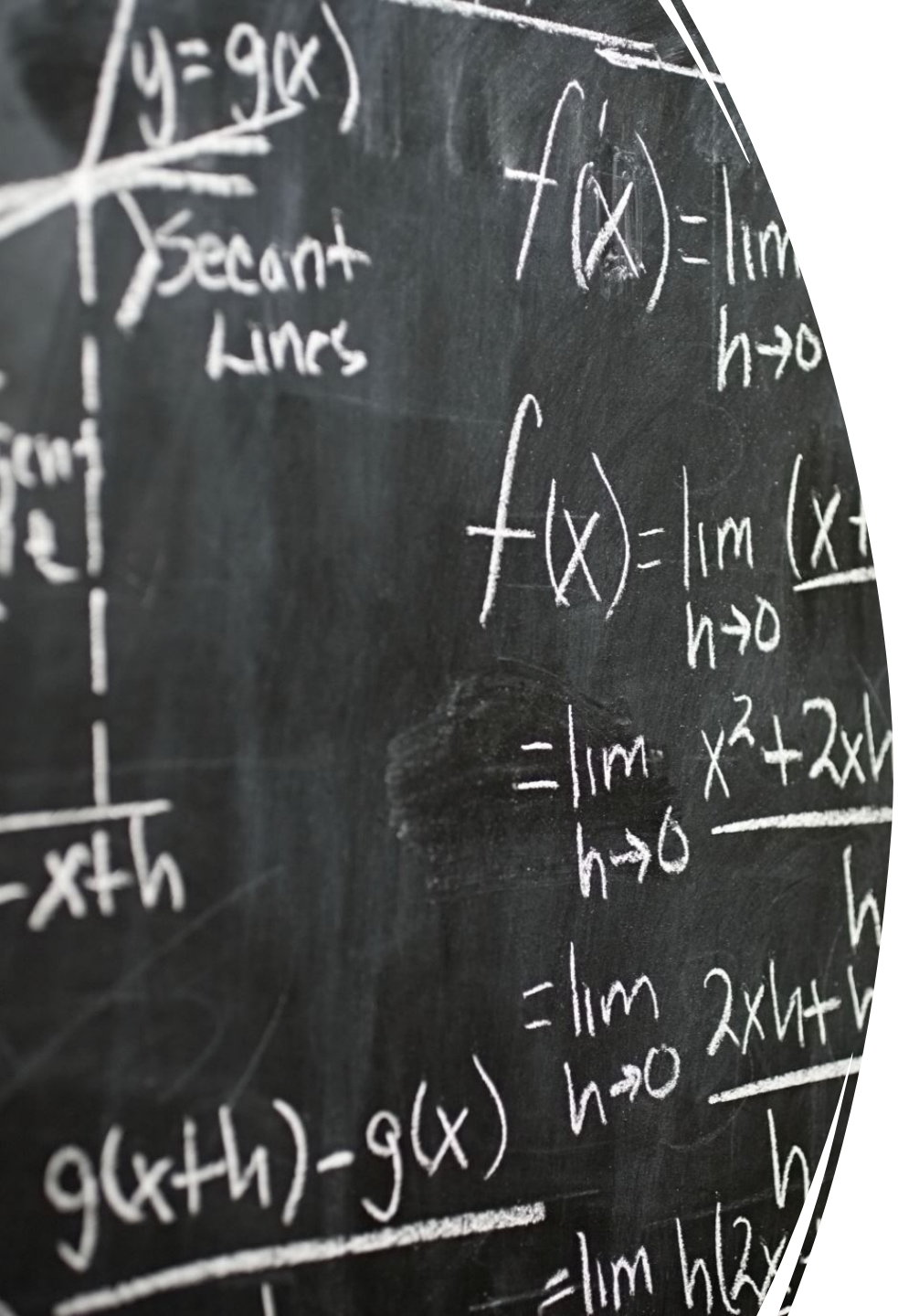


Lecture 14 -- Adversarial Attack on ML models

Prof Xiaowei Huang

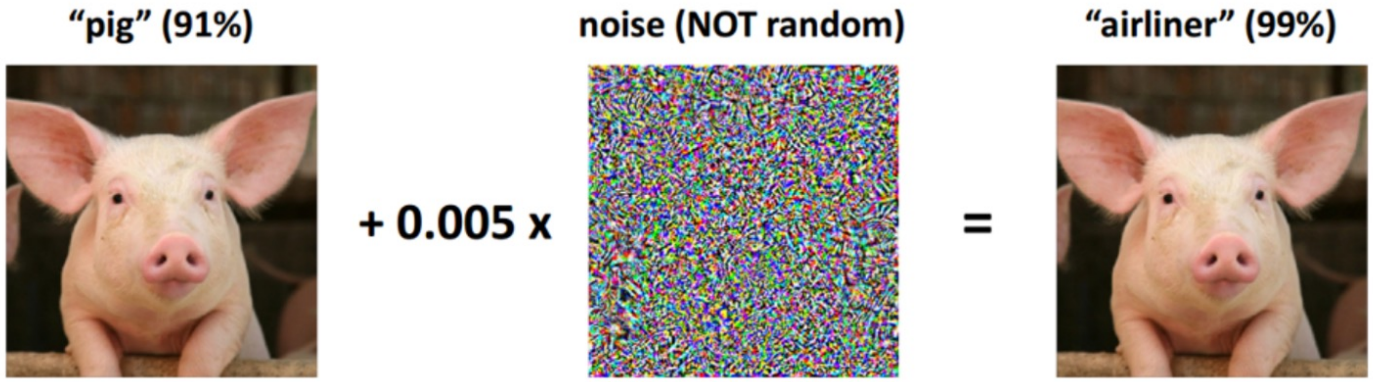
https://cgi.csc.liv.ac.uk/~xiaowei/

(Attendance Code: **283149**)

