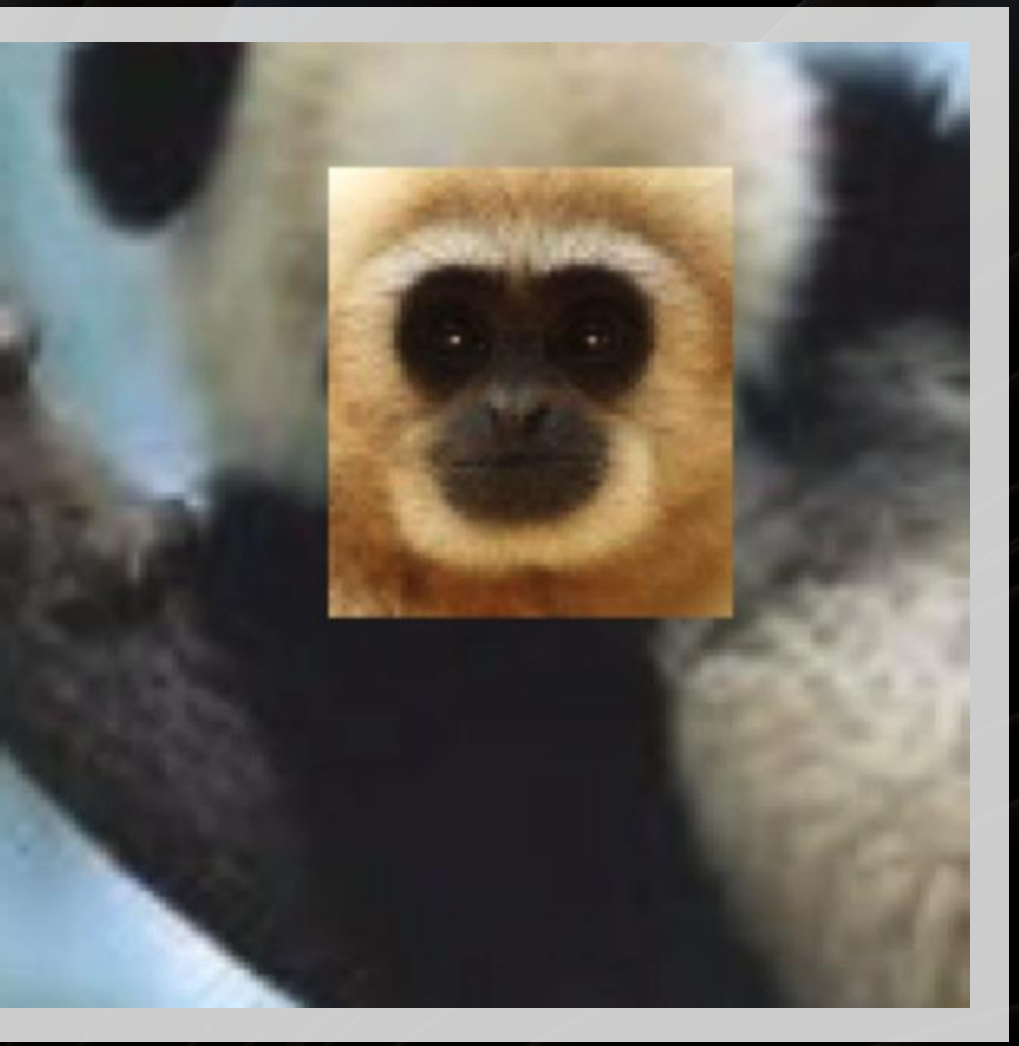


Tutorial of Adversarial Attacks on Machine Learning

Wong Wai Tuck (me@waituck.sg)

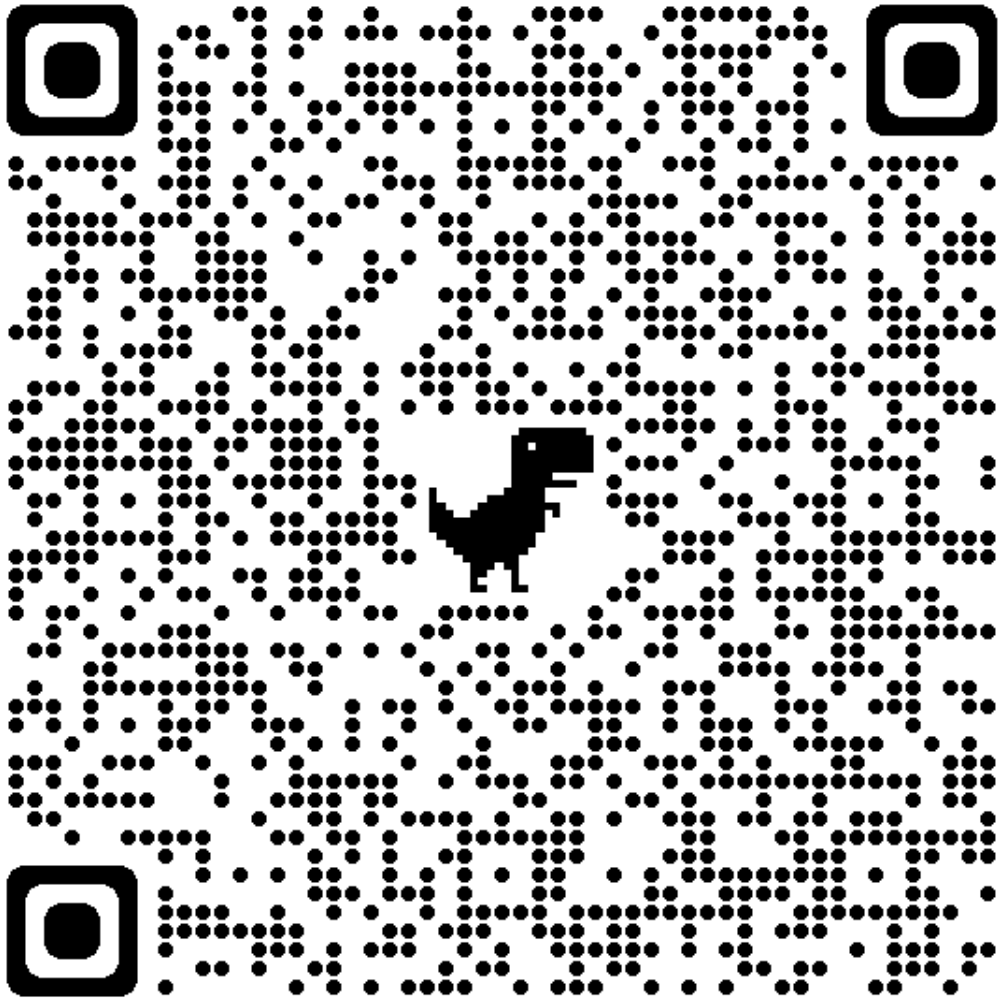
(image reused with permission, credits to Dan Dan Thio)

Credits

- Some slides borrowed from
  - https://adversarial-ml-tutorial.org/adversarial_ml_slides_parts_1_4.pdf
  - https://adversarial-ml-tutorial.org/adversarial_ml_slides_parts_2_3.pdf

Workshop Materials and Exercises Available on Colab

https://colab.research.google.com/github/wongwaituck/adversarial-attack-ml-workshop/blob/main/AdversarialExamplesTutorial.ipynb



whoami?

Wong Wai Tuck

Part Time PhD Student @ SMU

Senior Offsec Engineer @ *<Confidential>*

(we're hiring, PM me for details)

MSc Information Security, Carnegie Mellon University

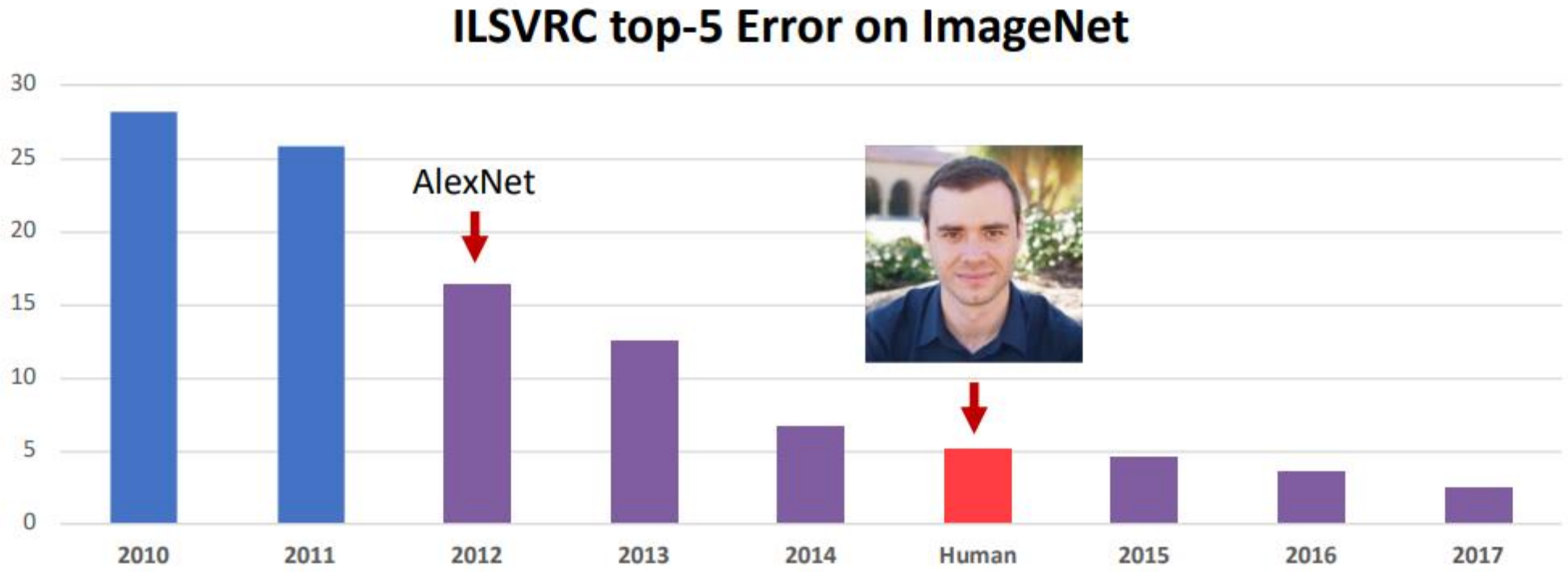
BSc Information Systems/Applied Statistics, SMU

@wongwaituck (LinkedIn/GitHub)

Overview

- Why Machine Learning, and Why Adversarial Examples?
- Background: Machine Learning, Backpropagation
  - **Break**
- Threat Model
- Generating Adversarial Examples
- Defenses
  - **Break**
- Case Study: Neural Hash
- Conclusion
- Q& A

ImageNet: An ML Home Run



But what do these results *really* mean?

Can We Truly Rely on ML?

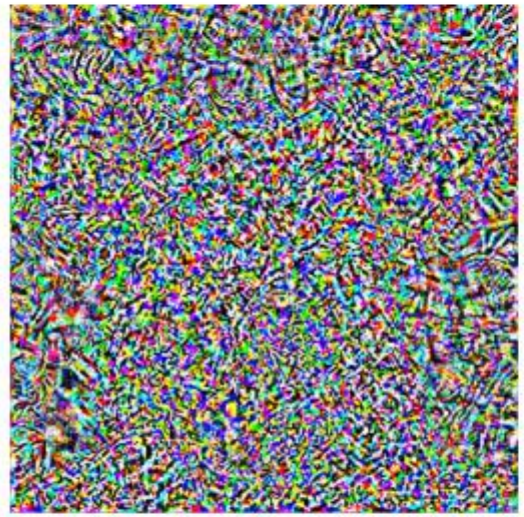
Is ML **truly** ready for real-world deployment?

ML Predictions Are (Mostly) Accurate but Brittle

“pig” (91%)

+ 0.005 x

noise (NOT random)



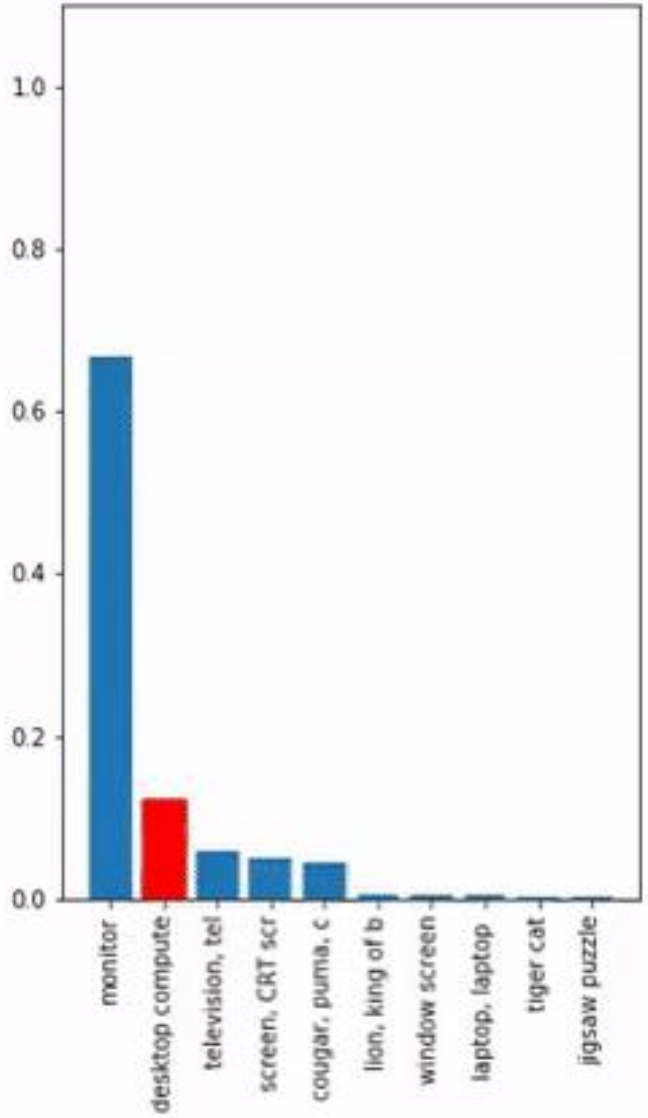
=

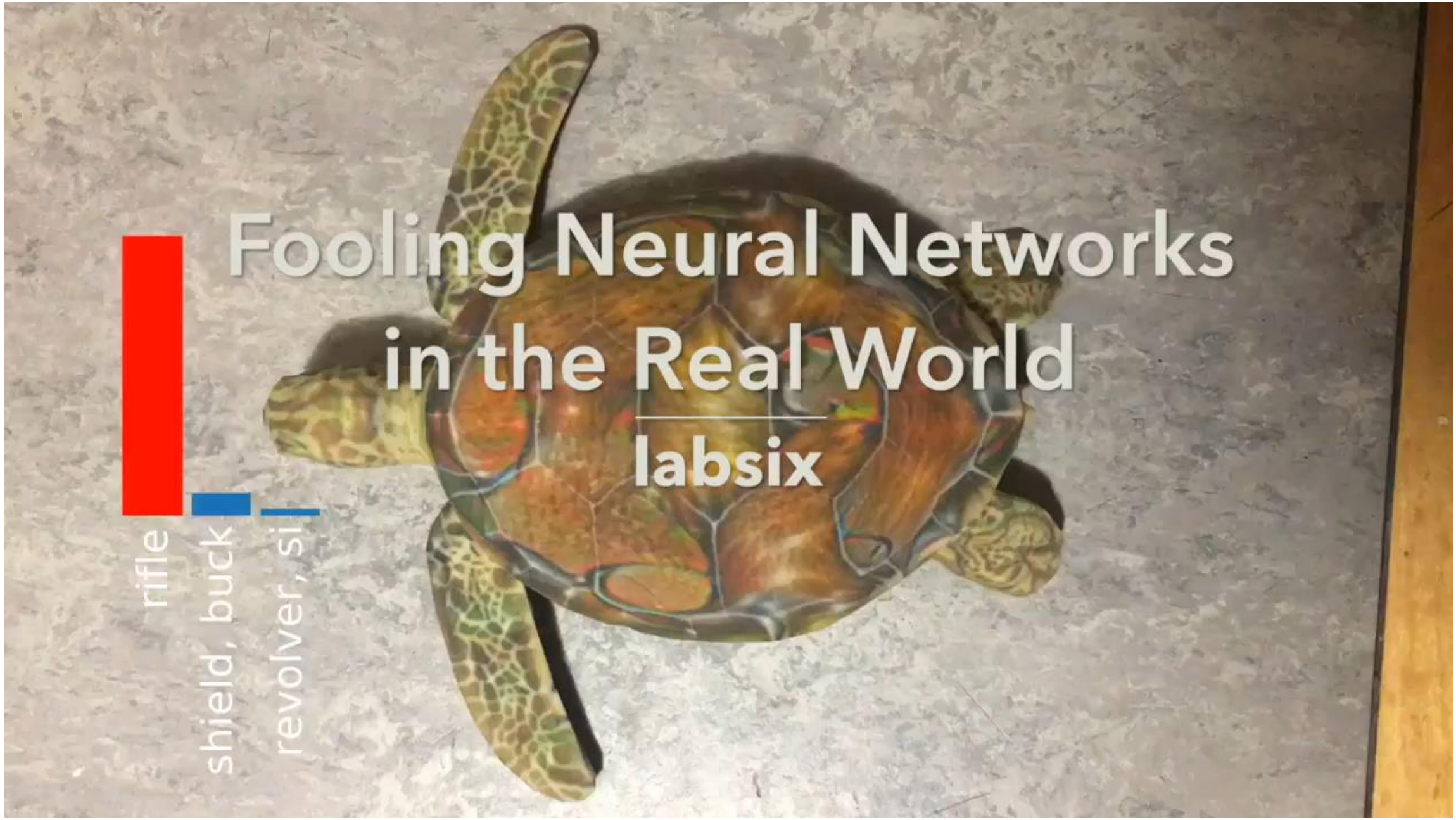
“airliner” (99%)

