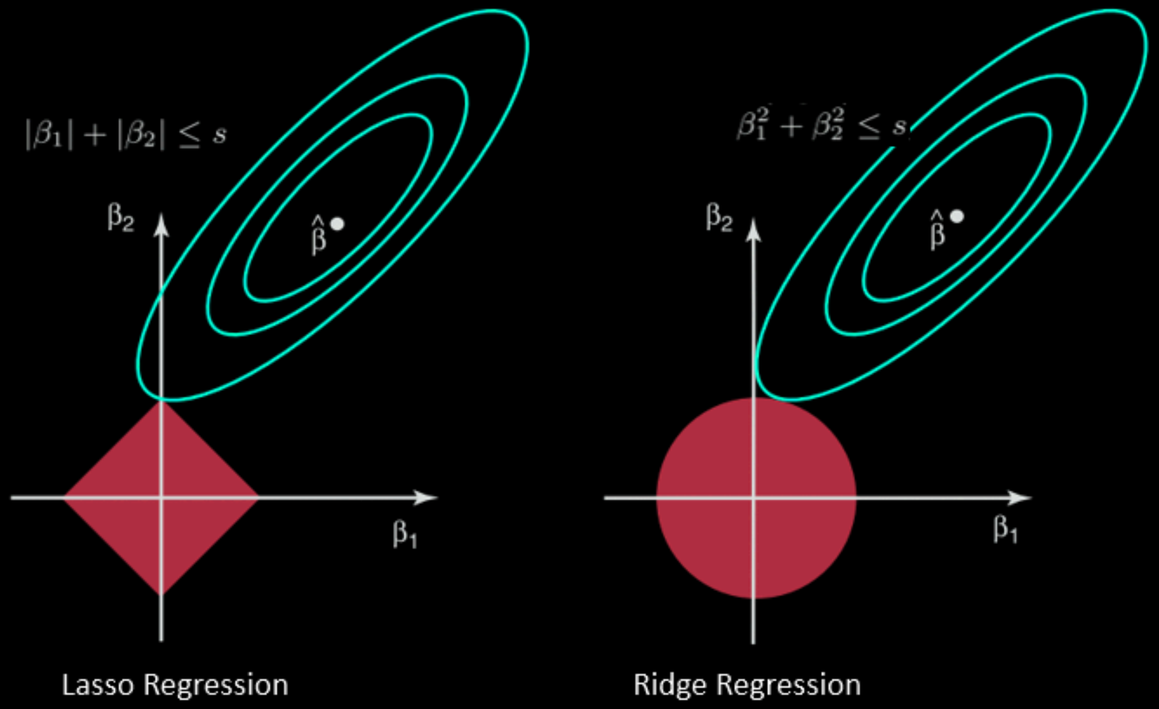
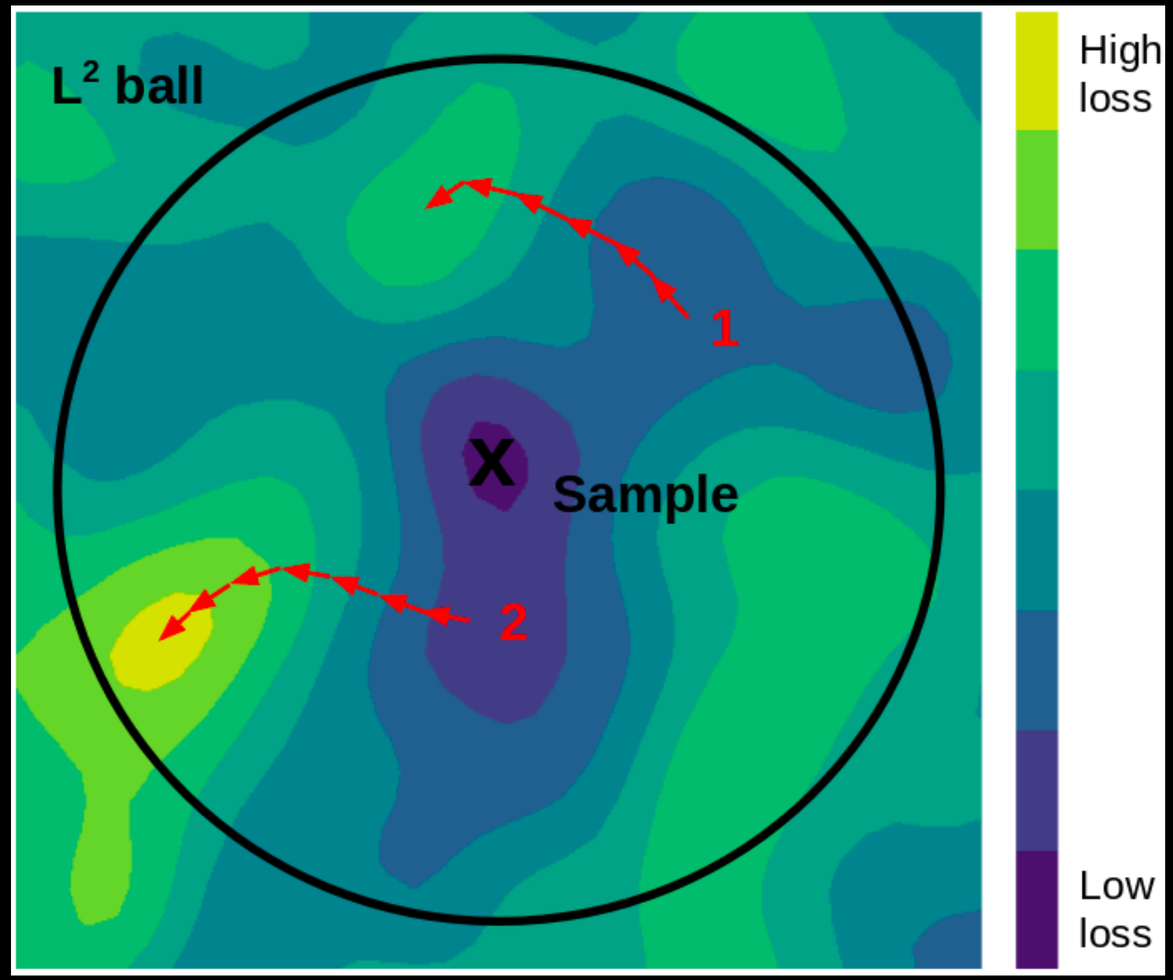