Topics

1. Attack on Decision Tree
2. Attack on K-nn
3. Attack on Logistic Regression
4. Attack on Deep Learning

Illustration of Adversarial Attack



What is Adversarial Example

Input:

- A machine learning model f
- A genuine example x

Aim: find a perturbed example x', such that

- f produces a different decision on x'
- x' is close to x

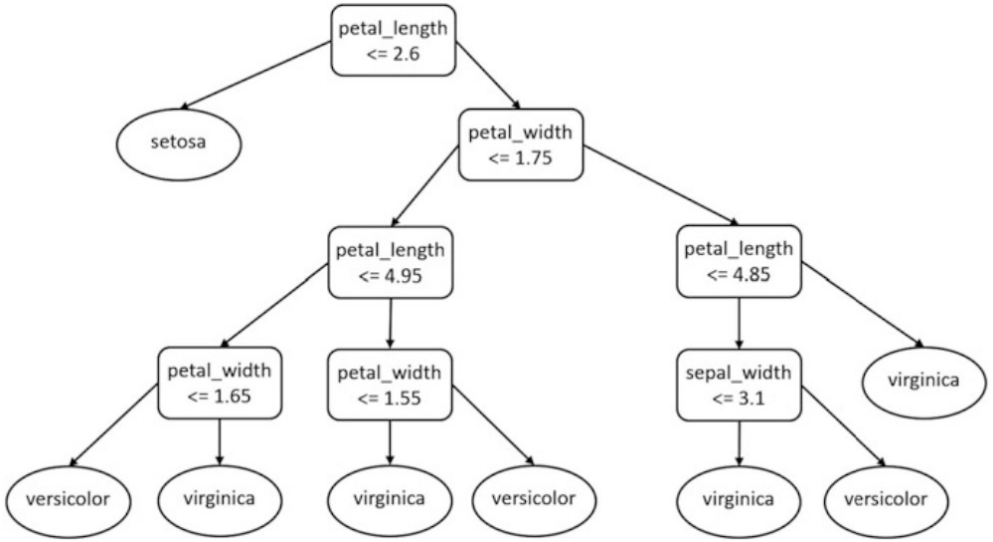
Completeness

- A complete approach can find an adversarial example if there is any

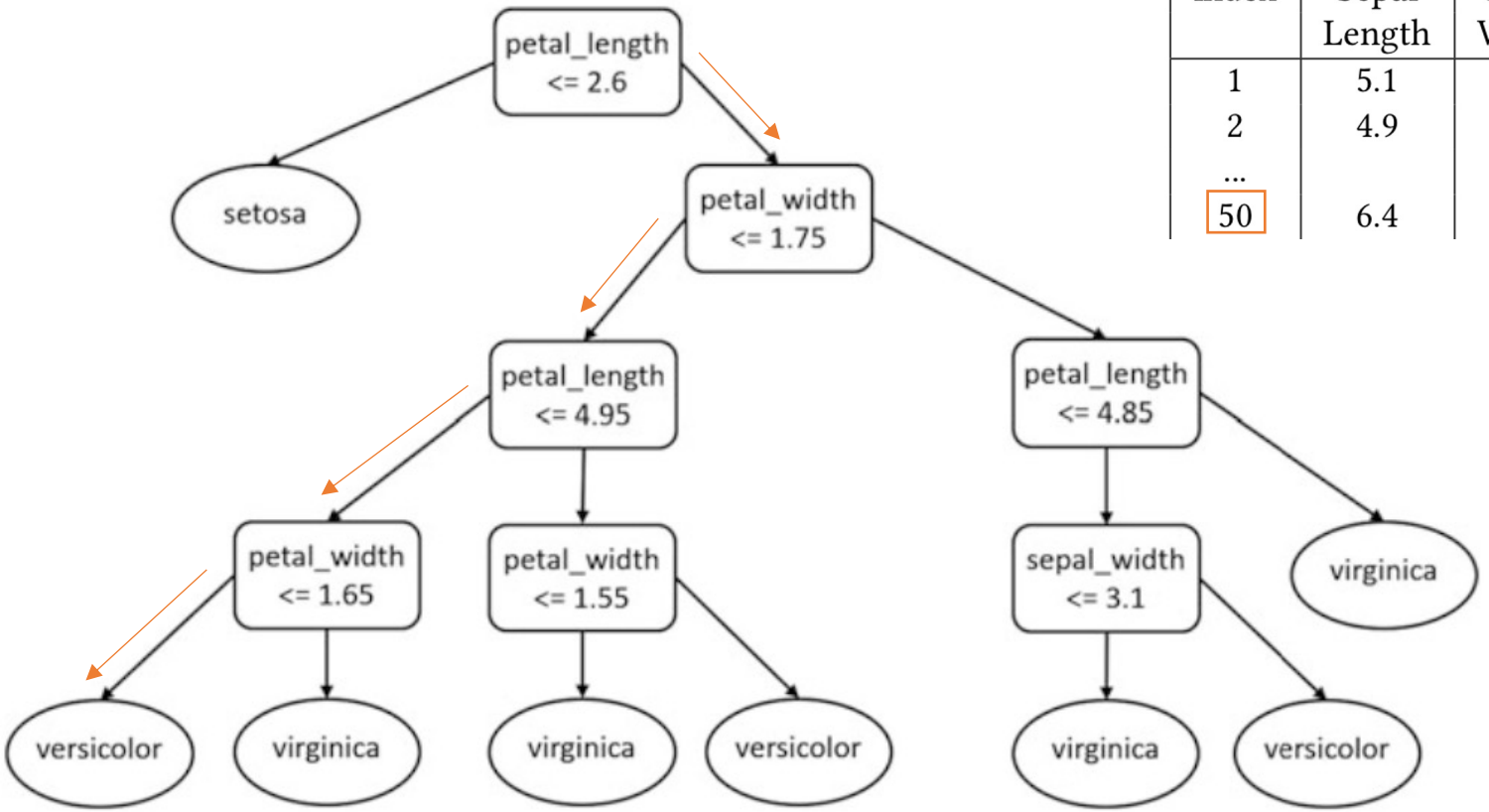
Optimality

- x' that is closest to x and with a different decision

Adversarial Example for Decision Tree



Example



index	Sepal Length	Sepal Width	Petal Length	Petal Width	Class Label
1	5.1	3.5	1.4	0.2	iris setosa
2	4.9	3.0	1.4	0.2	iris setosa
...					
50	6.4	3.5	4.5	1.2	iris versicolor

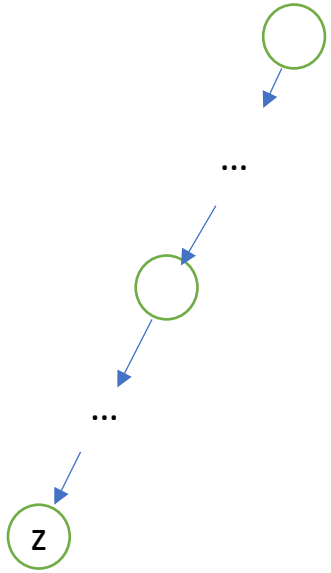
Adversarial attack for decision tree