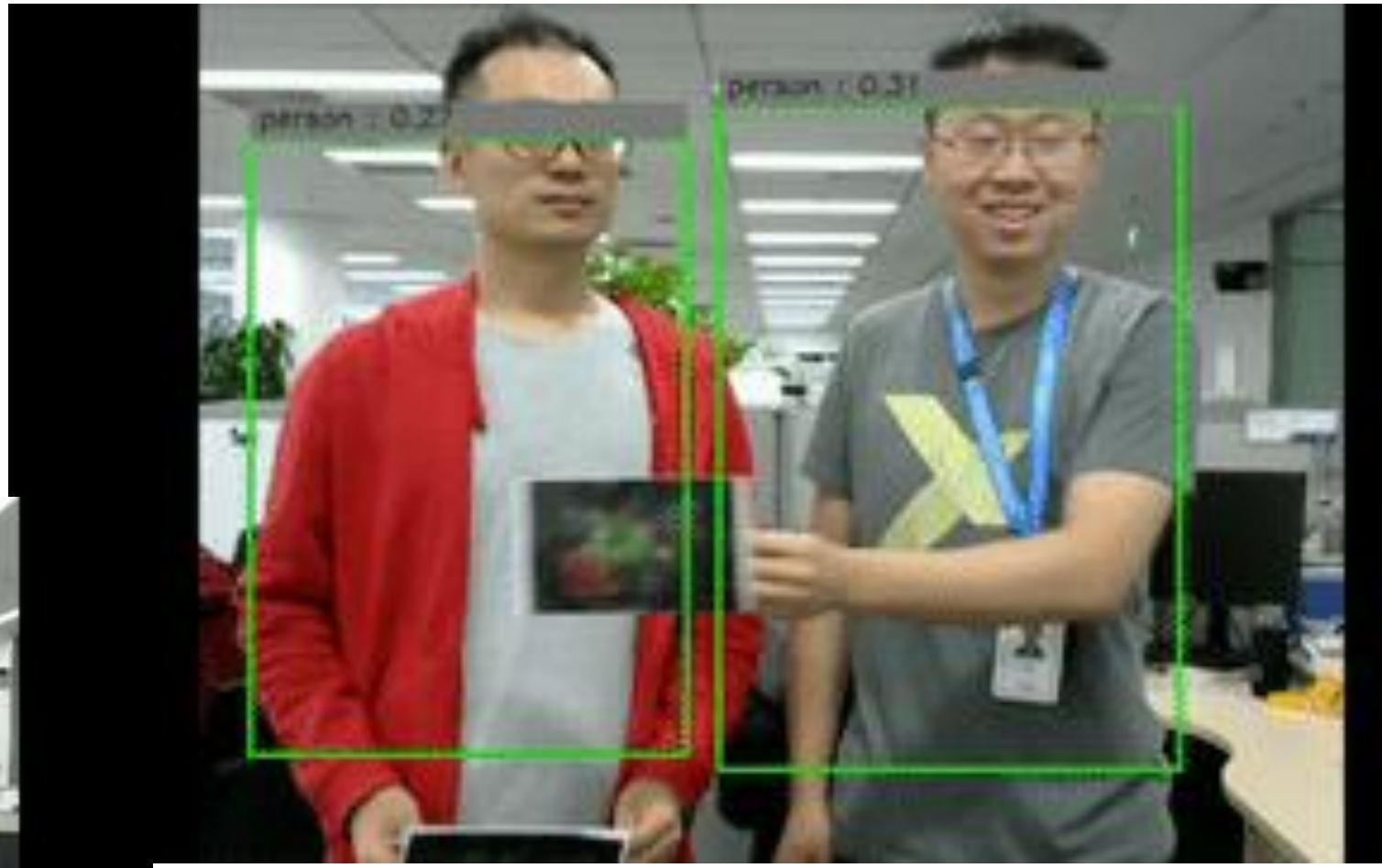
[Szegedy Zaremba Sutskever Bruna Erhan Goodfellow Fergus 2013]
[Biggio Corona Maiorca Nelson Srndic Laskov Giacinto Roli 2013]

But also: [Dalvi Domingos Mausam Sanghai Verma 2004][Lowd Meek 2005]
[Globerson Roweis 2006][Kolcz Teo 2009][Barreno Nelson Rubinstein Joseph Tygar 2010]
[Biggio Fumera Roli 2010][Biggio Fumera Roli 2014][Srndic Laskov 2013]

Robust Adversarial Inputs



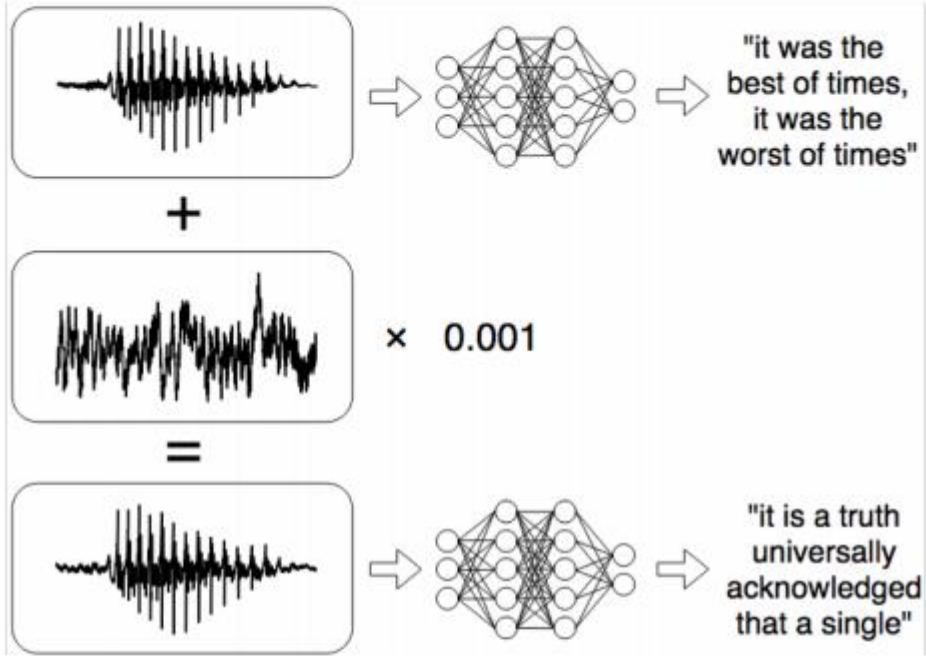

rifle
shield, buck
revolver, si
Fooling Neural Networks
in the Real World
labsix


person : 0.31

Why Is This Brittleness of ML a Problem?

→ Security

[Carlini Wagner 2018]:
Voice commands that are unintelligible to humans

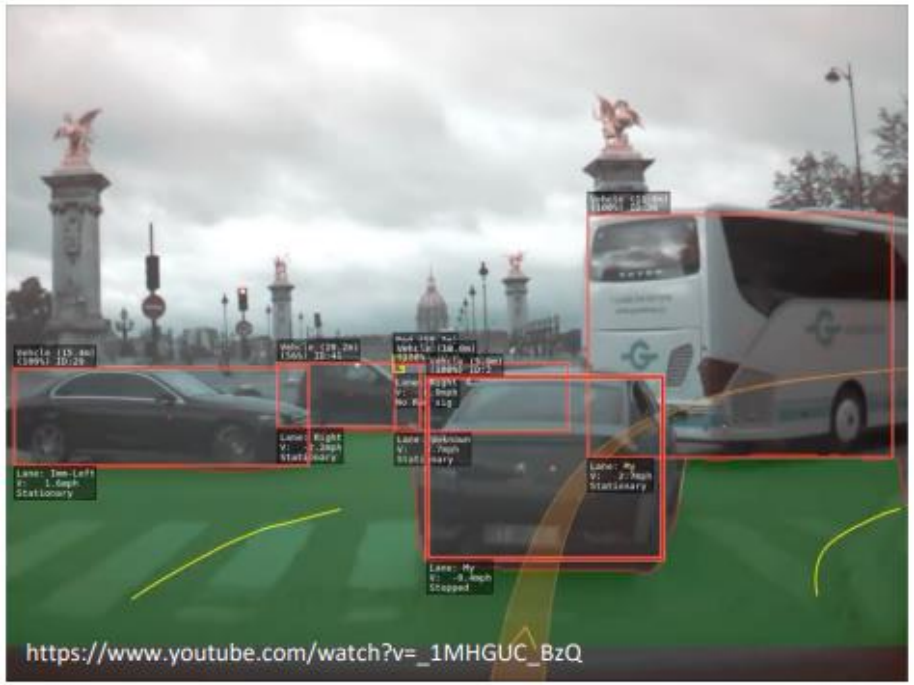


[Sharif Bhagavatula Bauer Reiter 2016]:
Glasses that fool face recognition

Why Is This Brittleness of ML a Problem?

→ Security

→ Safety



[Eykholt Evtimov Fernandes Li Rahmati Xiao Prakash Kohno Song 2017]

Learning Incorrect Things

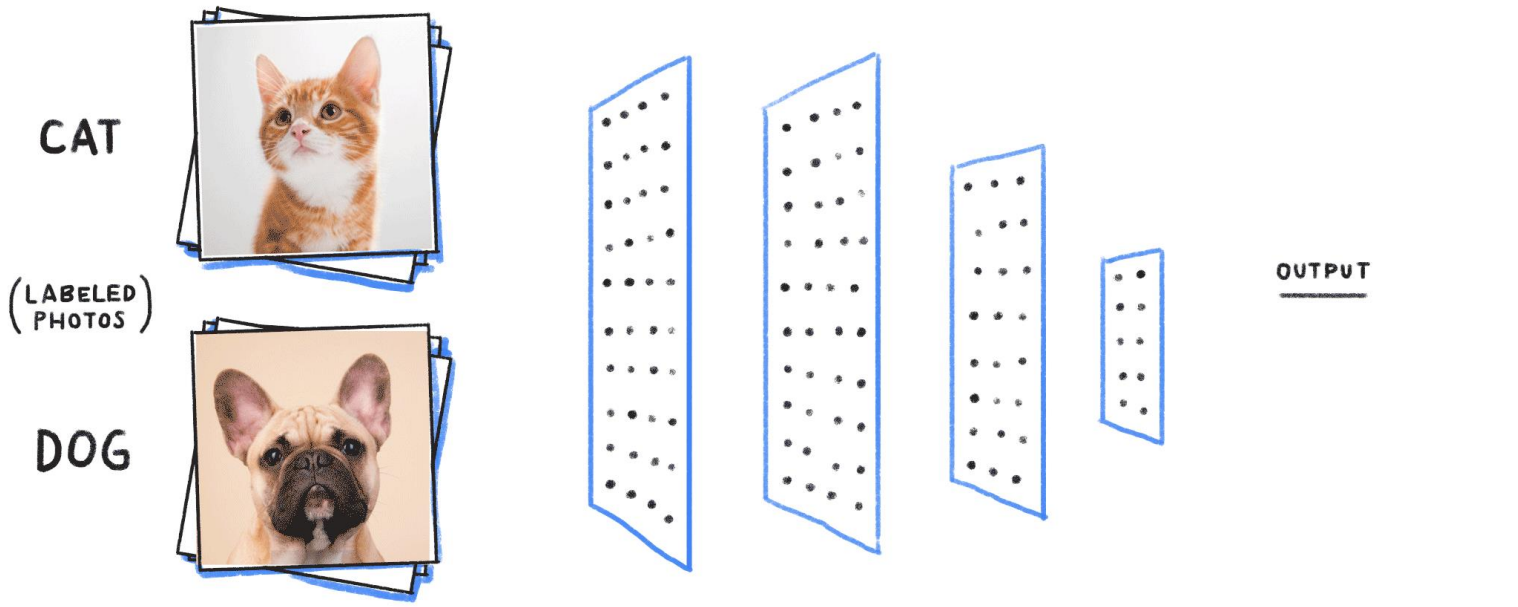
Background

What is supervised machine learning?