Image credit: Elements of Statistical Learning (Hastie, Tibshirani, Friedman, 2001)

Adversarial Perturbation

Compared to Stochastic Noise

- **Stochastic perturbation**: Select random element in a ball centred on input according to prob.distribution.
- **Adverserial perturbation**: Select perturbation in the ball which maximally displaces the output.
- **SO**: adv. attacks are **worst case** perturbations, often close to the perimeter of the ball.
- **How do we defend against this?**

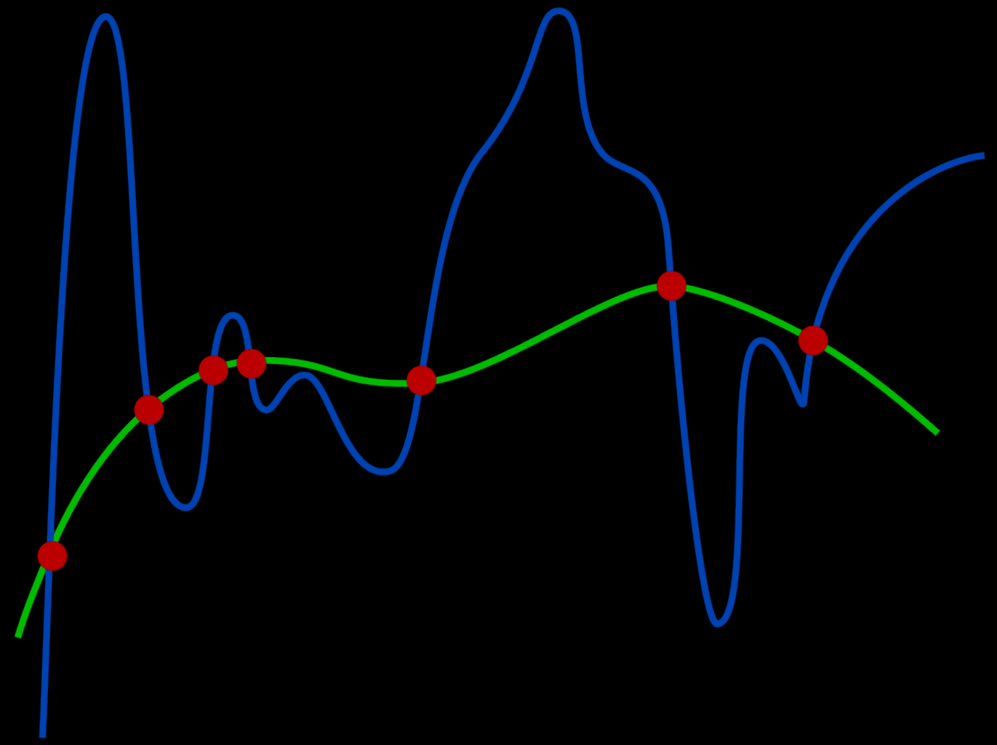


First Line of Defence

Classic Regularisation

Reducing condition number helps adversarial robustness.

- **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 activation,** e.g., ELU, CELU, GELU.