Given an instance **x** and a decision tree:

The algorithm proceeds with the following steps:

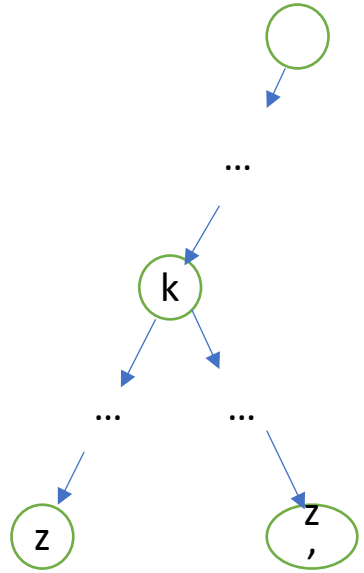
1. Given an input $\mathbf{x}$, it will lead to some leaf node z with label y.
2. Consider a targeted label $y' \neq y$, we find the shortest path from z to any leaf node with label y'. Let the new leaf node be z'.
3. Then, we can identify the common ancestor of z and z' on the shortest path, and construct a path from the root node to the common ancestor and then to z'.
4. Construct an input $\mathbf{x}'$ from the constructed path such that $||\mathbf{x} - \mathbf{x}'||$ is minimised. If $||\mathbf{x} - \mathbf{x}'|| < \delta$ then we return $\mathbf{x}'$ as an adversarial example.

Illustration of Adversarial Attack Algorithm



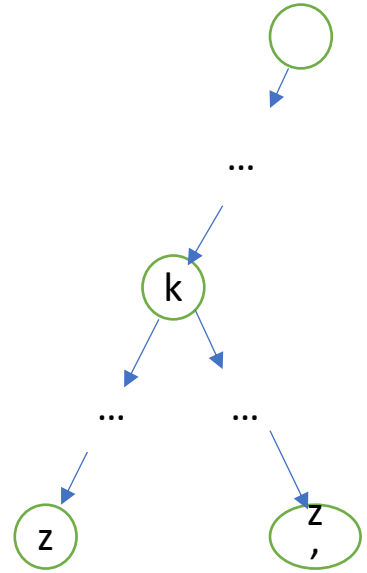
Label=y

(Step 1): get leaf node z with label y for input x



Label=y Label=y'