X: input data

f: function we want to learn

y: output



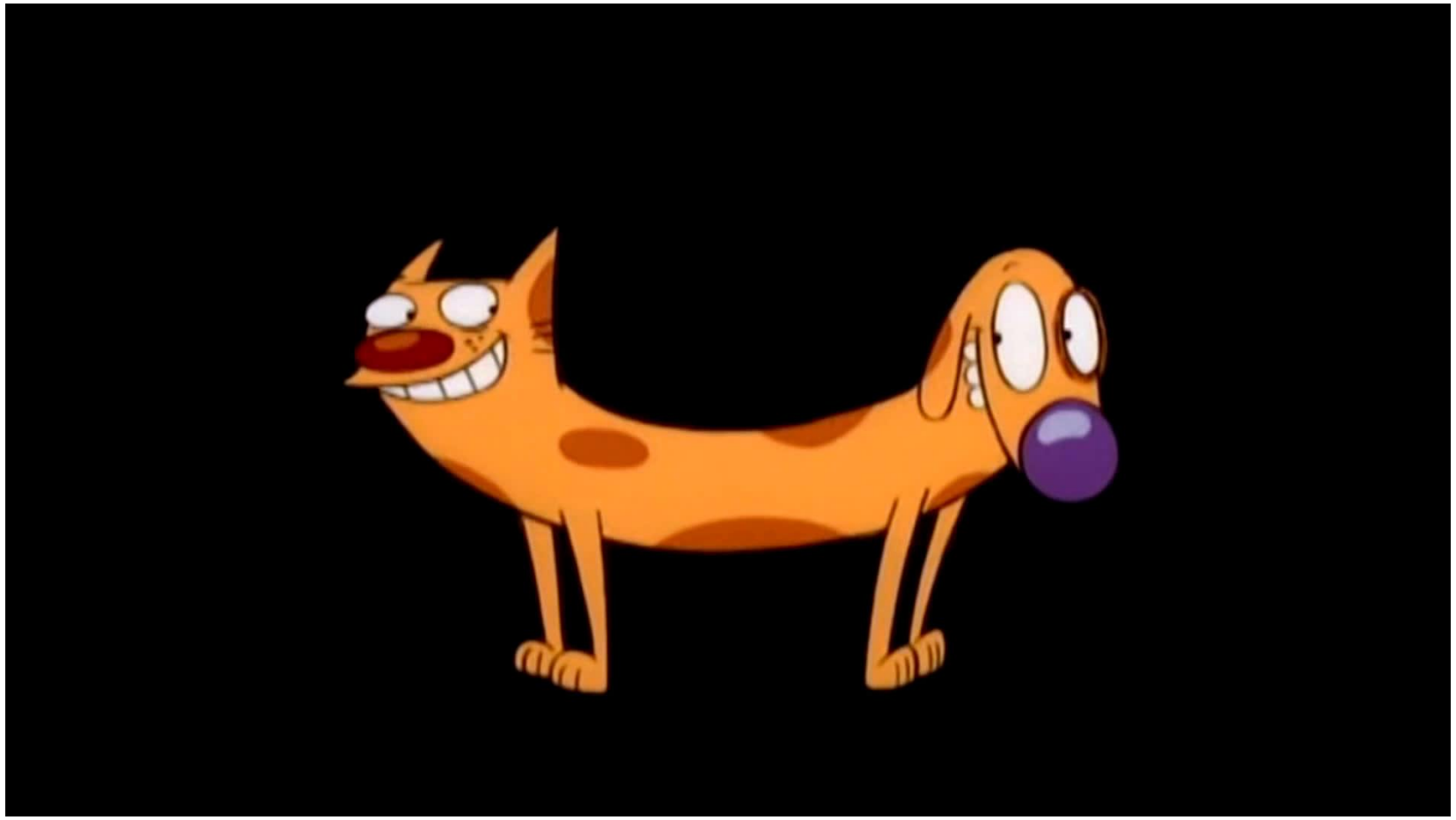
From: https://github.com/ReiCHU31/Cat-Dog-Classification-Flask-App

First, we need a goal

$f($  $) \rightarrow$ dog

$f($  $) \rightarrow$ cat

But what if we fed this to the network?



$f($  $)$ → **Probability (DOG)=1.00**

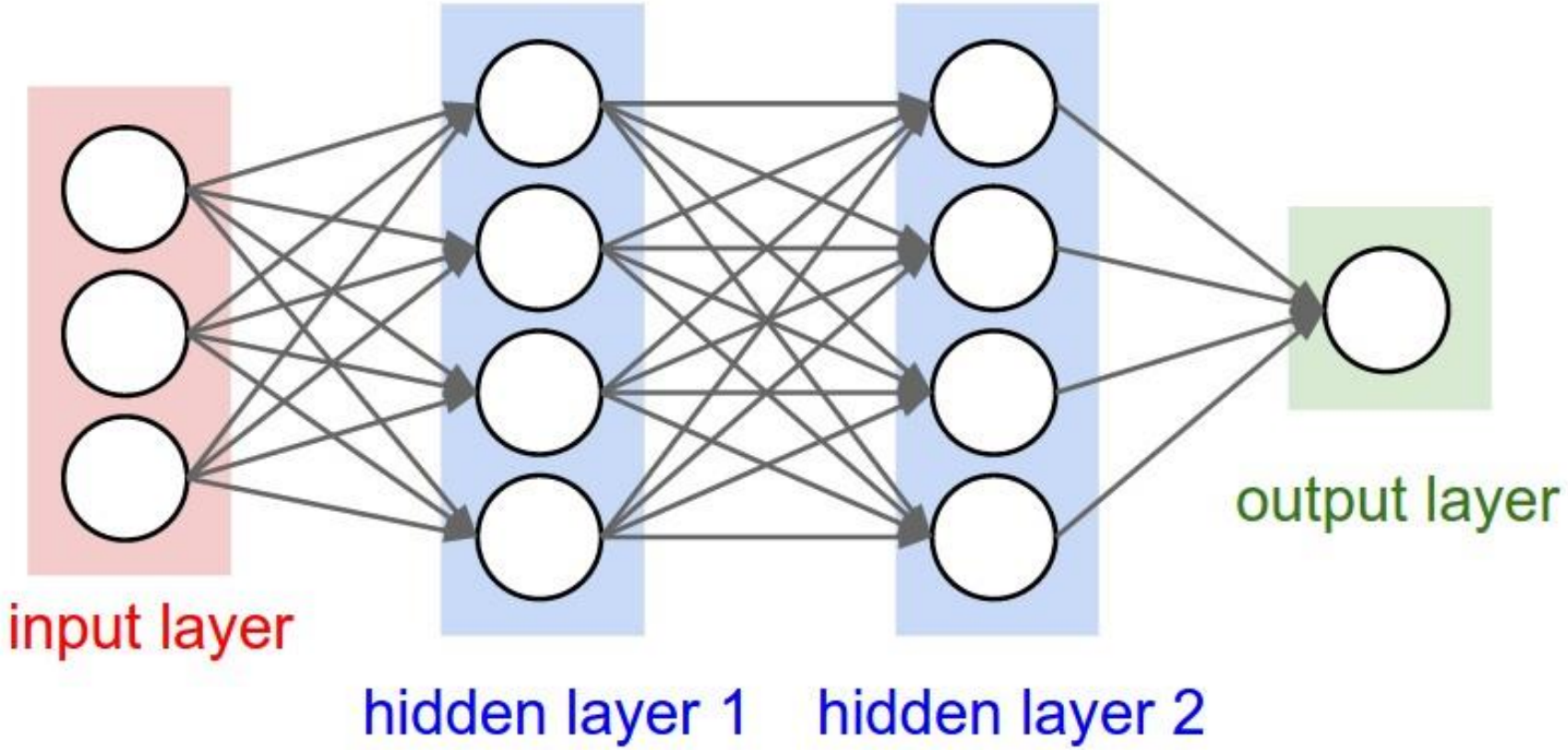
$f($  $)$ → **Probability (CAT)=1.00**

What's next – LABELLED DATA!

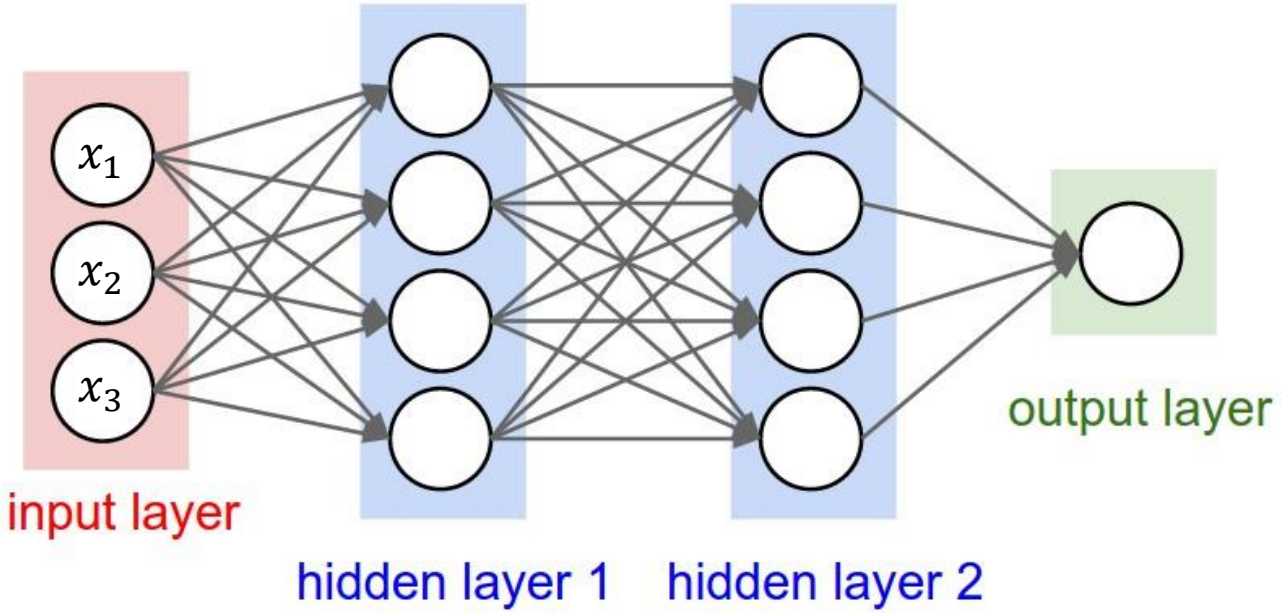
Cat

Dog

We need something that can learn…



What my parents think I do

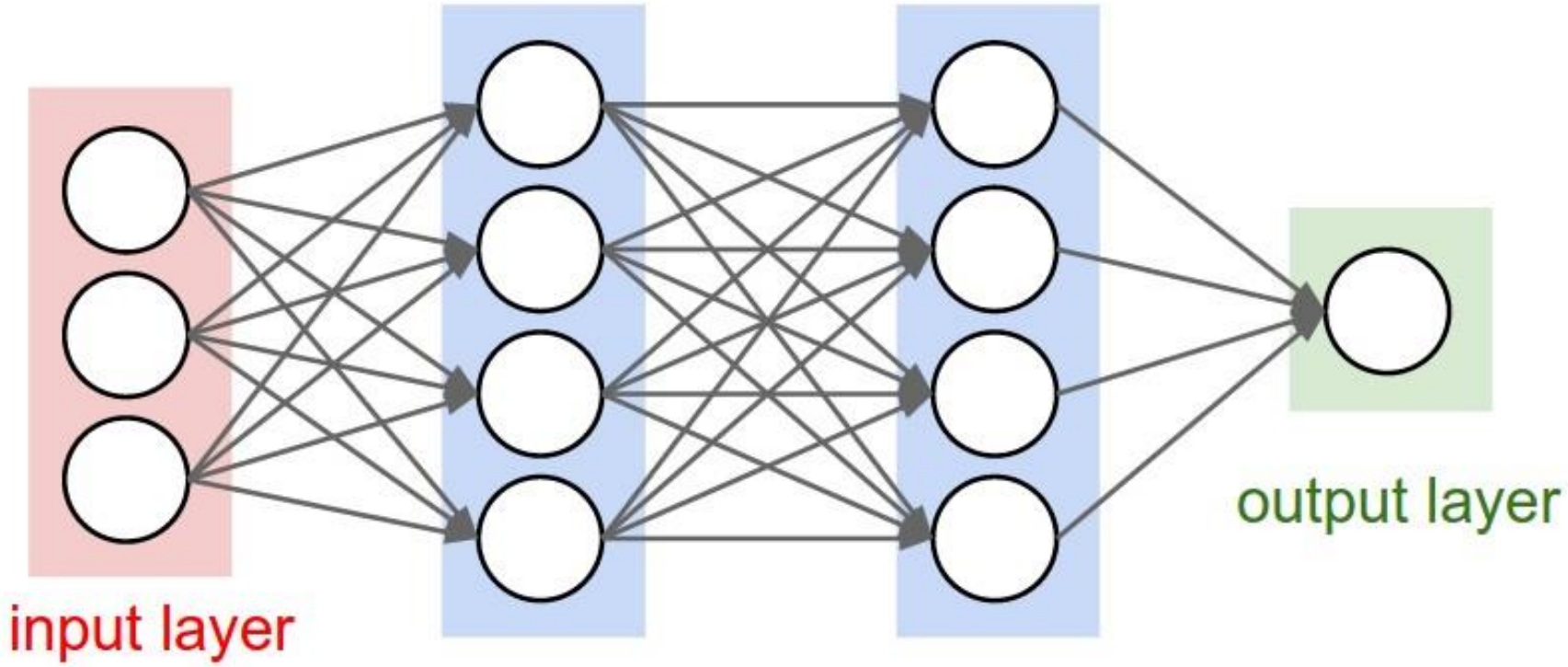


What I actually do

$w_1 x_1 + w_2 x_2 + \cdots$

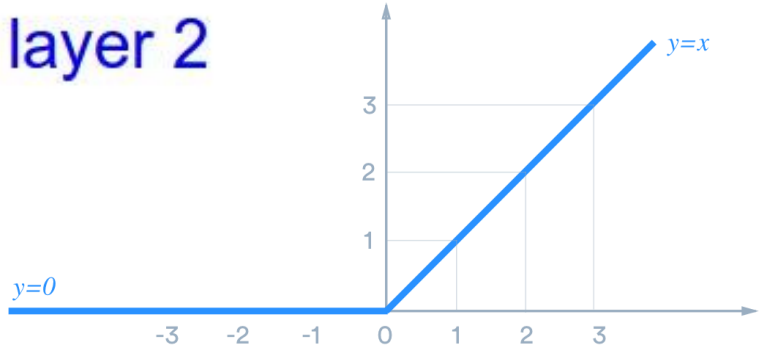
e.g. $\mathbf{4.3}x_1 - \mathbf{1}x_2 \ldots$

Machine learning is all about finding the right $\boldsymbol{w}$



$$w_1 x_1 + w_2 x_2 + \cdots$$

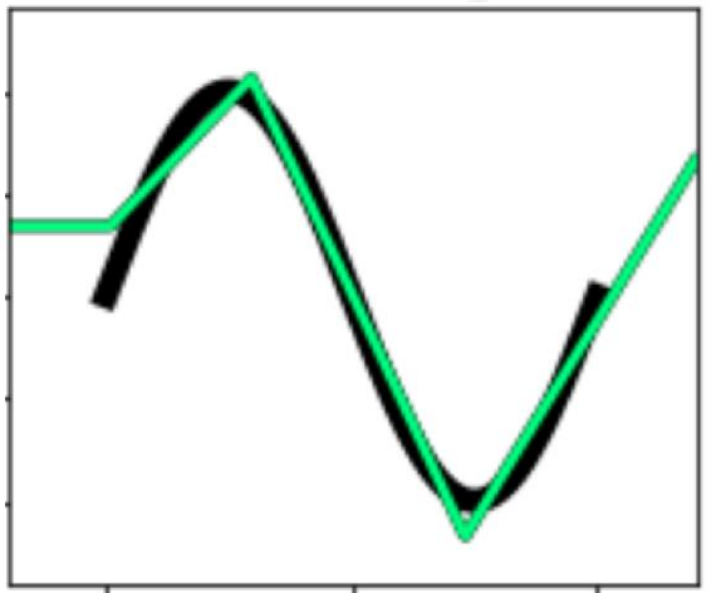
plus!