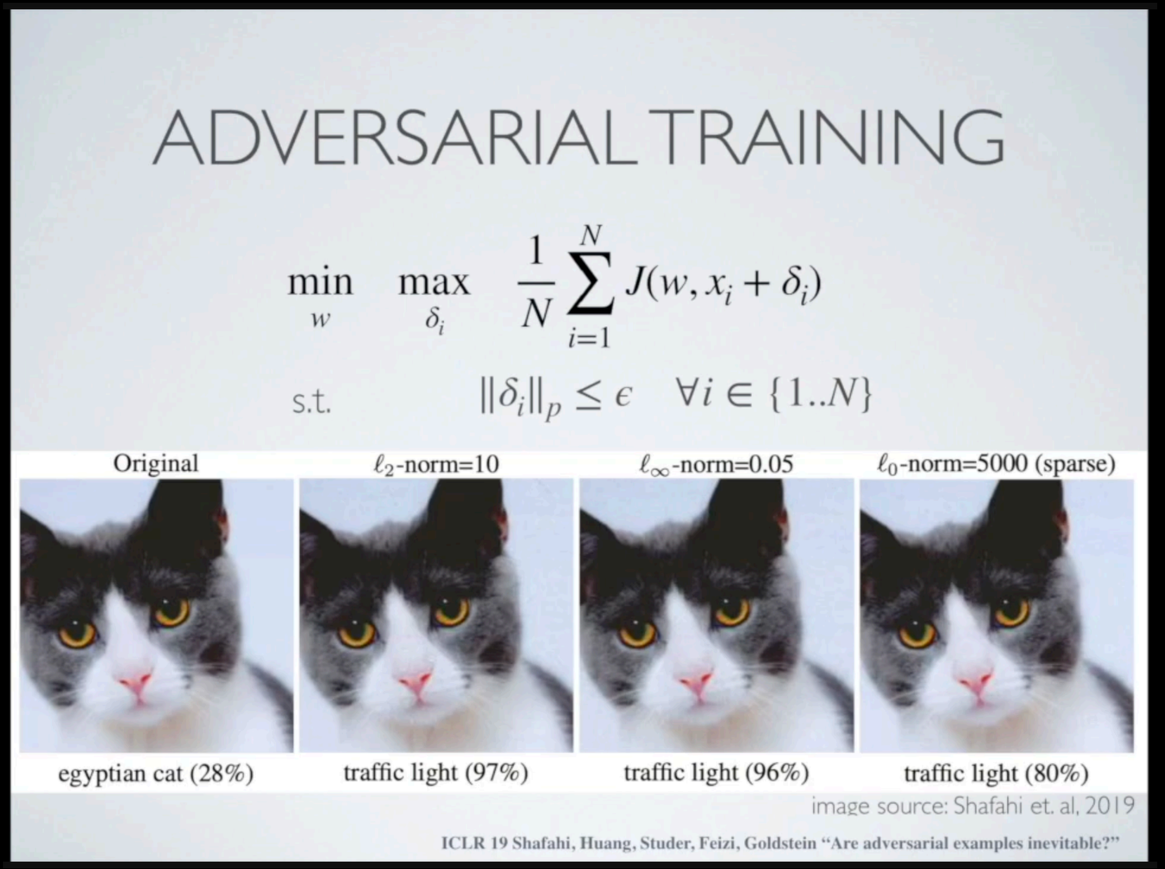


Alternate Defences 1

Adversarial Training

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

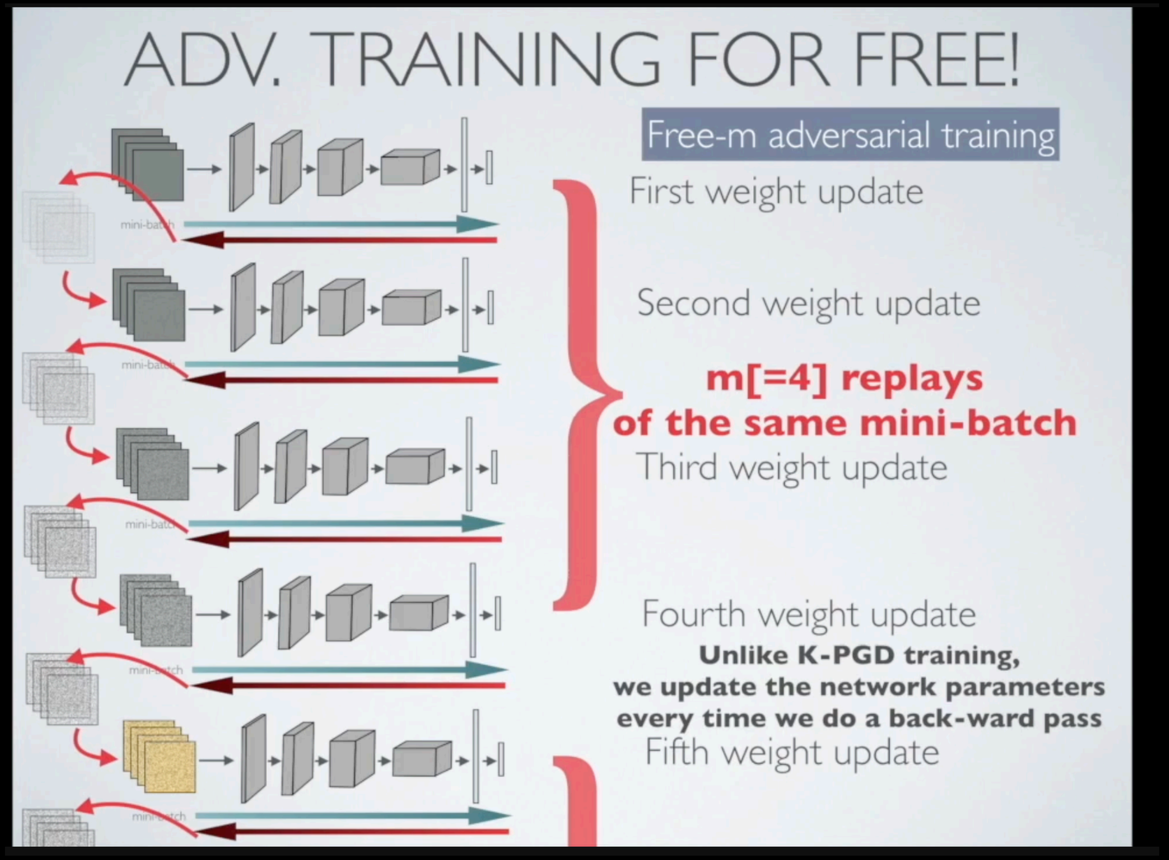
In practice: Expensive, reduced predictive power.



Alternate Defences 1

Adversarial Training: for free?

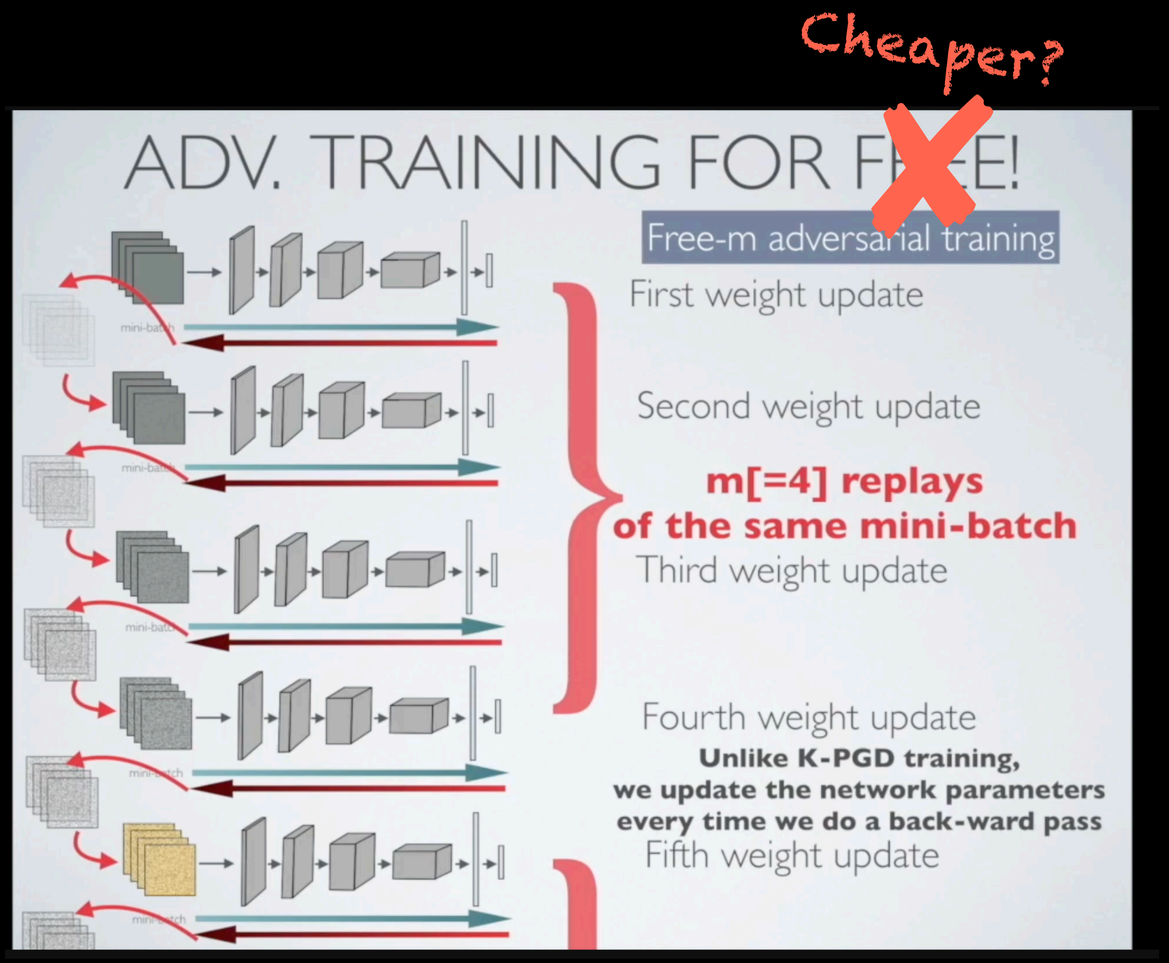
- Shafahi et.al. (2019) proposes concurrent estimation by learning and updating a global perturbation, using m “replay steps” over the same batch.
- **Pros:** Backward passes are now performed for every iteration, which is more effective.
- **Cons:** Potentially harms the stochasticity of the gradients. Certainly not “free”.