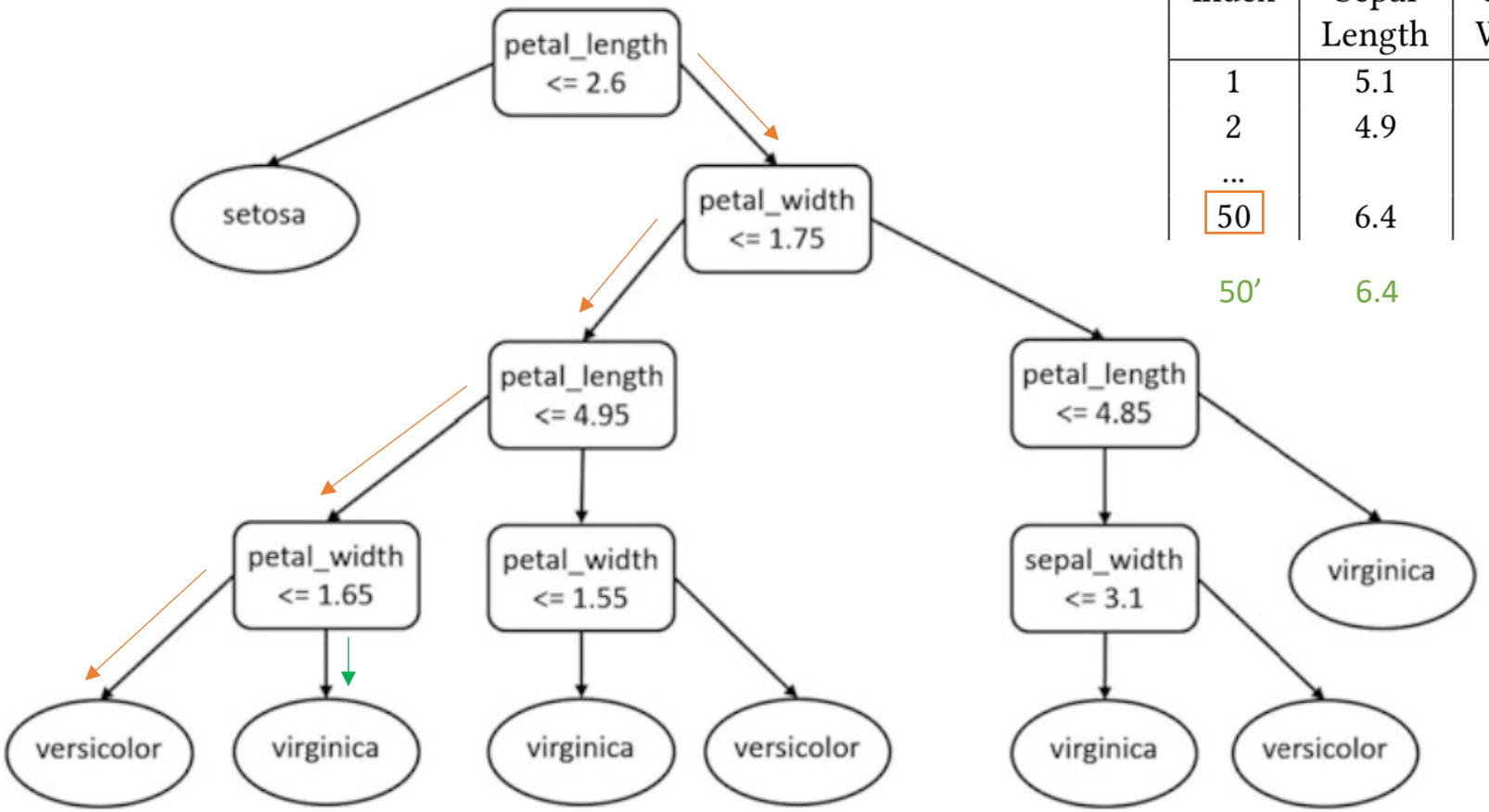
(Steps 2-3): find shortest path from z to any node with label y'. Let k be the common ancestor



Label=y Label=y'

(Step 4) Get adversarial example from root to z' through k

Example



index	Sepal Length	Sepal Width	Petal Length	Petal Width	Class Label
1	5.1	3.5	1.4	0.2	iris setosa
2	4.9	3.0	1.4	0.2	iris setosa
...					
50	6.4	3.5	4.5	1.2	iris versicolor
50'	6.4	3.5	4.5	1.66	iris virginica

Class change

x' is close to x

Make petal_width=1.66

Adversarial attack for decision tree

Is **x'** an adversarial example?

- Yes, because **x'** follows a path from the root node to the leaf node **z'**, which is labelled as y', different from y.

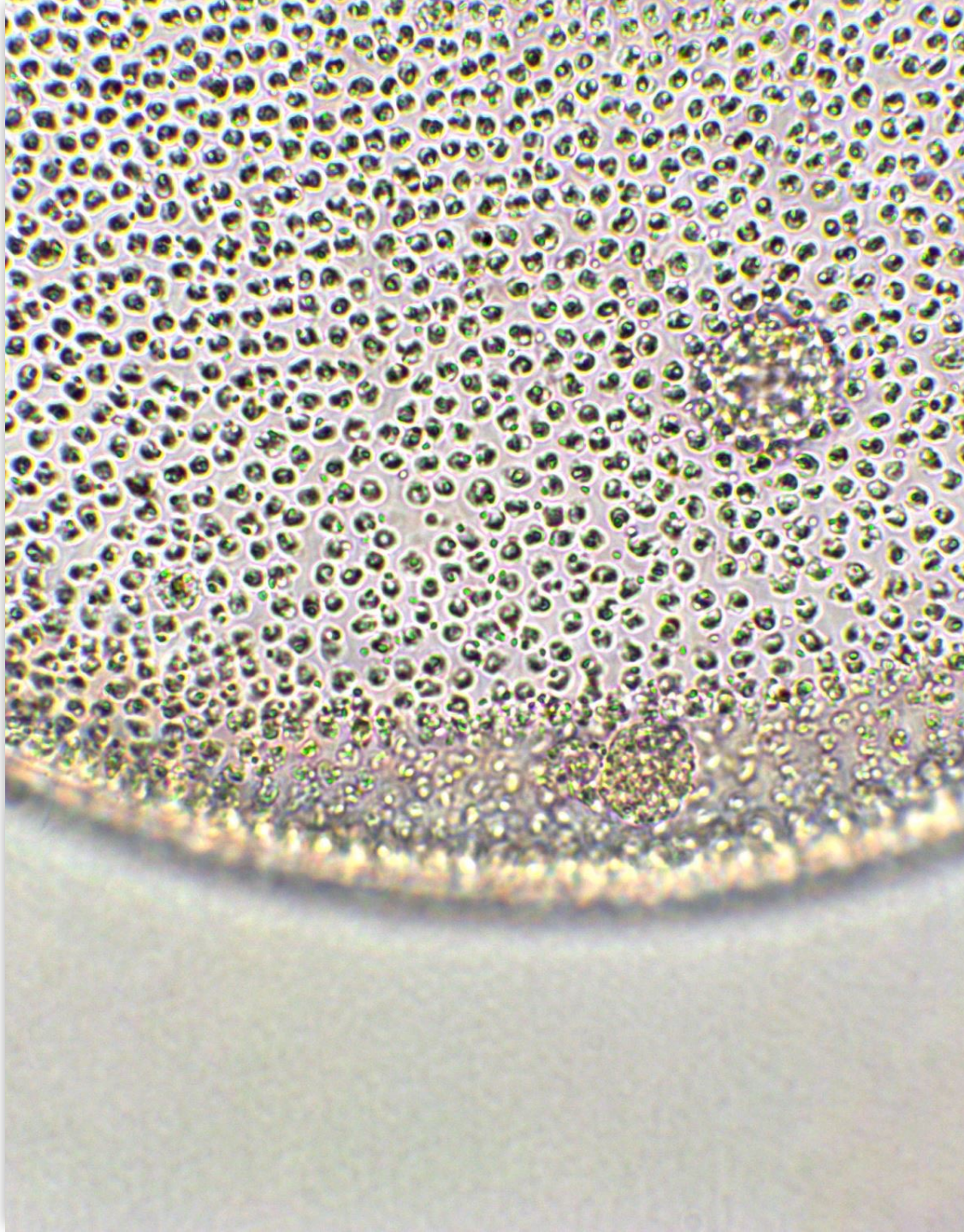
Is this approach complete?