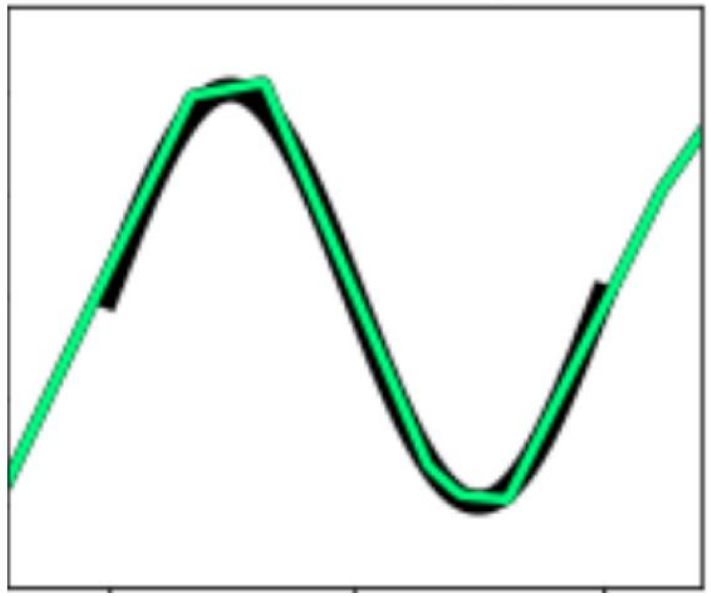
ReLU "activation"
(non-linearity)

Universal Function Approximator! (Hornik, 1991)

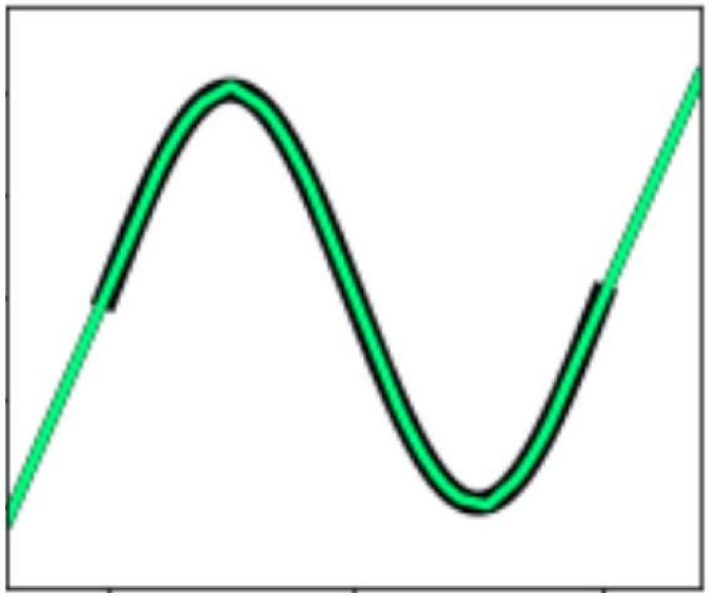
5 ReLUs

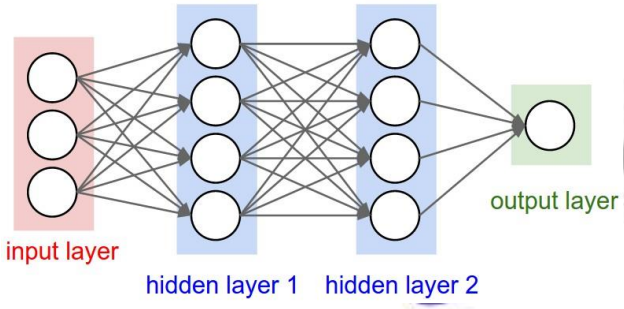


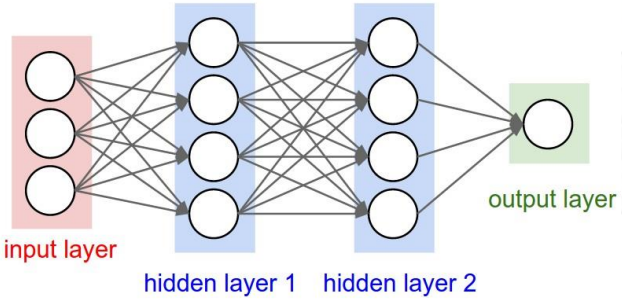
10 ReLUs



50 ReLUs



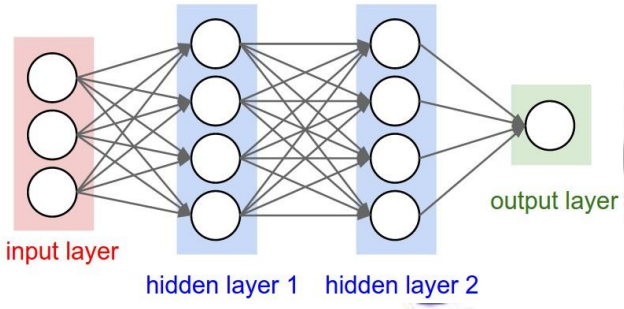


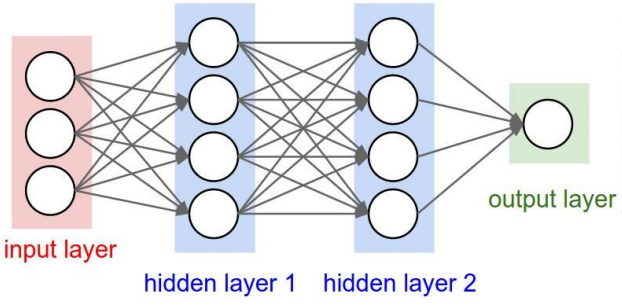


Probability (DOG)=1.00

Probability (CAT)=1.00

Expectations vs Reality

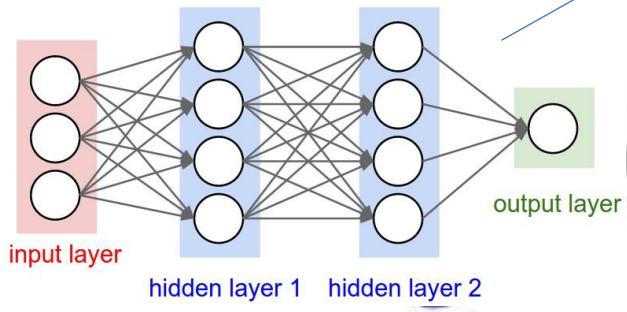




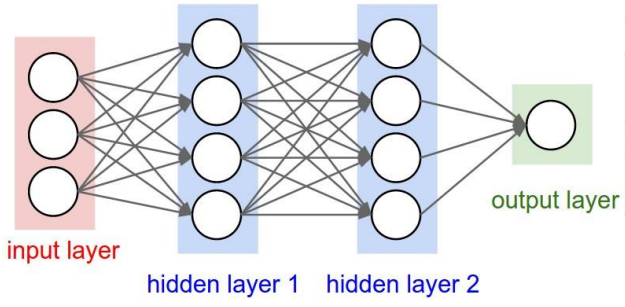
Probability (DOG)=0.27

Probability (CAT)=0.01

How do we make them learn?



Probability (DOG)=0.27



Probability (CAT)=0.01

Penalizing Errors - Loss Function

Probability (DOG)=0.27

Probability (DOG)=1.00

Probability (CAT)=0.01

Probability (CAT)=1.00

The difference can be thought of as the "**loss**"!

*(more specifically, the difference between two probability distributions can be measured with KL divergence, which embeds the **CrossEntropy** of the two probability distributions. In practice, **CrossEntropyLoss** is used.)*

Penalizing Errors - Loss Function

Probability (DOG)=0.27

Probability (DOG)=1.00

Probability (CAT)=0.01

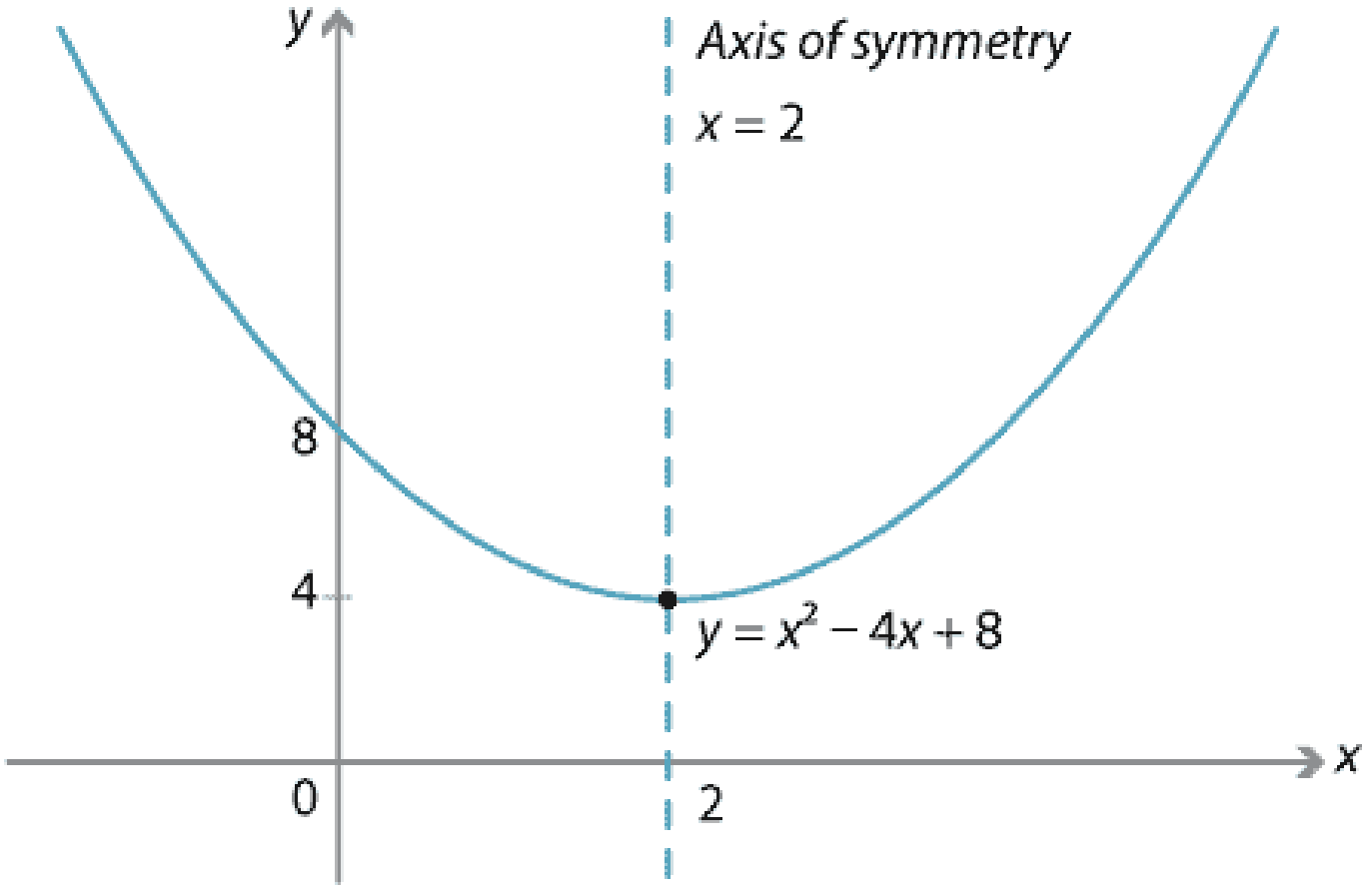
Probability (CAT)=1.00

The difference can be thought of as the "**loss**"!

*(more specifically, the difference between two probability distributions can be measured with KL divergence, which embeds the **CrossEntropy** of the two probability distributions. In practice, CrossEntropyLoss is used.)*

Learning – Secondary School Edition

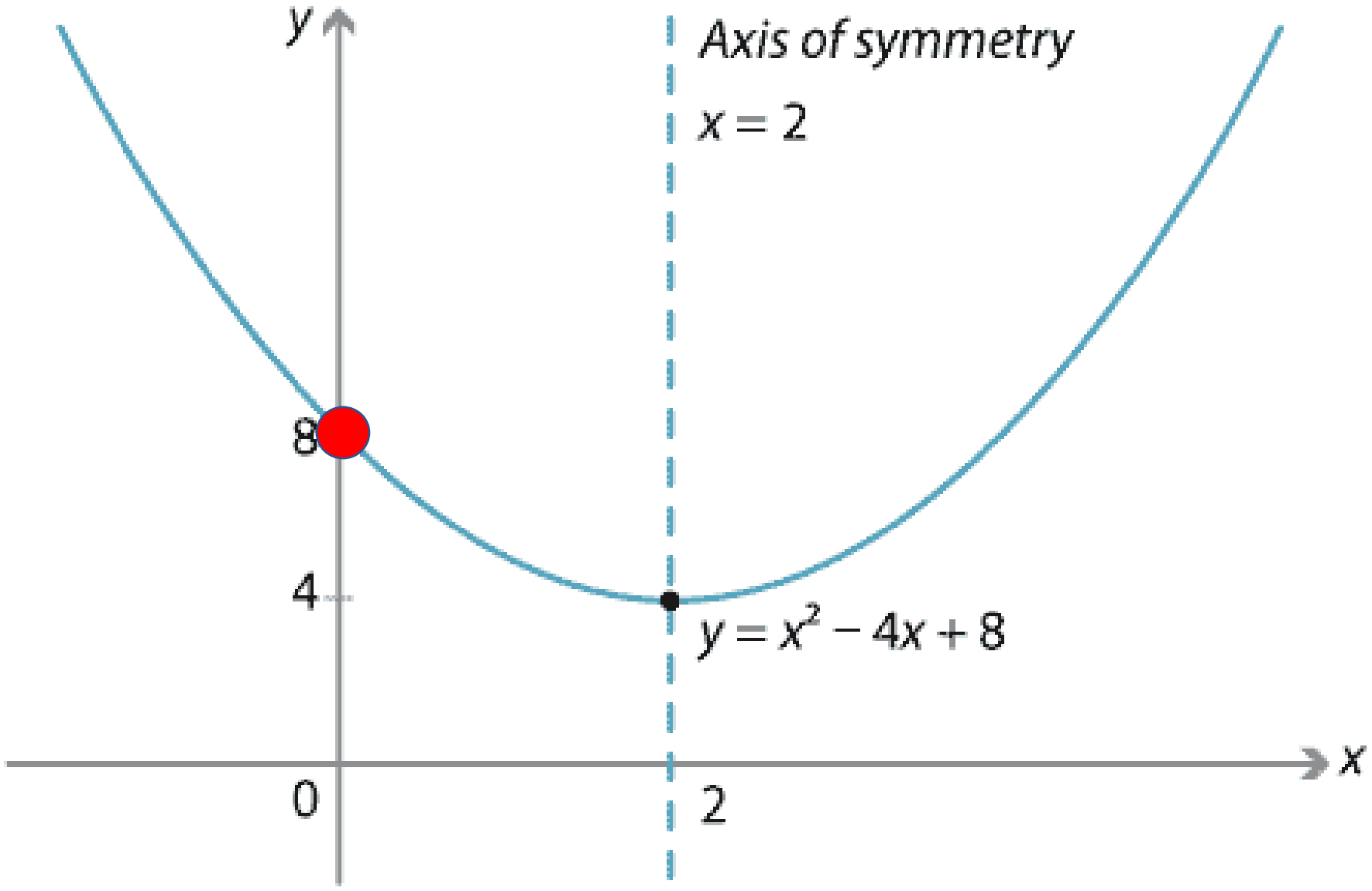
Let's Minimize This!



Say x = 0

(you can imagine x to be a proxy for **w**, and y=f(x) to be the loss function)

We want to minimize the loss!