Alternate Defences 1

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, which is more effective.
- **Cons:** Potentially harms the stochasticity of the gradients. Certainly not "free".



Alternate Defences 2

Preprocessing: Quantisation

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.

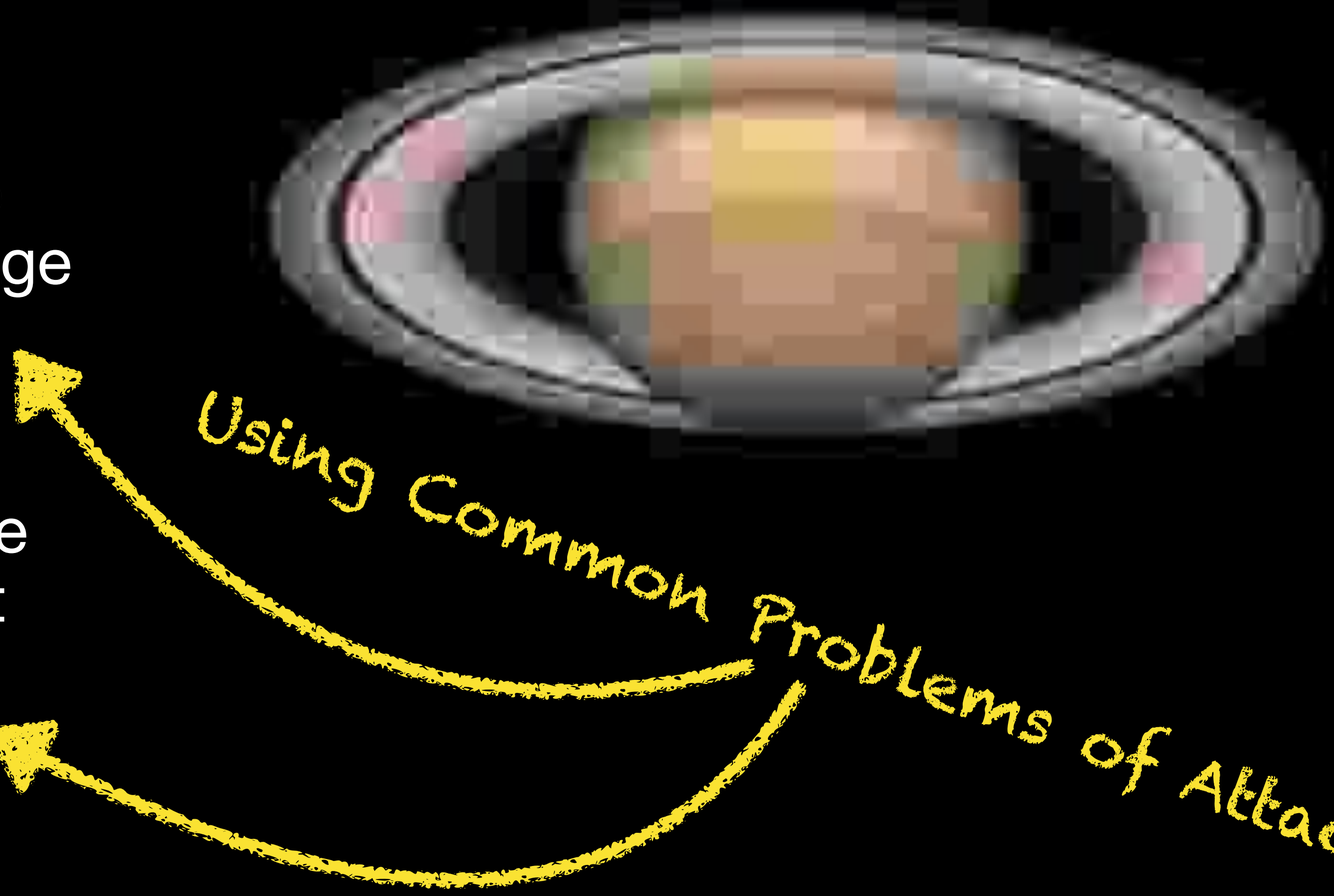


Alternate Defences 2

Preprocessing: Quantisation

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.