- A complete approach is able to find an adversarial example if there is any. Unfortunately, the above algorithm is incomplete.

Sub-optimality

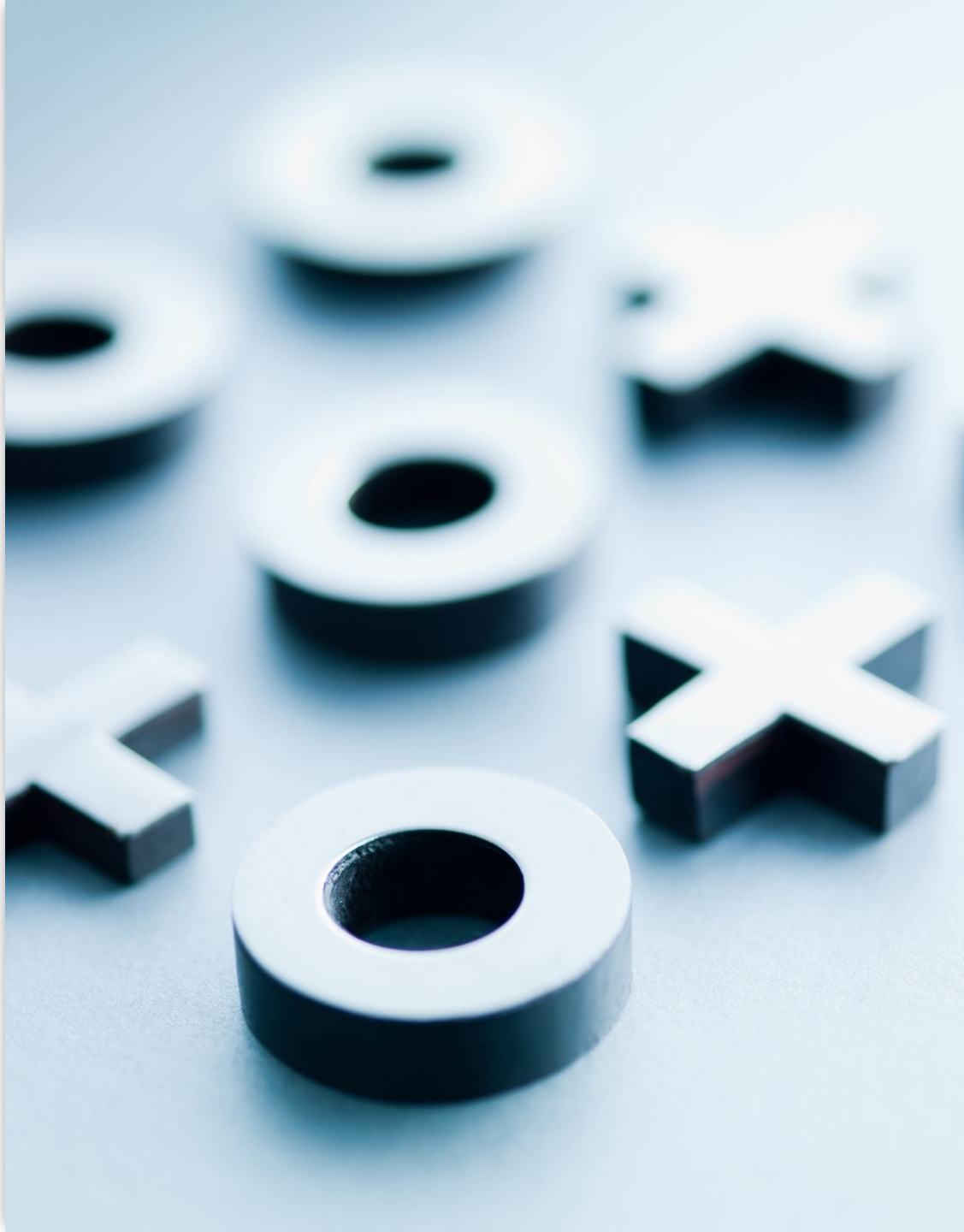
- This algorithm is also sub-optimal, i.e., the found **x'** does not necessarily be the optimal solution to the optimisation problem.