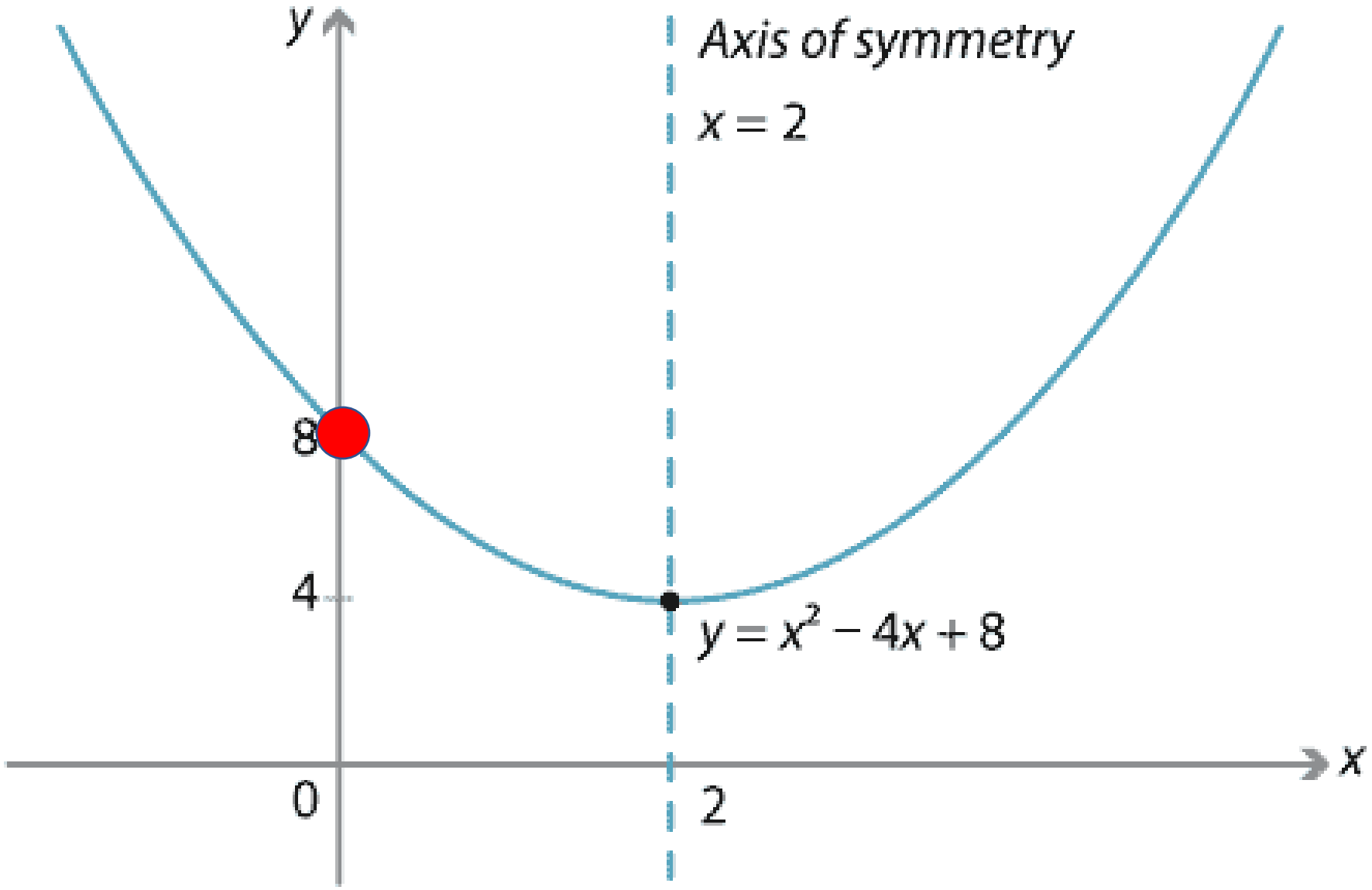
Say x = 0

$$\frac{dy}{dx} = 2x - 4$$

When $x = 0$

$$\frac{dy}{dx} = -4$$

This indicates that increasing the value of x by 1 changes the value of f(x) by -4



Say x = 0

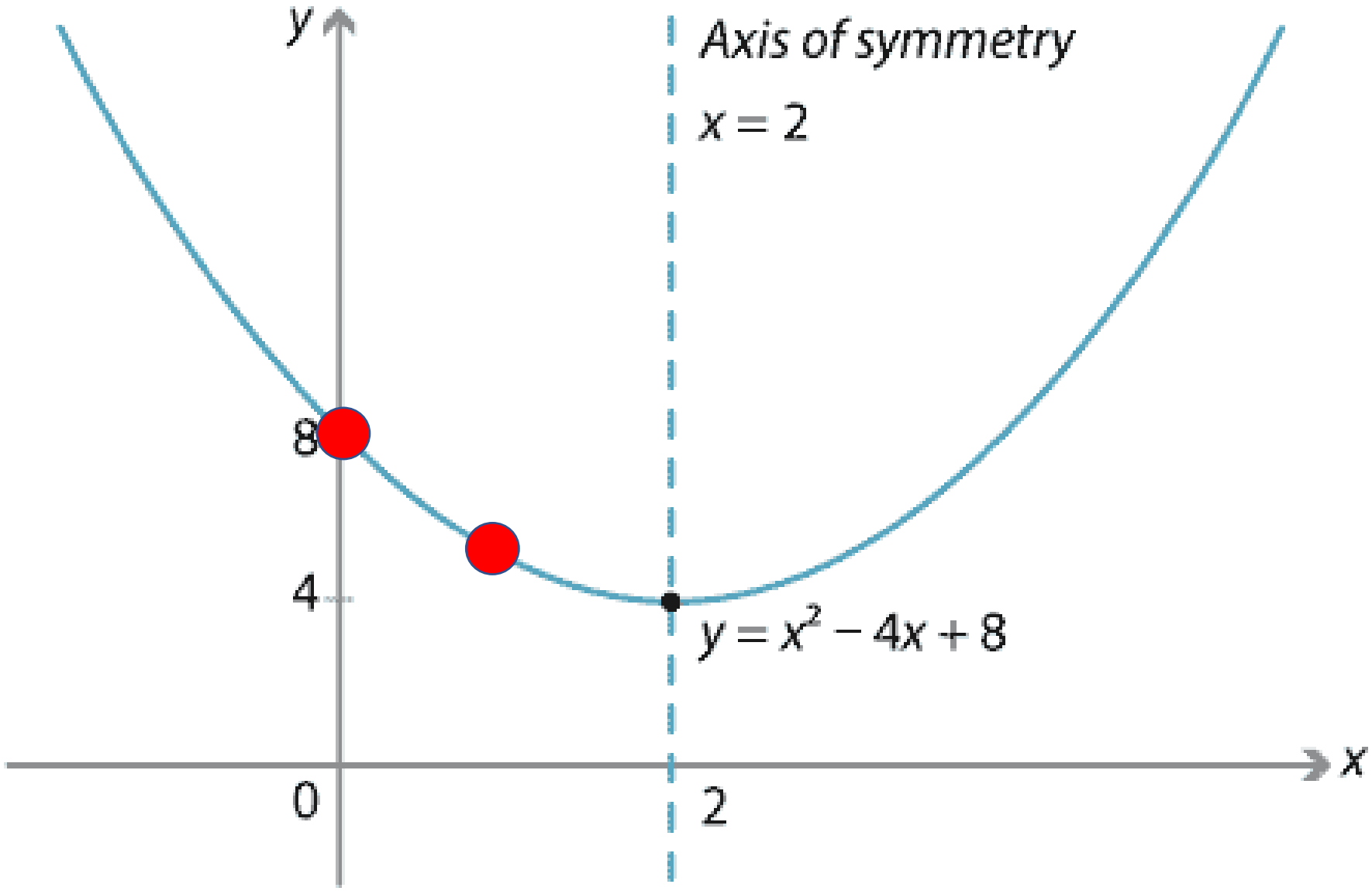
$$\frac{dy}{dx} = 2x - 4$$

When x = 0

$$\frac{dy}{dx} = -4$$

This indicates that increasing the value of x by 1 changes the value of f(x) by -4

(if you draw a straightline/tangent at the point)



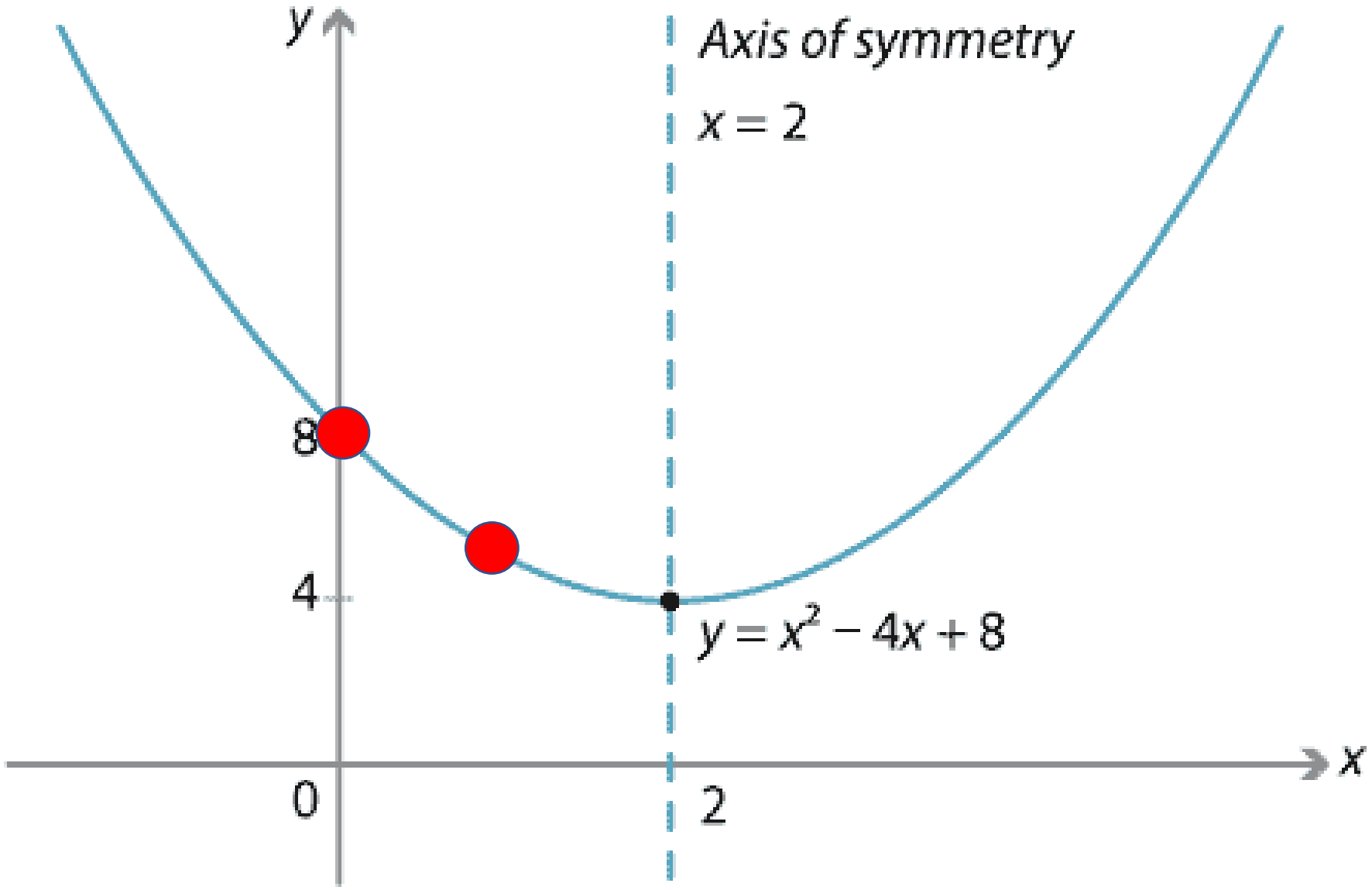
Say x = 0

$$\frac{dy}{dx} = 2x - 4$$

When $x = 0$

$$\frac{dy}{dx} = -4$$

Increasing value of x ➔ decreasing value of f(x)

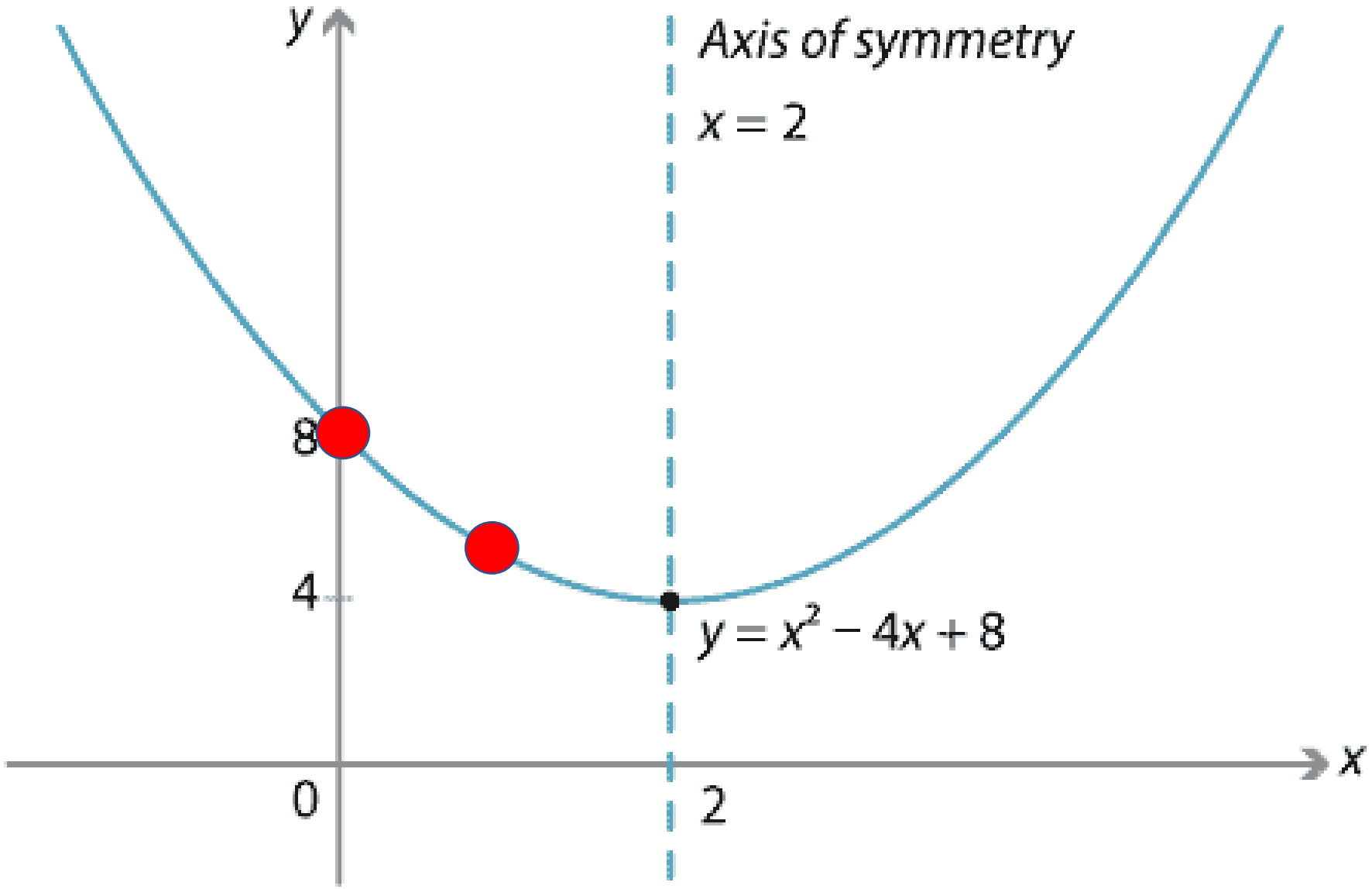


Say x = 0

$$\frac{dy}{dx} = -4$$

Increasing value of x ➔ decreasing value of f(x)