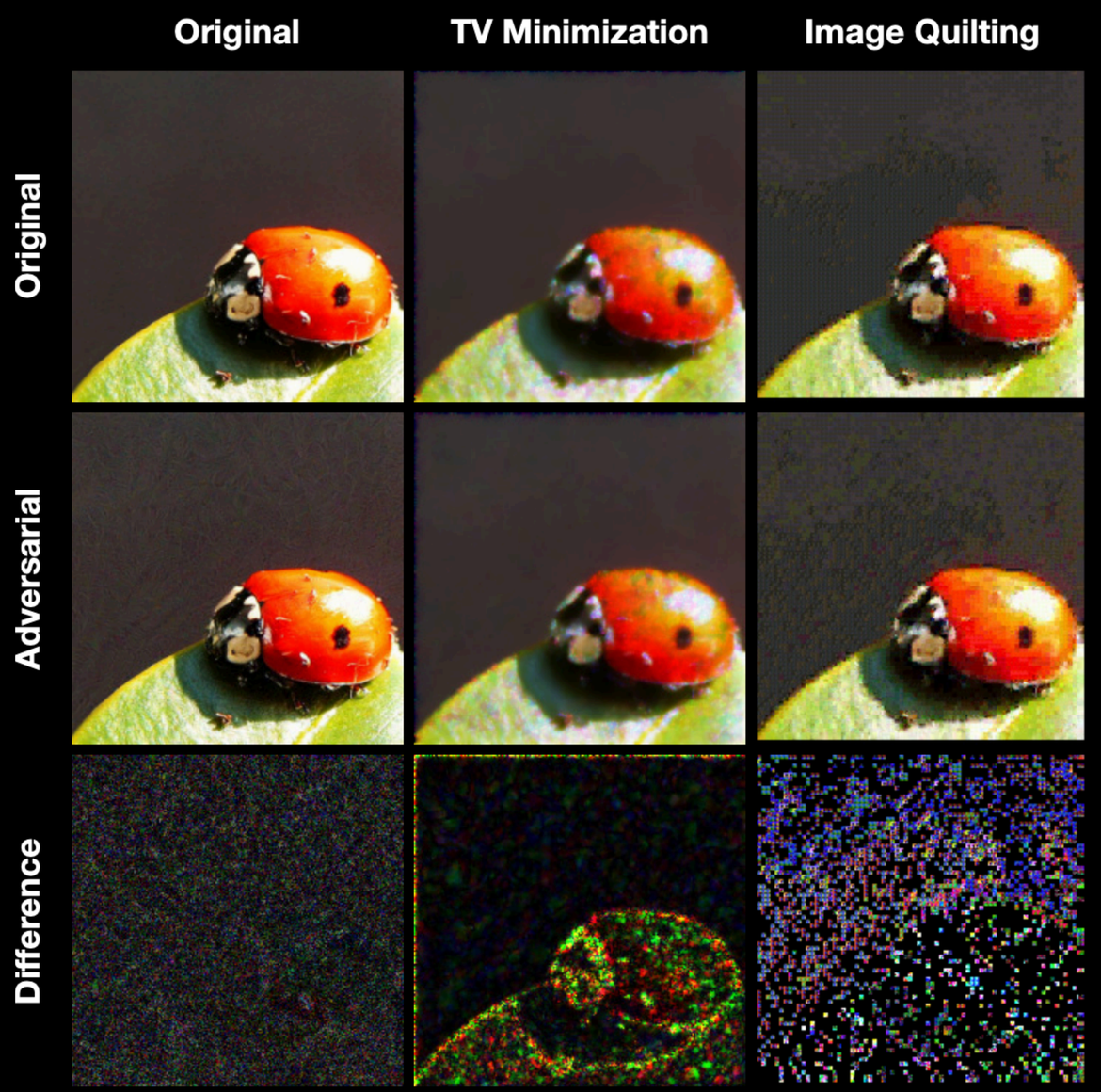


Alternate Defences 2

Preprocessing: Denoising

- Guo et. al. (2018) proposes two alternate methods.
  - **Total Variation Denoising:** Penalise normed difference between adjacent pixels.
  - **Quilting:** Learn a codebook of clean patches, and project image patches to closest clean patch.

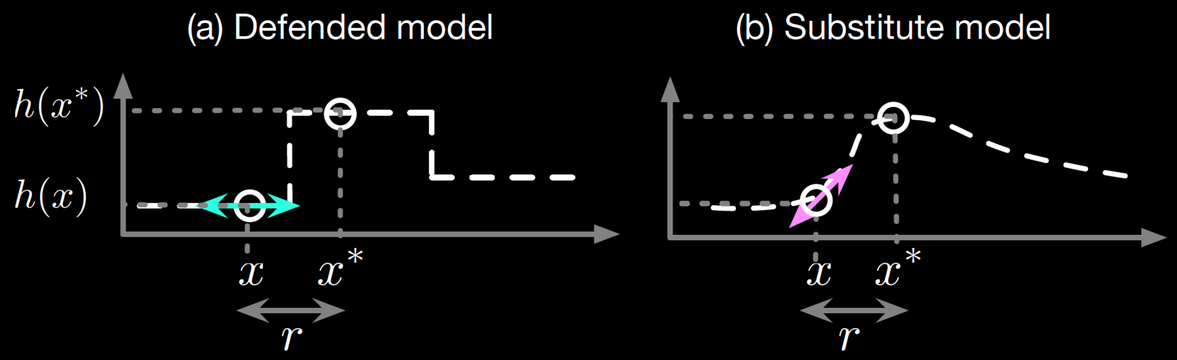
Cons: While effective, all these methods sacrifice image quality.



Alternate Defences 2

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.