Adversarial Example for K-nn

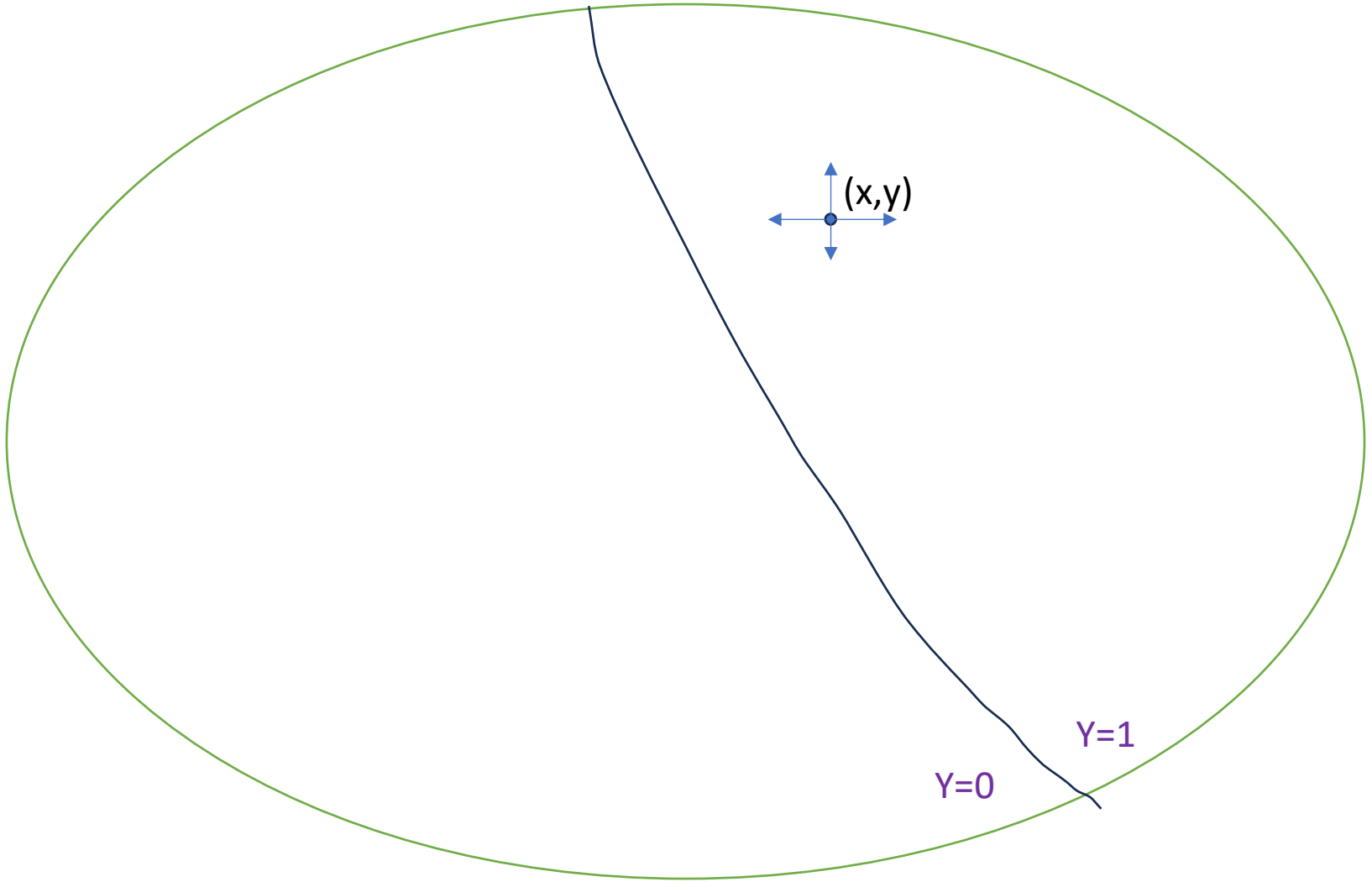


Adversarial attack for K-nn

- Assume that we have a data instance with label, (**x**,y), and want to find another one (**x'**,y') that is close to **x**.
- The general idea is to move the instance **x** gradually along the direction where the probability of being classified as y decreases the most, until the class change.



Example



along the direction where the probability of being classified as y decreases the most, until the class change

Define

Step 1

$$\nabla_i^s f(\mathbf{x}) = \frac{f(\mathbf{x}_i^s) - f(\mathbf{x})}{||\mathbf{x} - \mathbf{x}_i^s||}$$

for all I and s, where i is the dimension and s is the direction

Adversarial attack for K-nn

First of all, we define

$$\mathbf{x}_i^s = \begin{cases} \mathbf{x} + \epsilon_i & \text{if s=+} \\ \mathbf{x} - \epsilon_i & \text{if s=-} \end{cases} \tag{3.19}$$

where ϵ_i is a 0-vector except for the entry for feature X_i, which has value $\epsilon > 0$. Then, the gradient along the direction defined by X_i and $s \in \{+, -\}$ is as follows:

$$\nabla_i^s f(\mathbf{x}) = \frac{f(\mathbf{x}_i^s) - f(\mathbf{x})}{||\mathbf{x} - \mathbf{x}_i^s||} \tag{3.20}$$

Then, for the next step, we move $\mathbf{x}$ along the following direction to $\mathbf{x}'$:

Step 2

$$\arg\min_{i,s} \nabla_i^s f(\mathbf{x}) \tag{3.21}$$

and check if there is a misclassification on $\mathbf{x}'$. We repeat the above move until there is a misclassification.

the direction that decreases the most

Adversarial attack for K-nn

Is **x'** an adversarial example?

- Yes, because the final **x'** is obtained by following a sequence of changes until witnessing a misclassification.

Is this approach complete?

- Unfortunately, the above algorithm is incomplete.

Sub-optimality

- The resulting **x'** does not necessarily be the optimal solution to the optimisation problem.