This gives us a direction to change x



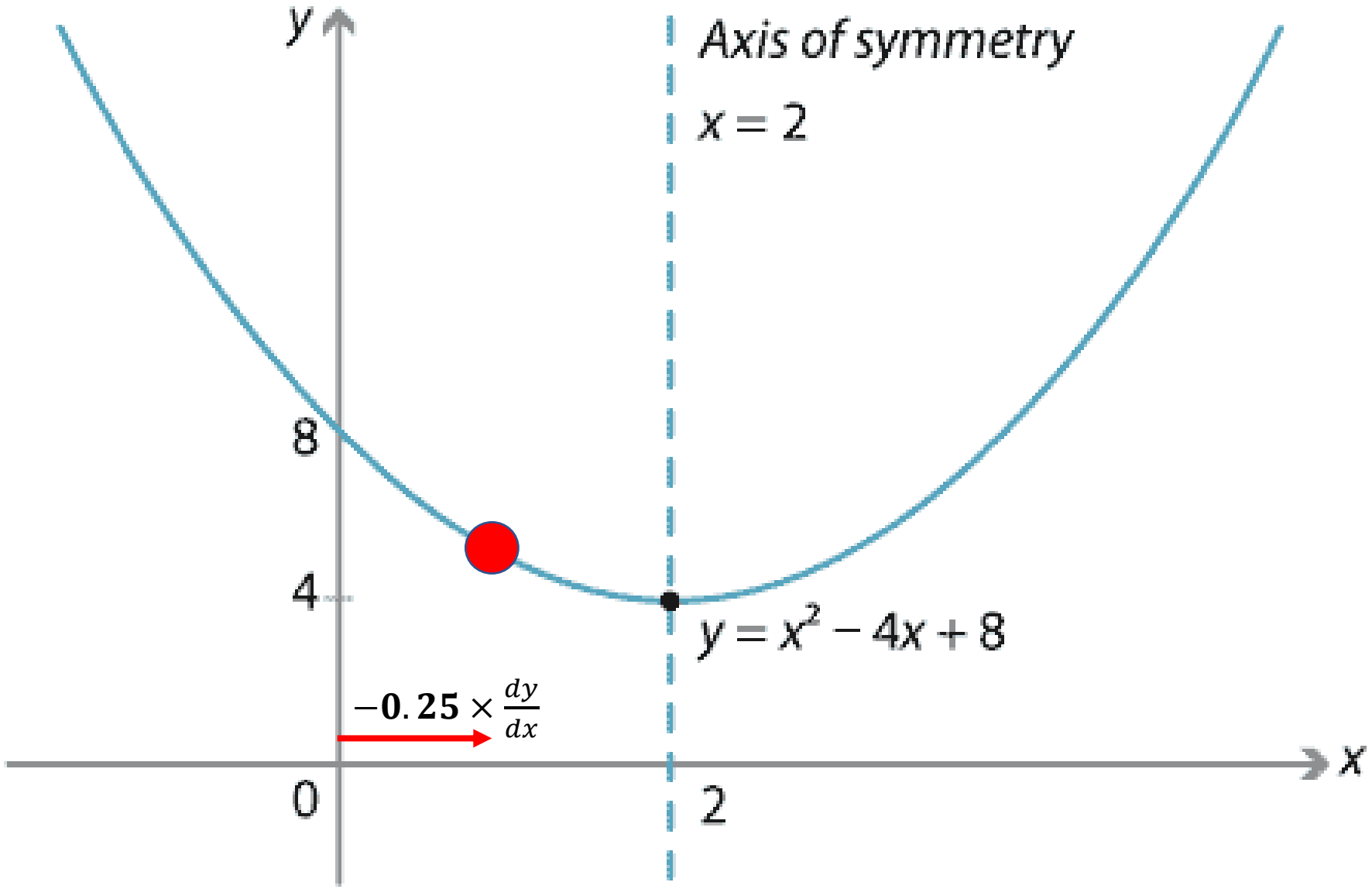
Say x = 0

Increasing value of x ➔ decreasing value of f(x)

This gives us a direction to change x

Let's increase x slightly by

$$x - 0.25 \times \frac{dy}{dx}$$

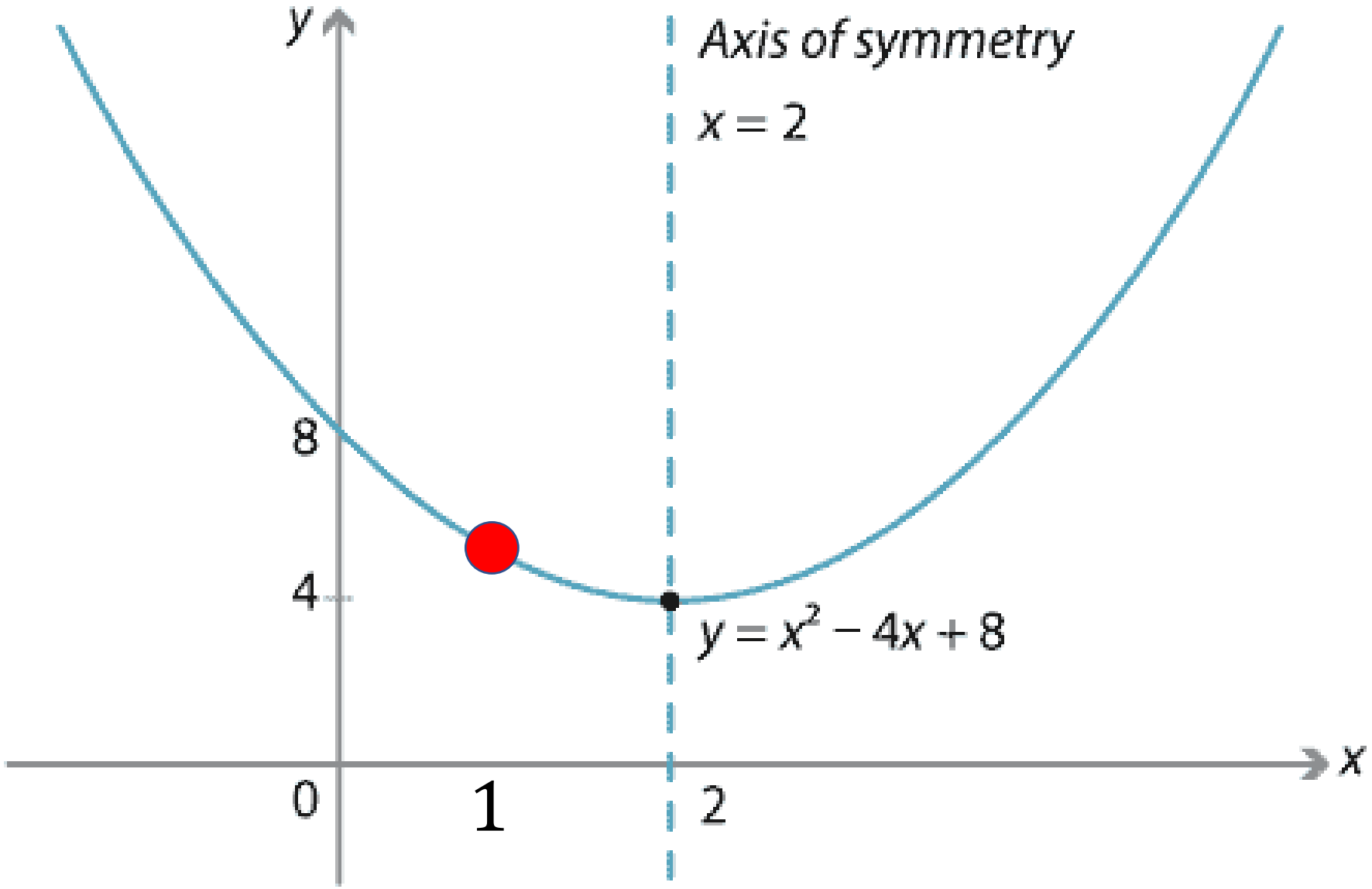


Learning rate $\alpha = 0.25$

Say x = 1

$$\frac{dy}{dx} = -2$$

$$\boldsymbol{x} = \boldsymbol{x} - \mathbf{0.25} \times \frac{dy}{dx}$$
$$= 1 + 0.5$$
$$= 1.5$$

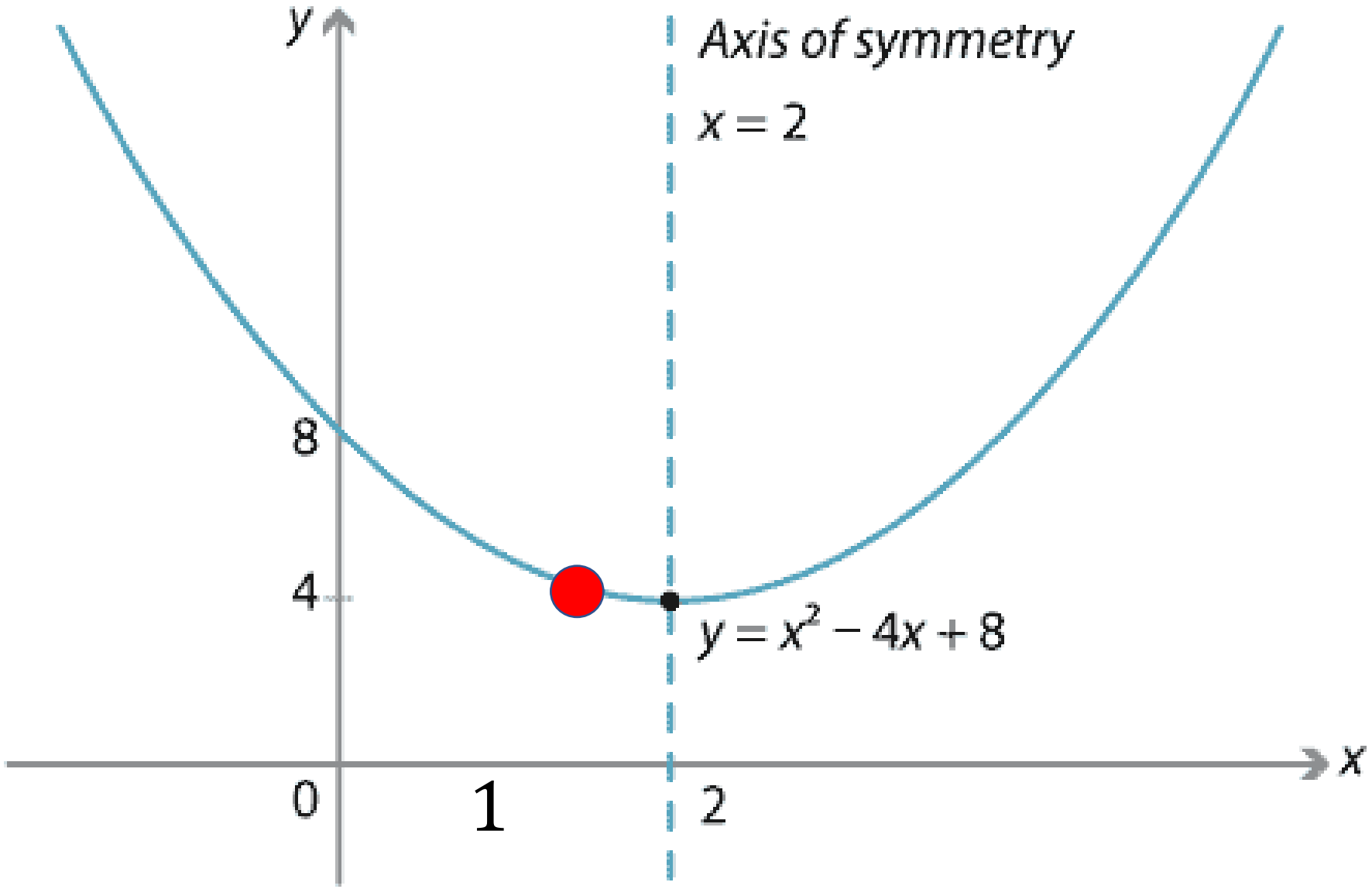


Learning rate α = -0.25

Say x = 1

$$\frac{dy}{dx} = -2$$

$$x = x - 0.25 \times \frac{dy}{dx}$$
$$= 1 + 0.5$$
$$= 1.5$$

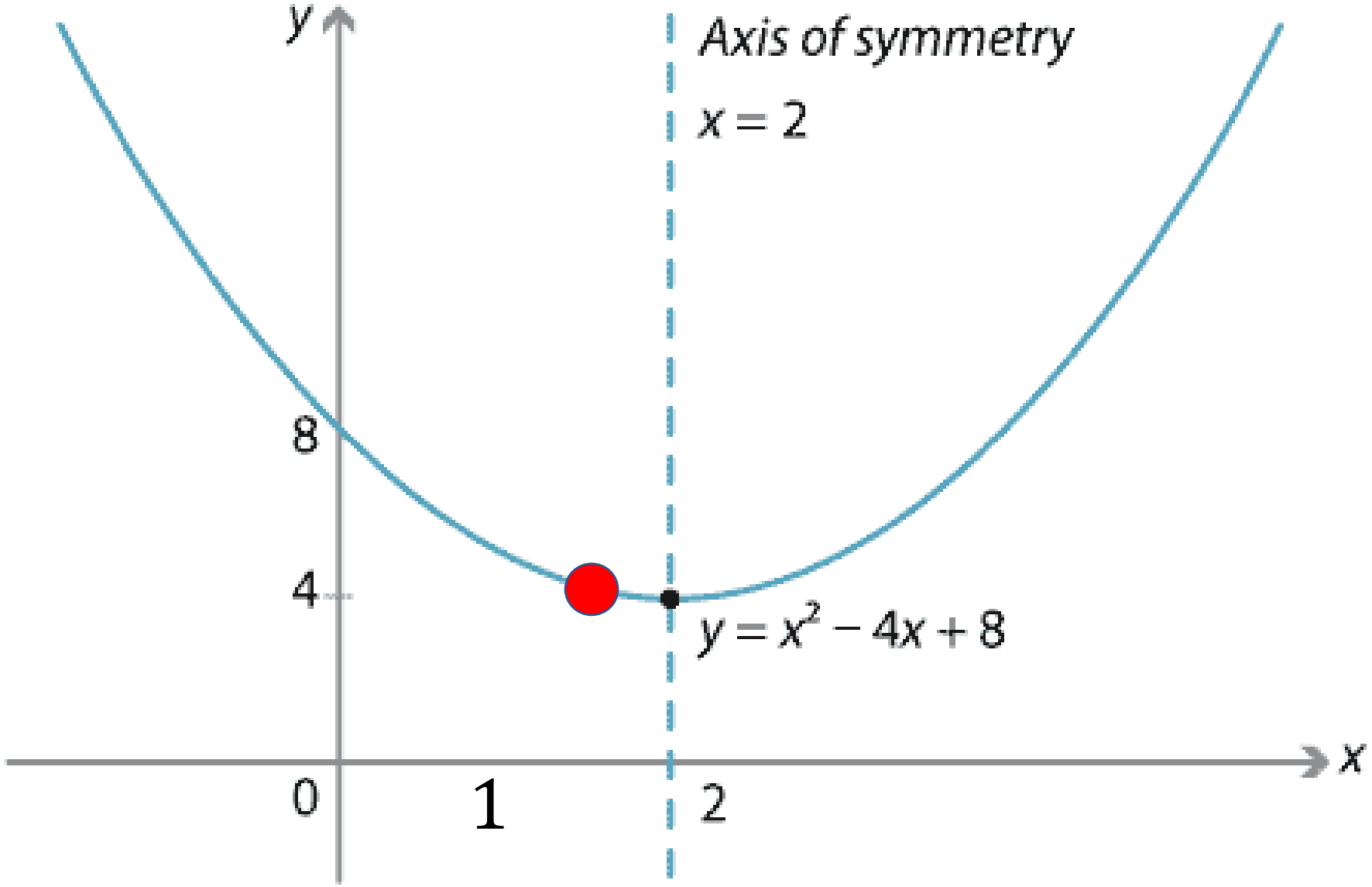


Learning rate $\alpha = 0.25$

Say x = …

$$\frac{dy}{dx} = \cdots$$

$$\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{\alpha} \times \frac{dy}{dx}$$