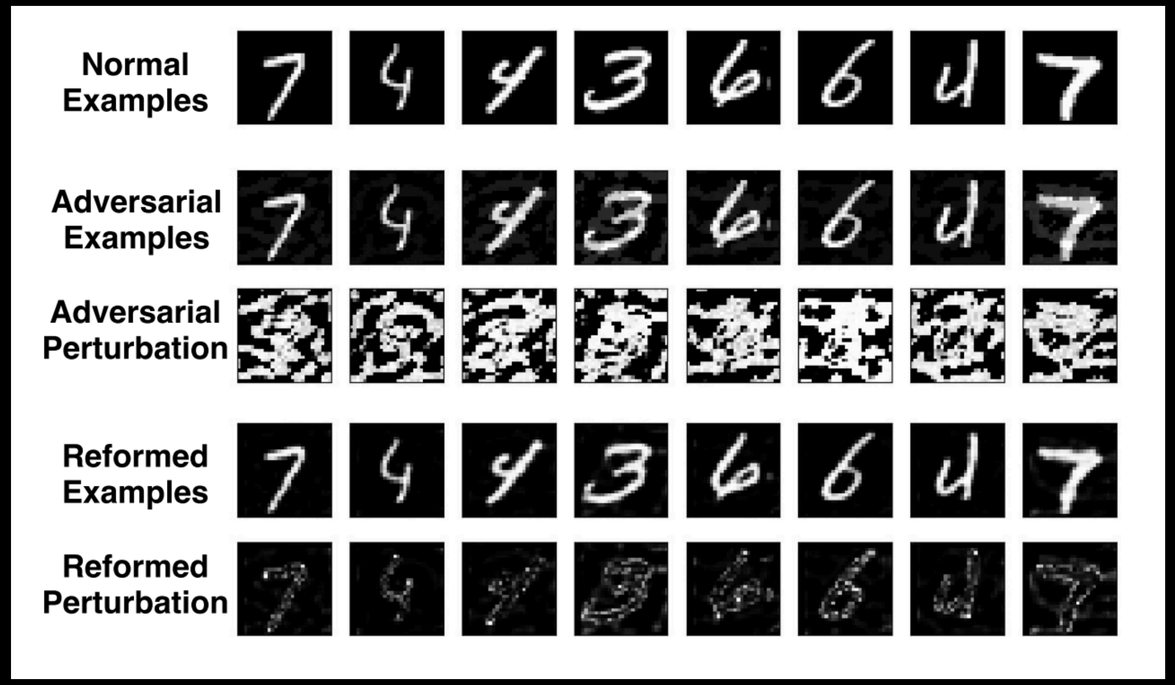


Alternate Defences 3

Ensemble Defence

- **Idea:** It is harder to find attacks that apply to multiple models, especially if they are complementary.
- Leveraged in **ensemble defence**: combined ensembles of models which work together.
- MagNet: Model agnostic. Uses separate detection network to detect attacks and “reform” them before passing to final model.

Cons: Naturally very expensive.

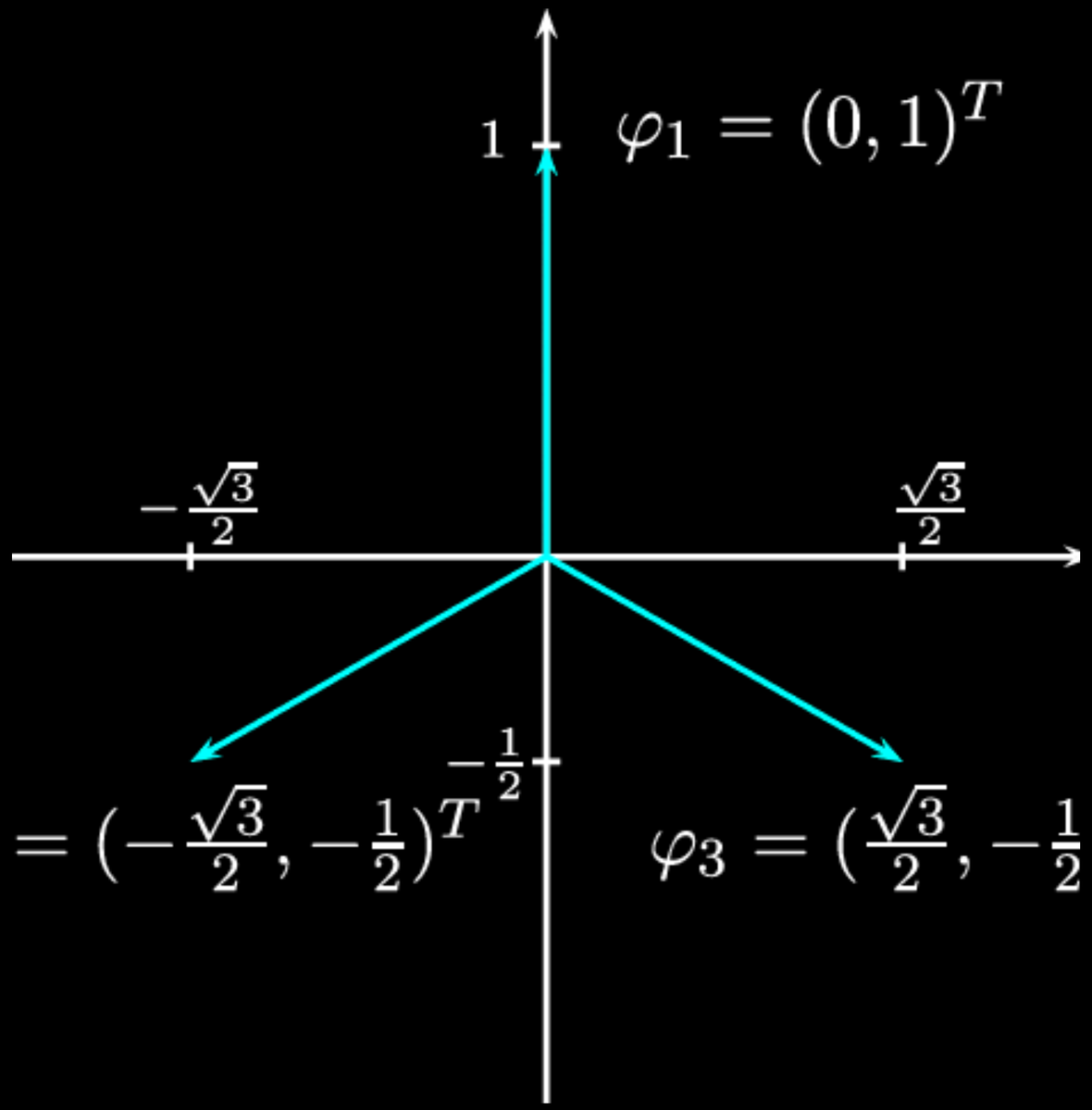


Alternate Defences 4

Regularisation

- **Parseval Networks:** Encourages weights to be approx. orthonormal by adding $\alpha\|W^TW - I\|_2$ to loss term.
- **Spectral Normalisation:** 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 4

Certified Defences

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

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

Pro: Theoretically sound.

Con: Still expensive.