Adversarial Example for Logistic Regression

Adversarial Example for Logistic Regression

- Considering a binary classification problem, i.e., $y \in \{+1, -1\}$ for any data instance **x**, the loss for a single instance (**x**,y) is

$$\hat{L}(f_{\mathbf{w}}, (\mathbf{x}, y)) = -\, y \log \sigma(\mathbf{w}^T \mathbf{x}) - (1 - y) \log[1 - \sigma(\mathbf{w}^T \mathbf{x})]$$

- The computation of adversarial example is to solve the following optimisation problem:

$$\text{maximise}_{||\delta|| \leq \epsilon} \quad \hat{L}(f_{\mathbf{w}}, (\mathbf{x} + \delta, y))$$

which finds the greatest loss within the constraint over the perturbation δ

Adversarial Example for Logistic Regression

$$\hat{L}(f_{\mathbf{w}}, (\mathbf{x}, y)) = -\, y \log \sigma(\mathbf{w}^T \mathbf{x}) - (1 - y) \log[1 - \sigma(\mathbf{w}^T \mathbf{x})]$$

Step 1

- Note that, this is equivalent to

$$\text{minimise}_{||\delta|| \leq \epsilon} \quad y\mathbf{w}^T \delta$$

Step 2

- Now, considering the L_∞ norm, i.e., $||\delta||_\infty \leq \epsilon$, the optimal solution is

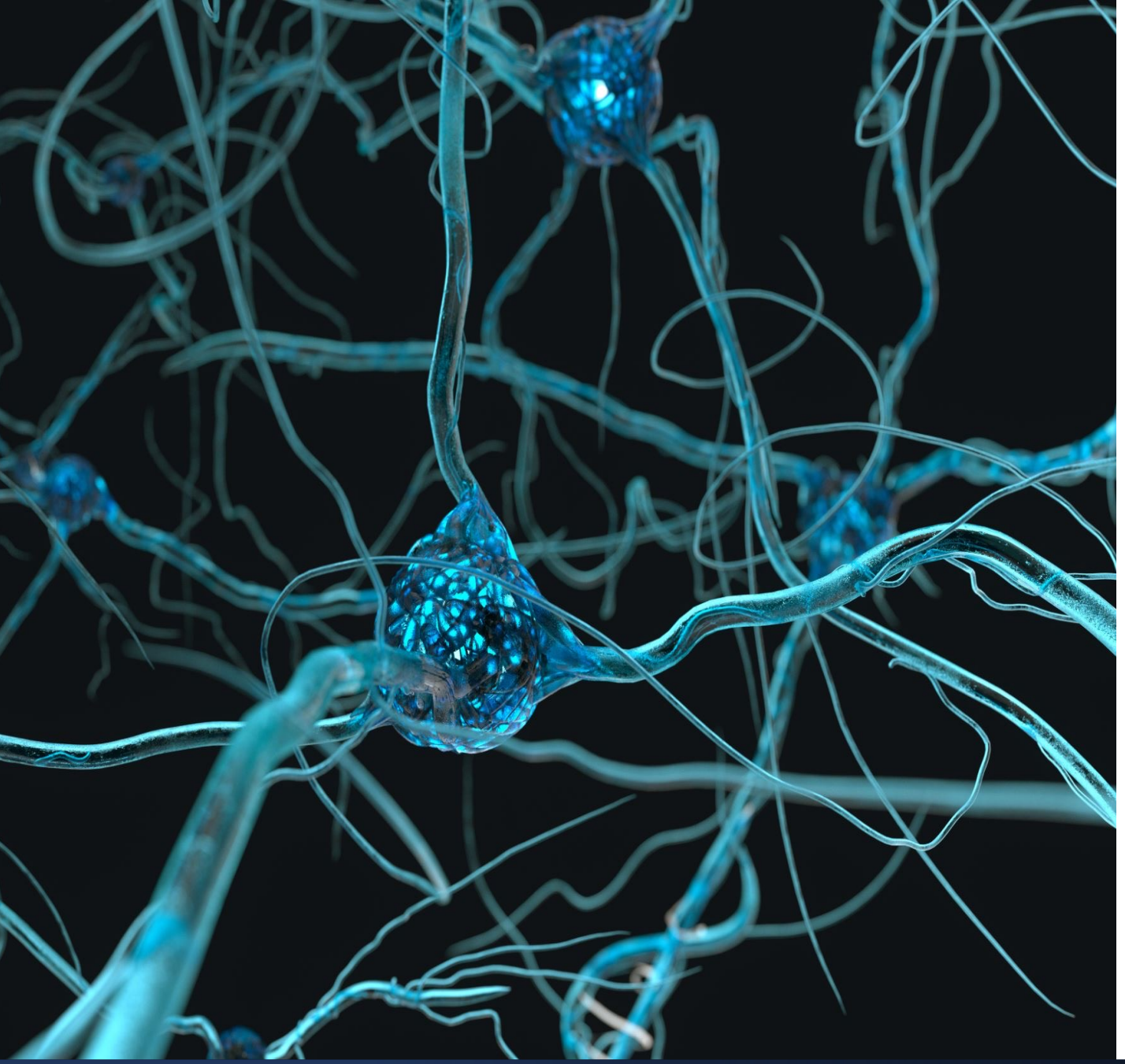
$$\delta^* = -y\epsilon sign(\mathbf{w})$$

All components are known! So, we can compute analytically.