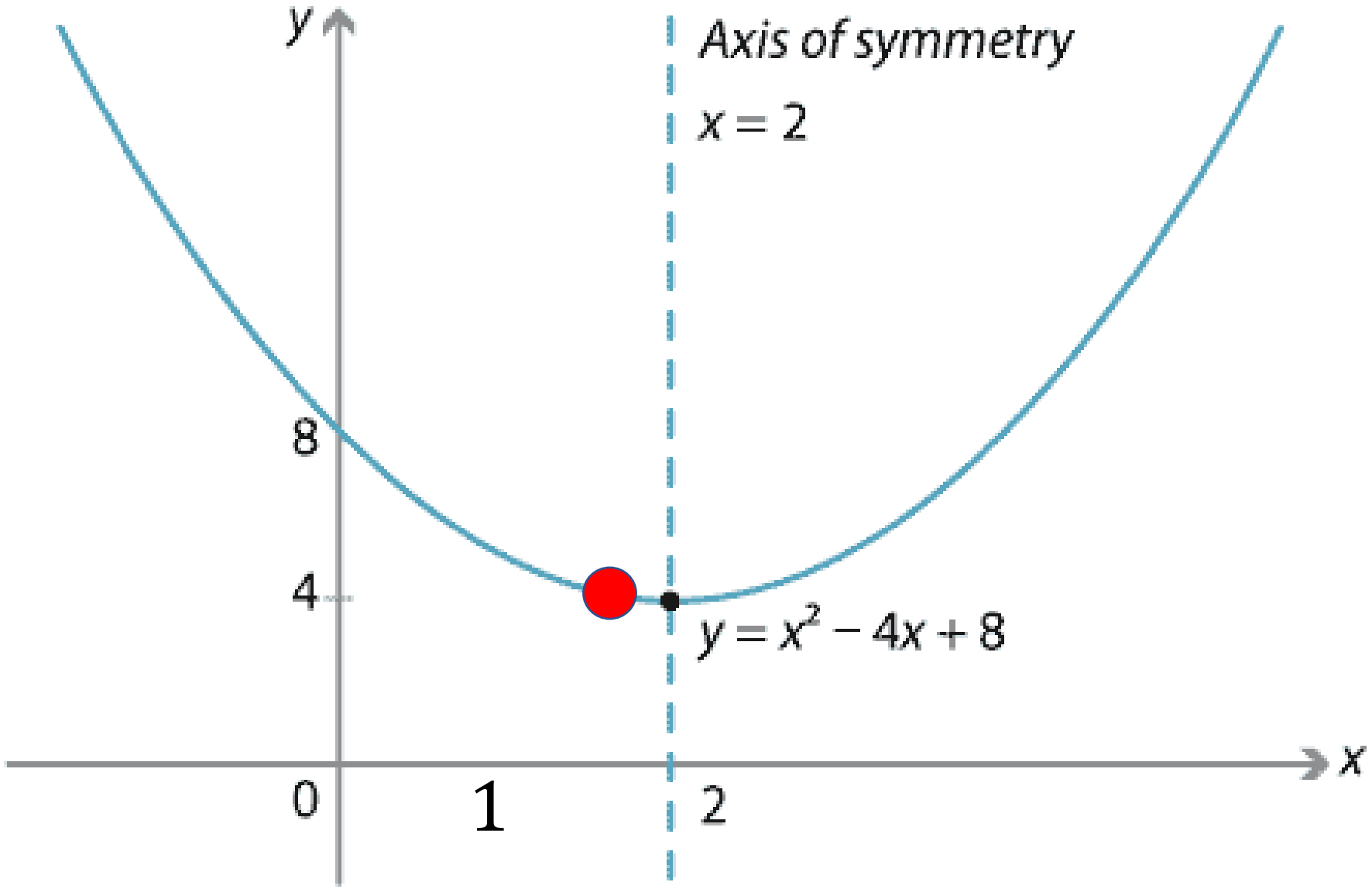


Learning rate $\alpha = 0.25$

Say x = …

$$\frac{dy}{dx} = \cdots$$

$$\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{\alpha} \times \frac{dy}{dx}$$

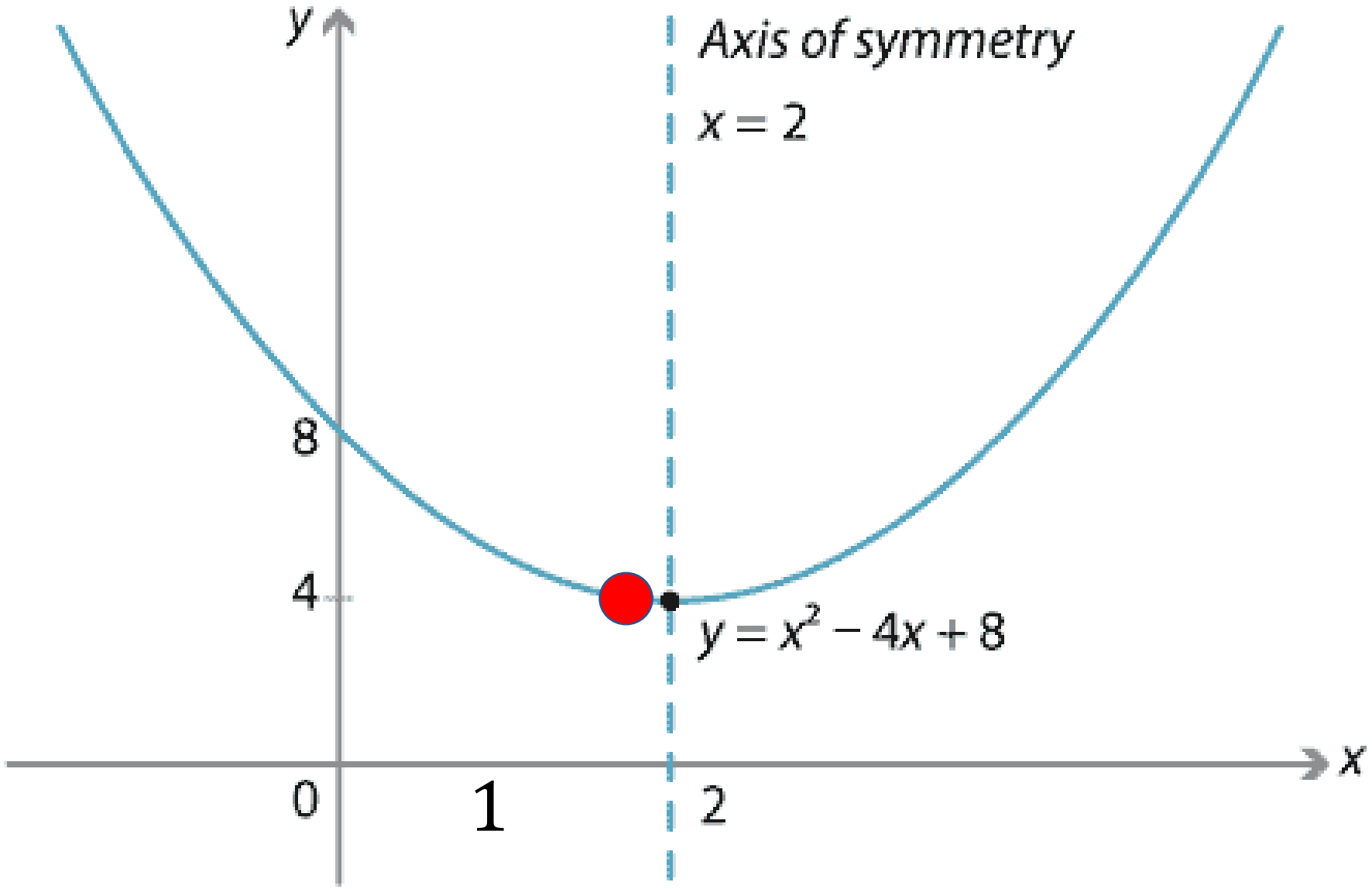


Learning rate $\alpha = 0.25$

Say x = …

$$\frac{dy}{dx} = \cdots$$

$$\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{\alpha} \times \frac{dy}{dx}$$

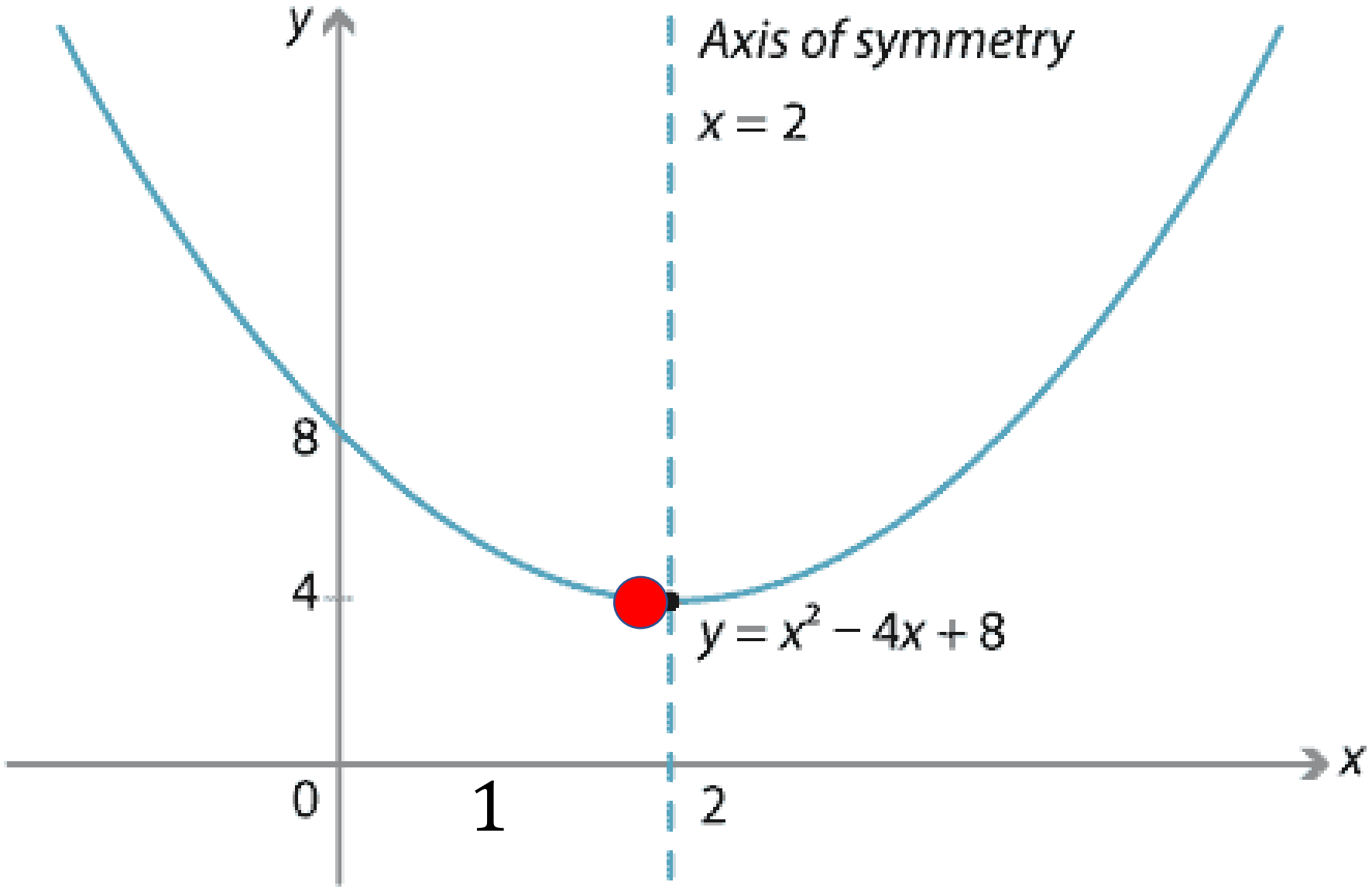


Learning rate $\alpha = 0.25$

Say x = …

$$\frac{dy}{dx} = \cdots$$

$$\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{\alpha} \times \frac{dy}{dx}$$

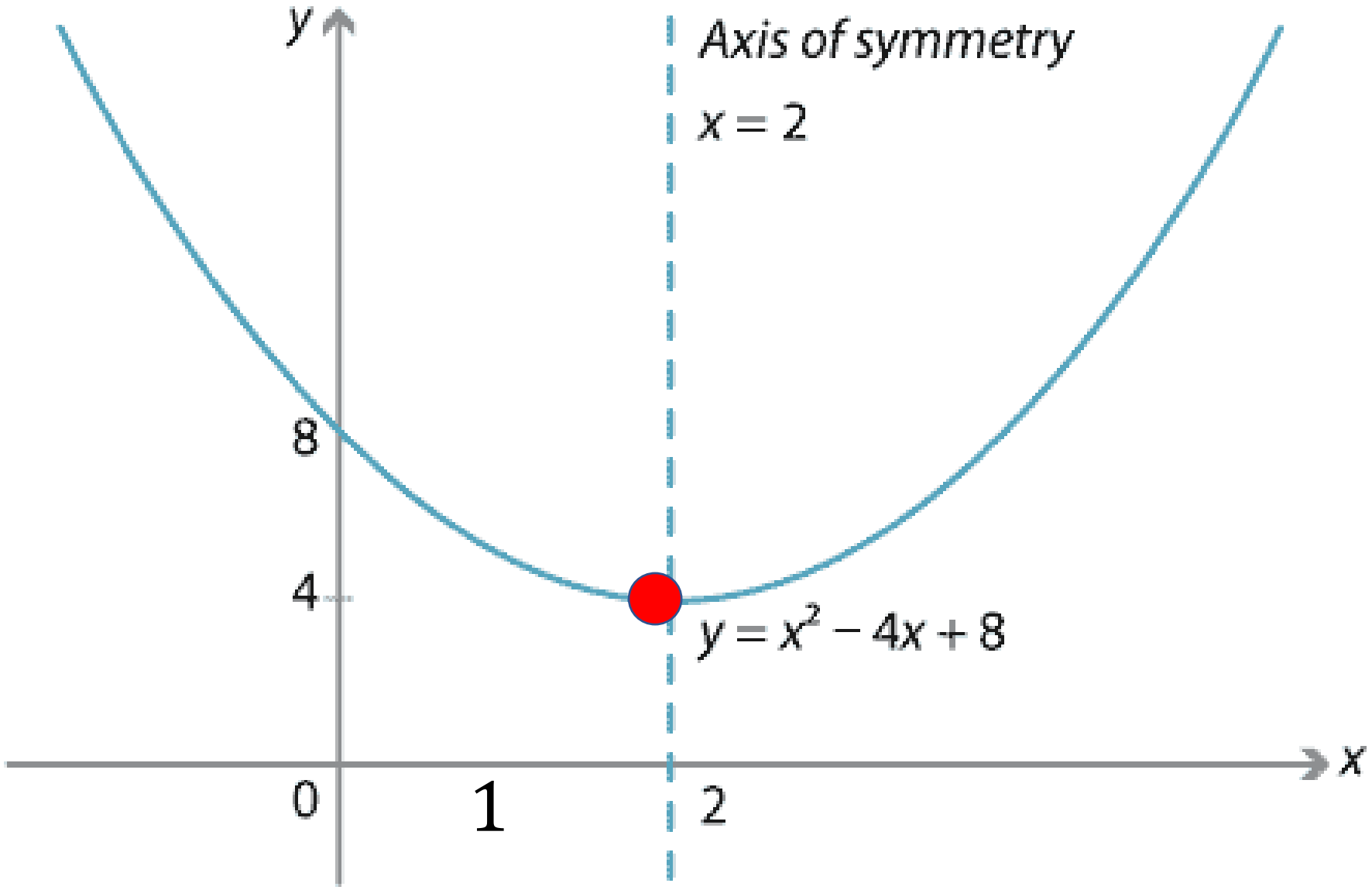


Learning rate $\alpha = 0.25$

Say x = …

$$\frac{dy}{dx} = \cdots$$

$$\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{\alpha} \times \frac{dy}{dx}$$