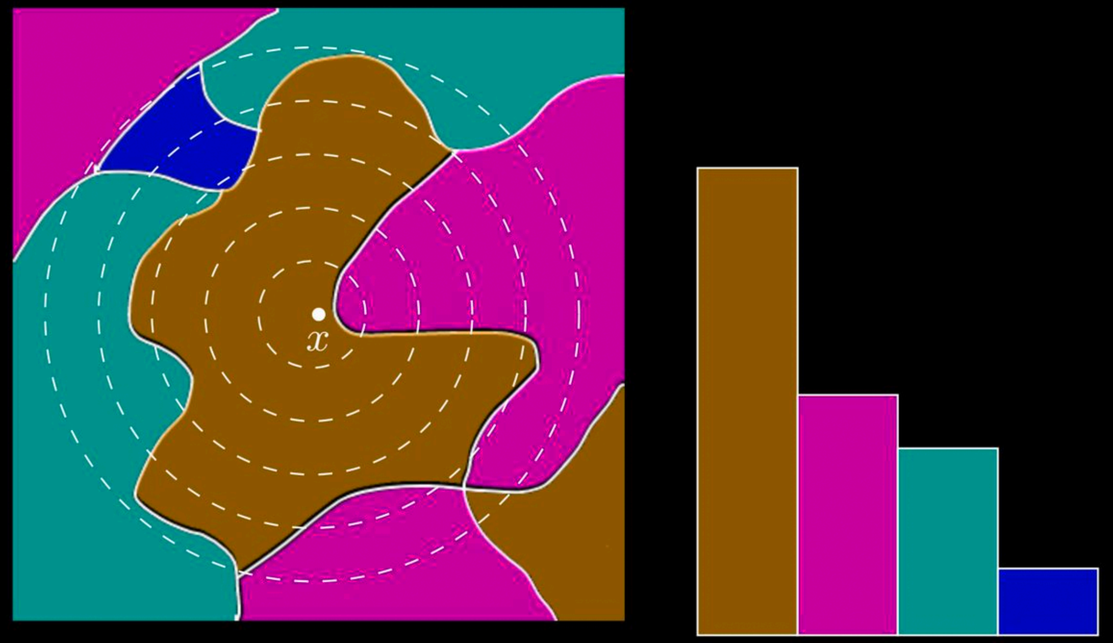
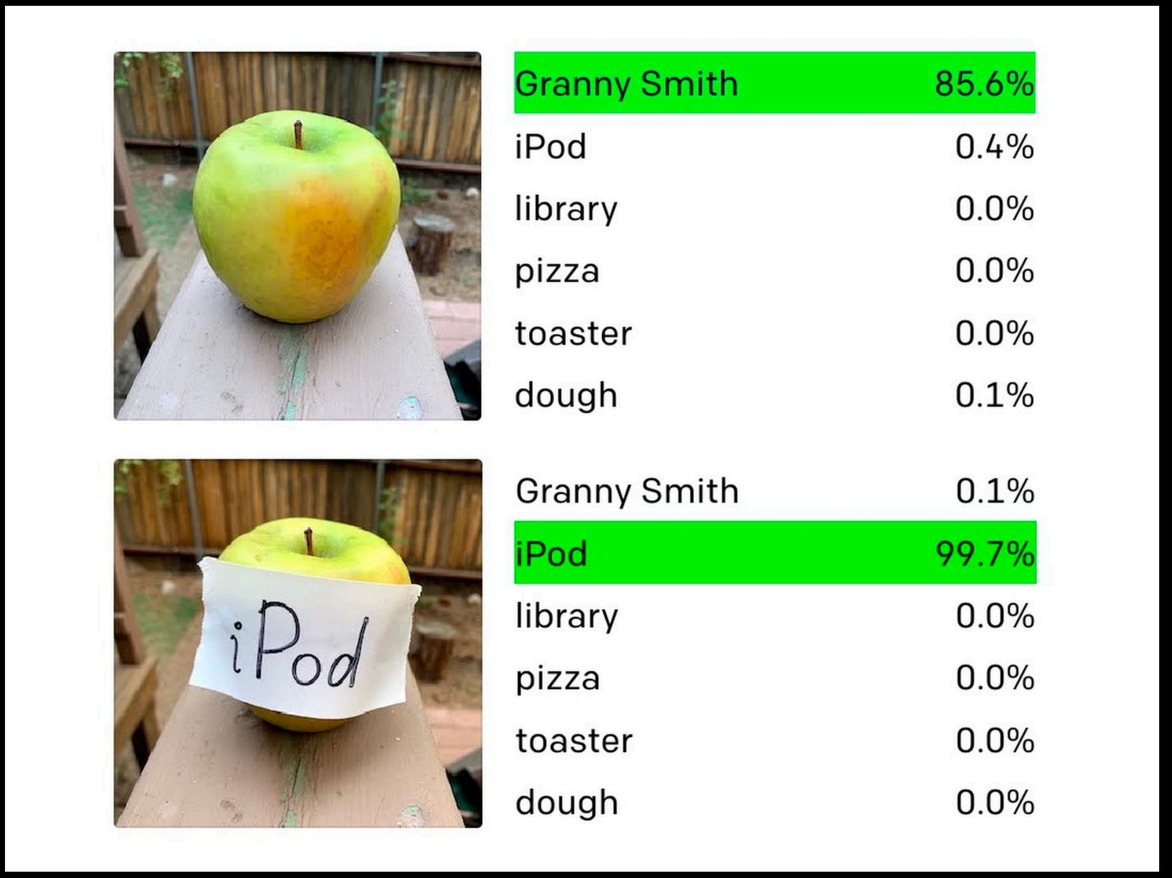


Figure 1. Illustration of randomized smoothing. **Left**: the decision regions of the base classifier f are drawn in different colors. The dotted lines are the level sets of the distribution $\mathcal{N}(x, \sigma^2 I)$. **Right**: the distribution $f(\mathcal{N}(x, \sigma^2 I))$. Here, $g(x)$ is "blue."

Strategies: Summary

Adversarial Defence Strategies

- Classic regularisation, quantisation, 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.
- Try to have a clear understanding of the **taxonomy** of defences and how they work.