Step 3

where *sign* is the sign function extracting the sign of real numbers, such that

$$sign(w) = \begin{cases} -1 & w < 0 \\ 0 & w = 0 \\ 1 & w > 0 \end{cases}$$

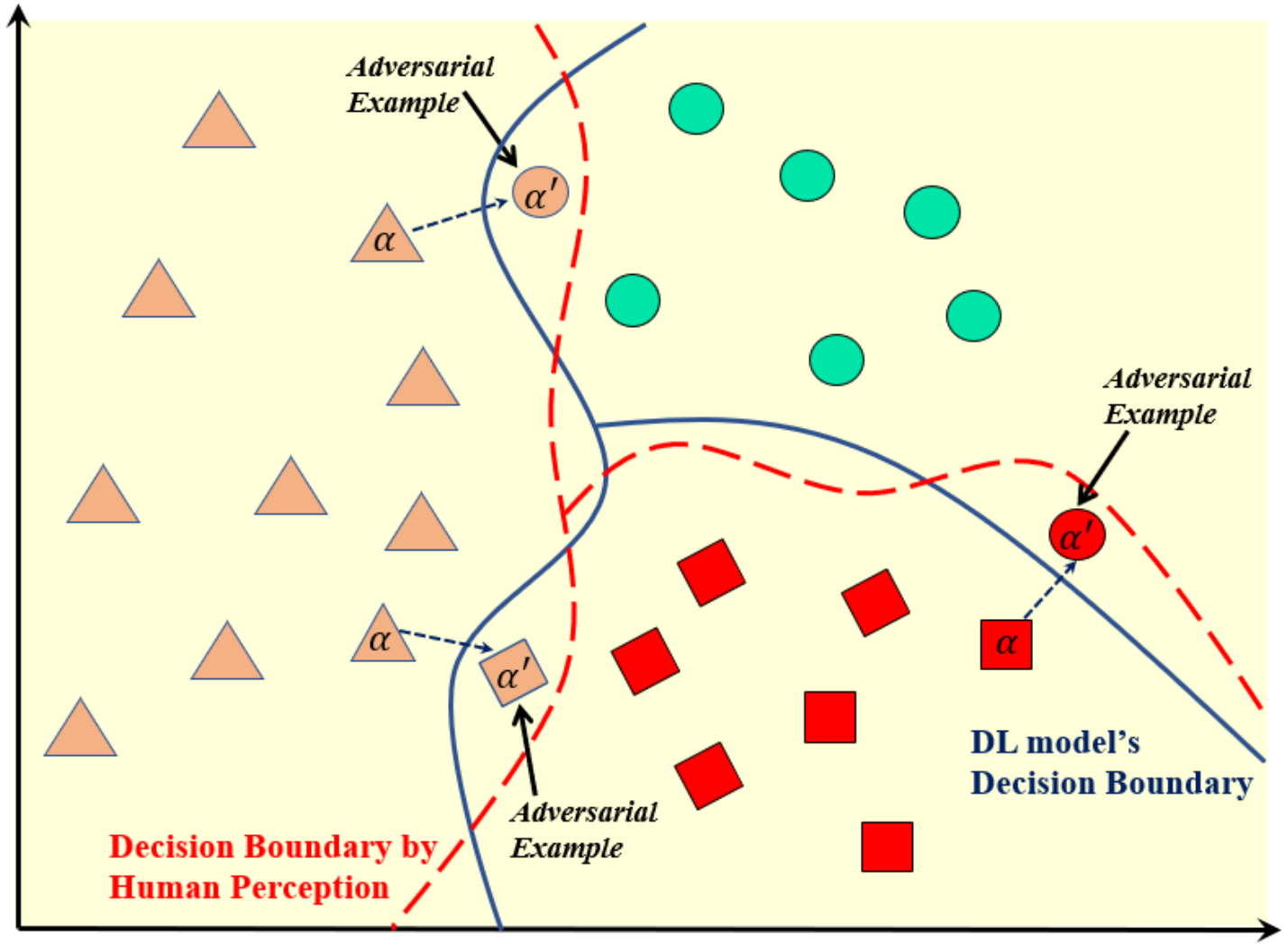


Adversarial Example for Convolutional Neural Networks

Illustration

- DL model: classifies α and α′ **differently**
- Human: should remain the **same**

We have been ignoring the definition of “closeness” by now.



Definition

One of earliest adversarial attack: optimization based formulation with L_2-norm metric

- Model $f : \mathbb{R}^{s_1} \to \{1 \dots s_K\}$ with s_K labels
- $x \in \mathbb{R}^{s_1} = [0, 1]^{s_1}$ is an input
- $t \in \{1 \dots s_K\}$ is a target misclassification label

Find the adversarial perturbation r via

$$\begin{aligned} & \min ||r||_2 && \text{assure human-decision unchanged} \\ s.t. \quad & \arg\max_l f_l(x + r) = t && \text{assure misclassification} \\ & x + r \in \mathbb{R}^{s_1} && \text{assure perturbed image feasible} \end{aligned} \tag{1}$$

So, the ”closeness” of two instances x and x’ is defined over the human’s ability (e.g., perception on images)

Practically, we use L_0, L_1, L_2, L_{infty} norms, but others are also used.

Expressed as constraints

FGSM Attack

Fast Gradient Sign Method is able to find adversarial perturbations with a fixed L_∞-norm constraint **very efficiently**

- θ: the model parameters,
- x, y: the input and the label
- $J(\theta, x, y)$: the loss function

Find adversarial perturbation r by linearizing the loss function around the current value of θ,

$$r = \epsilon \, \text{sign} \left(\nabla_x J(\theta, x, y) \right) \tag{2}$$

- A **one-step modification** to all pixel values to increase the loss function with a L_∞-norm constraint ϵ

Move along the the input gradient sign direction

PyTorch Code for FGSM

https://pytorch.org/tutorials/beginner/fgsm_tutorial.html

PyTorch has library function to compute gradient!

```
# FGSM attack code
def fgsm_attack(image, epsilon, data_grad):
    # Collect the element-wise sign of the data gradient
    sign_data_grad = data_grad.sign()
    # Create the perturbed image by adjusting each pixel of the input image
    perturbed_image = image + epsilon*sign_data_grad
    # Adding clipping to maintain [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    # Return the perturbed image
    return perturbed_image
```

More information

- If you are interested in this topic (and other related topics), you can check

Lecture 14 (Supplementary) -- IJCAI2021 tutorial in Canvas