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

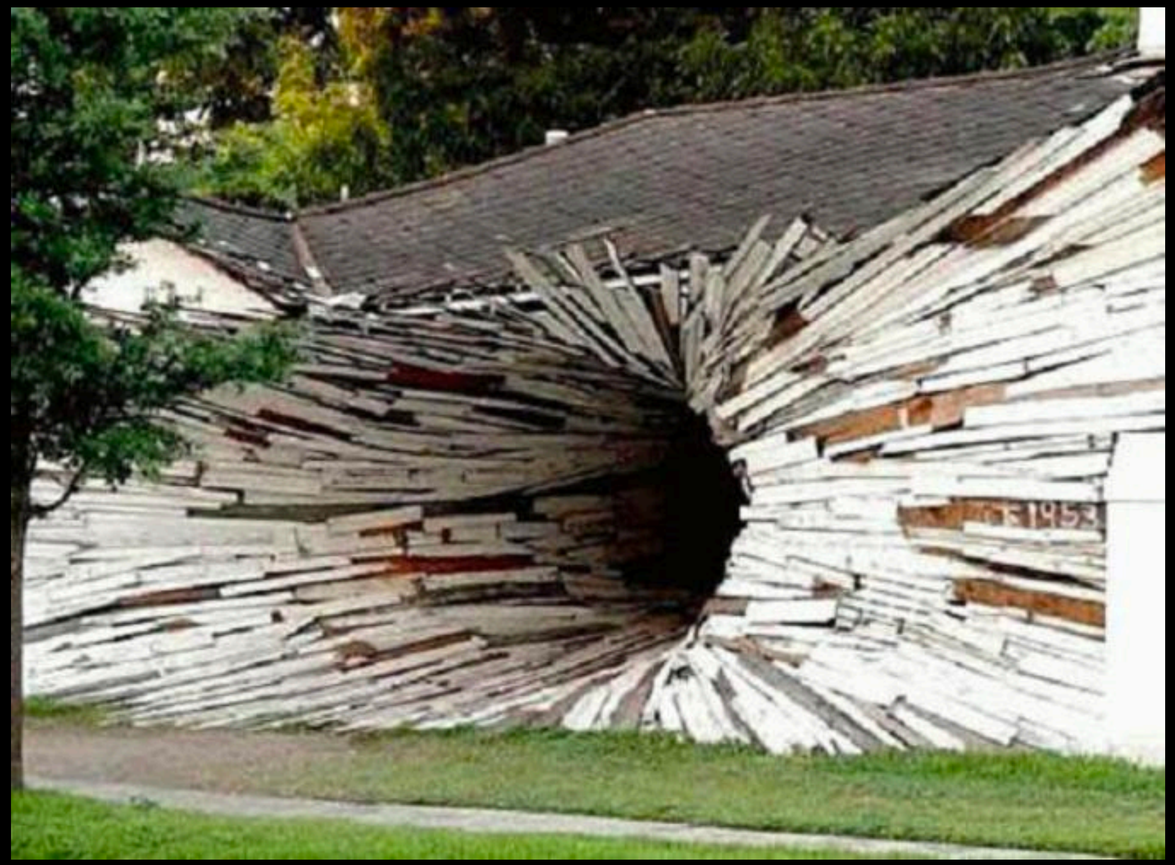
Architectural Particularites

CNN Robustness

Consider these four points:

1. Sing. values ς_i of convolution are its Fourier coefficients*.
2. **Riemann-Lebesgue lemma**: Fourier coefficients of an L^1 signal decay to zero as frequency increases.
3. SGD on linear operators converges to the least squares solution.
4. Variance of least squares estimate*: $\mathrm{Var}[\hat{W}] = \sum_n \sigma^2/\varsigma_n^2$.

Q: What does this imply?



* Assuming target $(y_i)_{i=1}^n$ are iid. with constant variance σ^2.

Architectural Particularites

ViT Robustness

- It is still a little early for in-depth discussion on Vision Transformer models (ViTs).
- For the curious minded student:

[Understanding Robustness of Transformers for Image Classification]

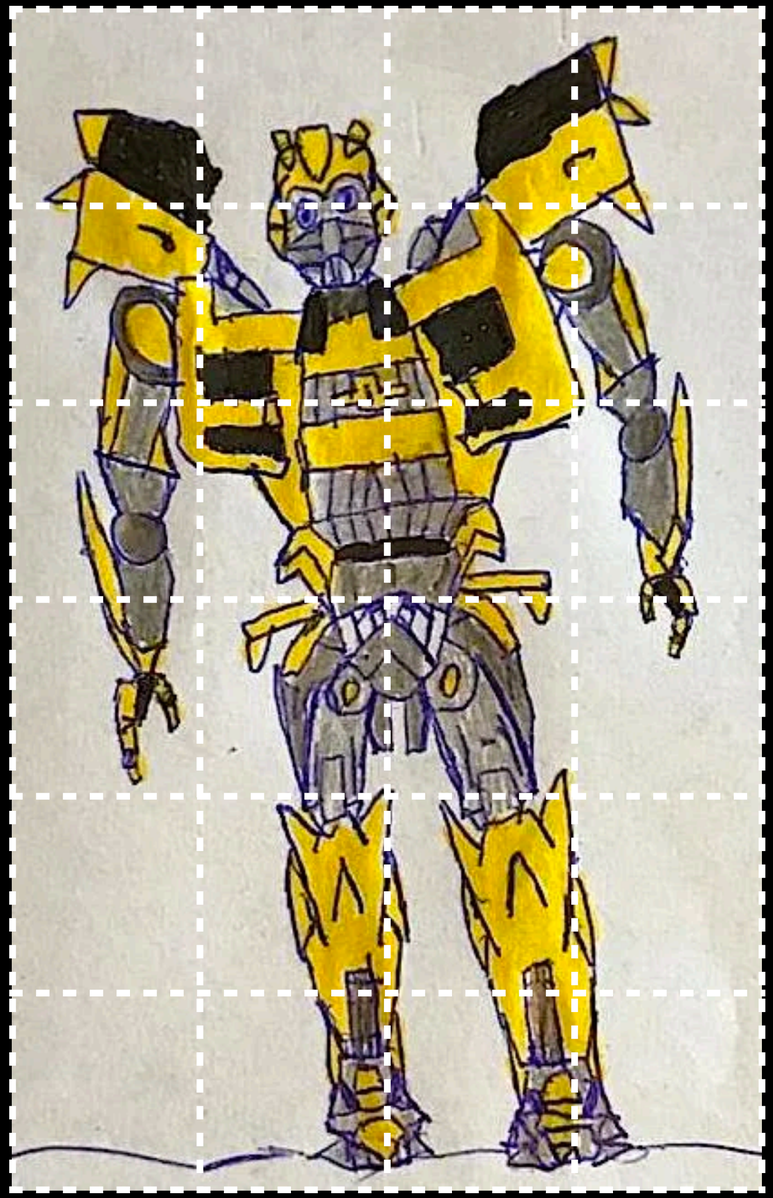


Image credit: V.B.B.B.Drawings - opensea.io