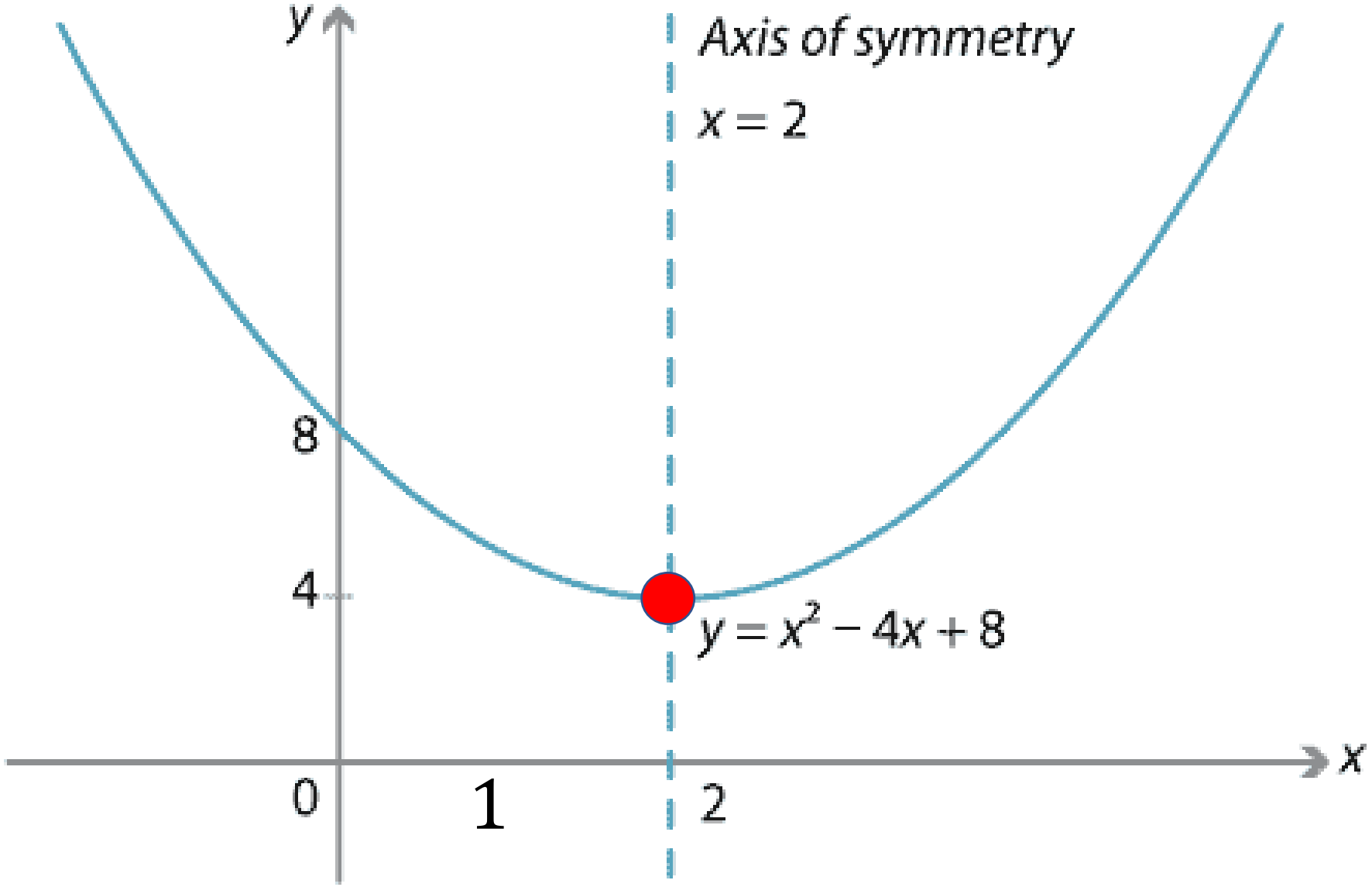
Learning rate $\alpha = 0.25$

Say x = 2

$$\frac{dy}{dx} = 0$$

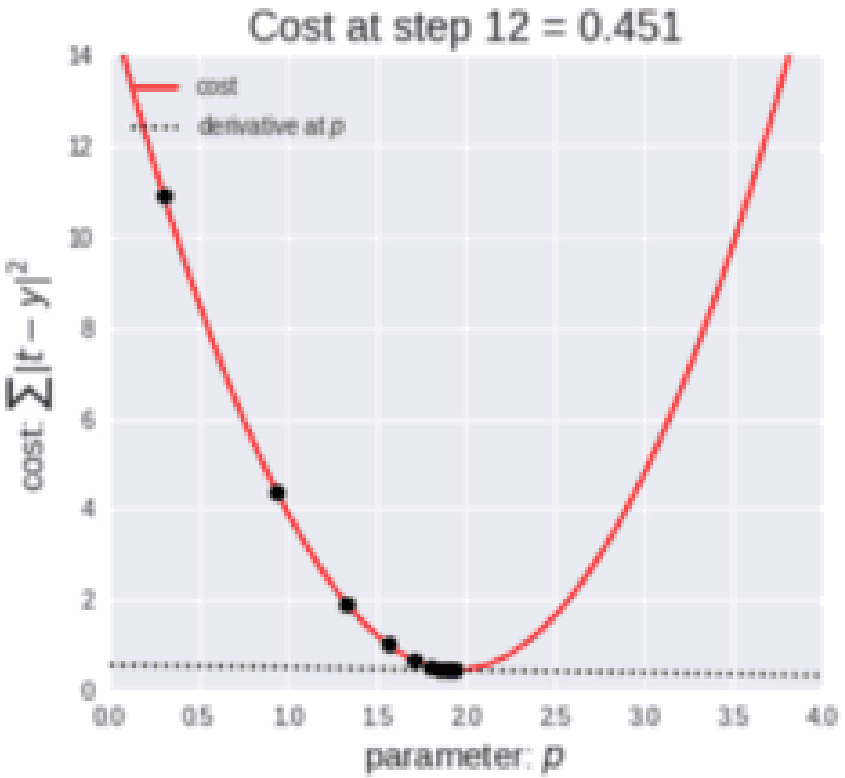
We are done!

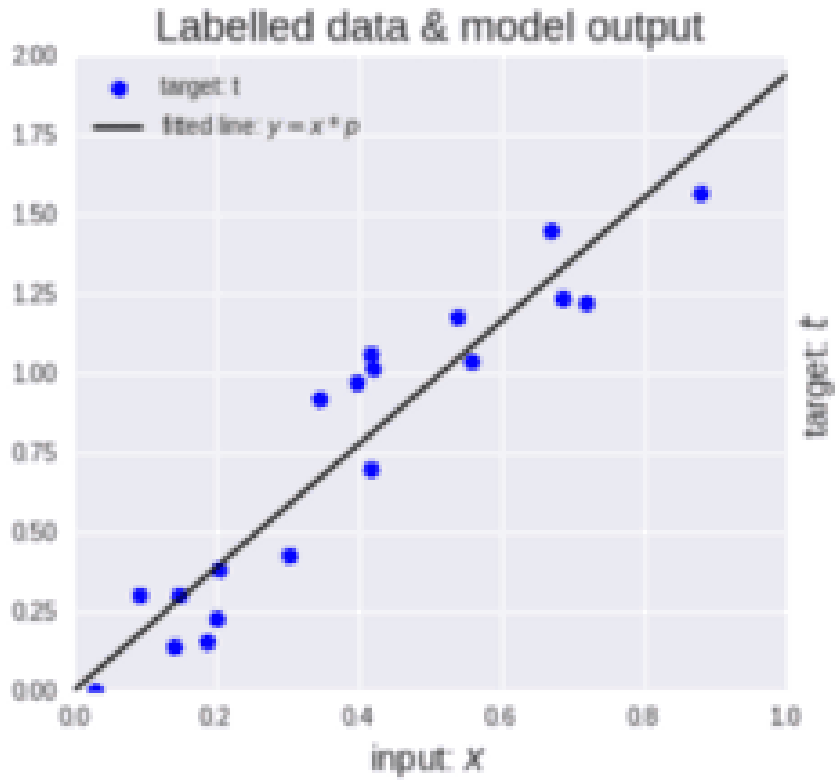
x=2 gives the minimum for y=f(x).



Learning rate $\alpha = 0.25$

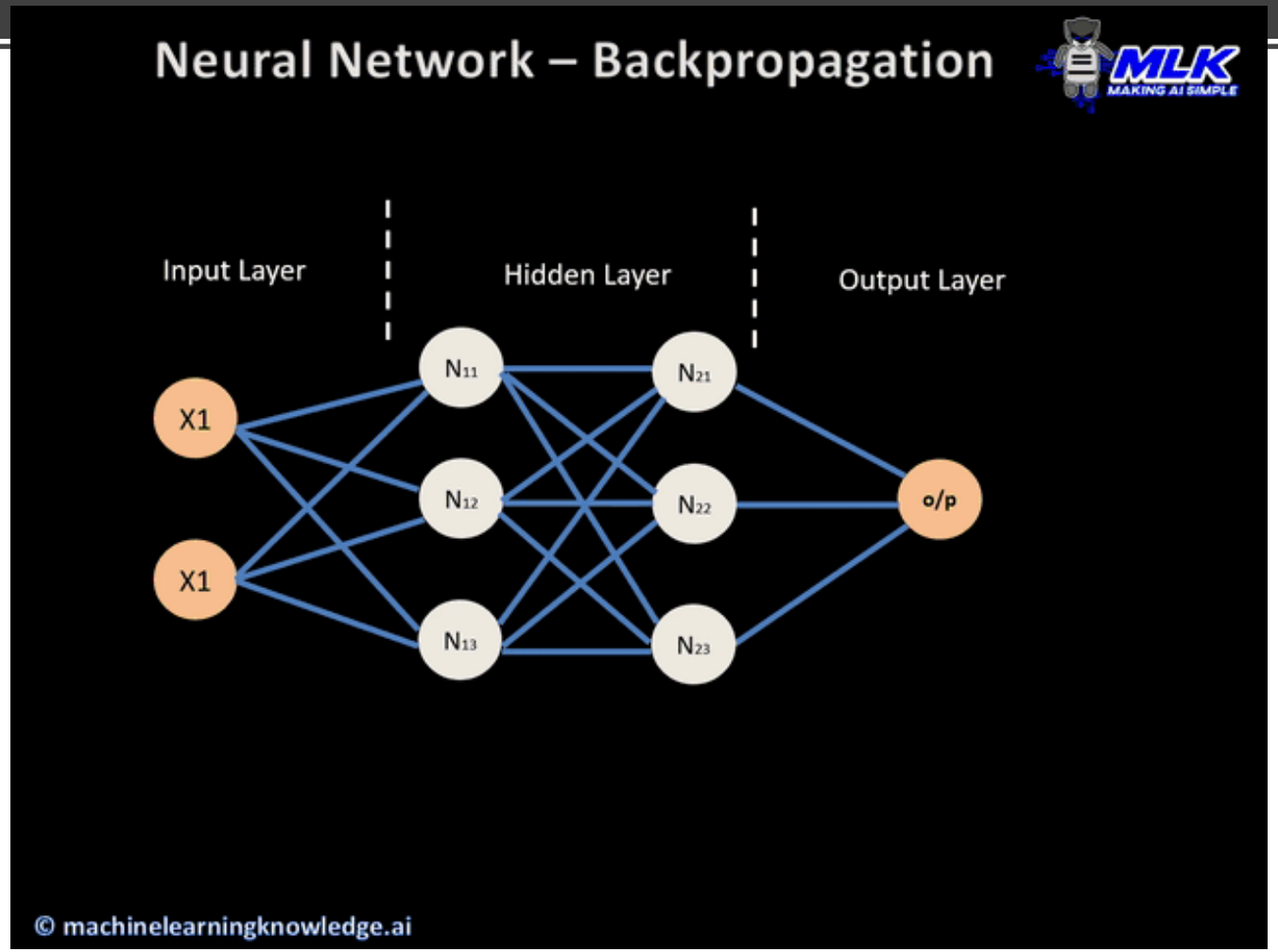
Learning: Gradient Descent!



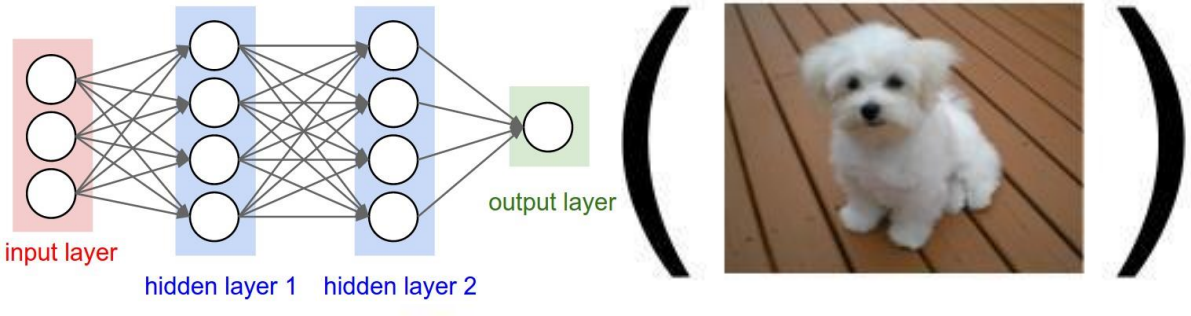


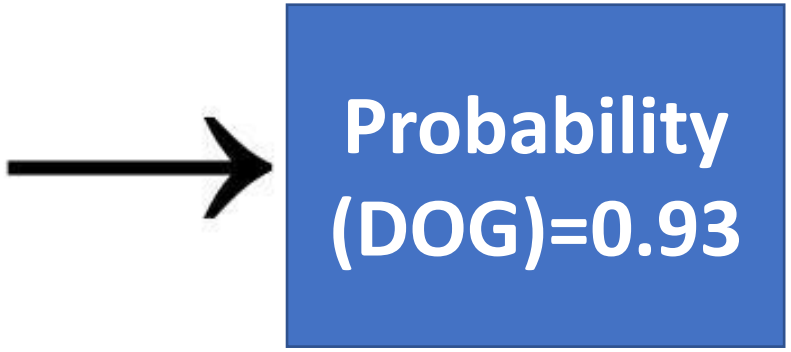
What about “Backpropagation”?

Apply “chain rule” so not just one parameter is updated, but **all weights** are updated

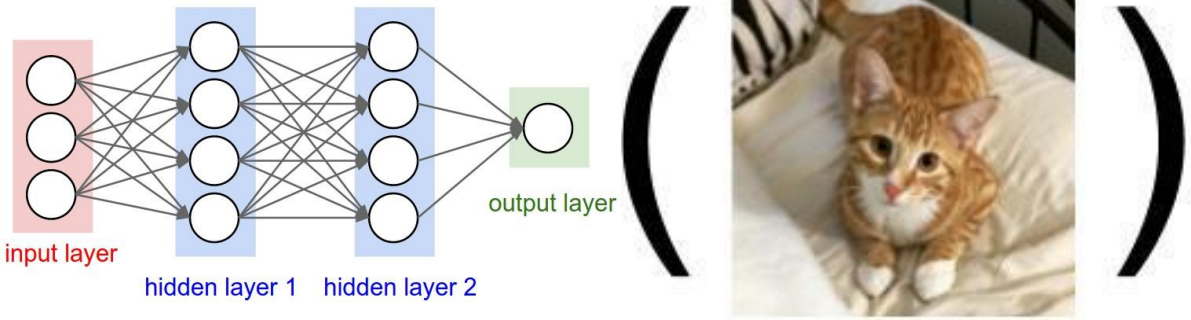


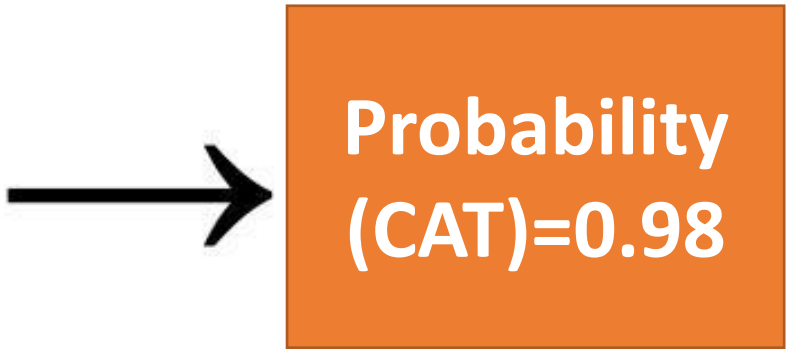
Your trained model!





Probability (DOG)=0.93





Probability (CAT)=0.98

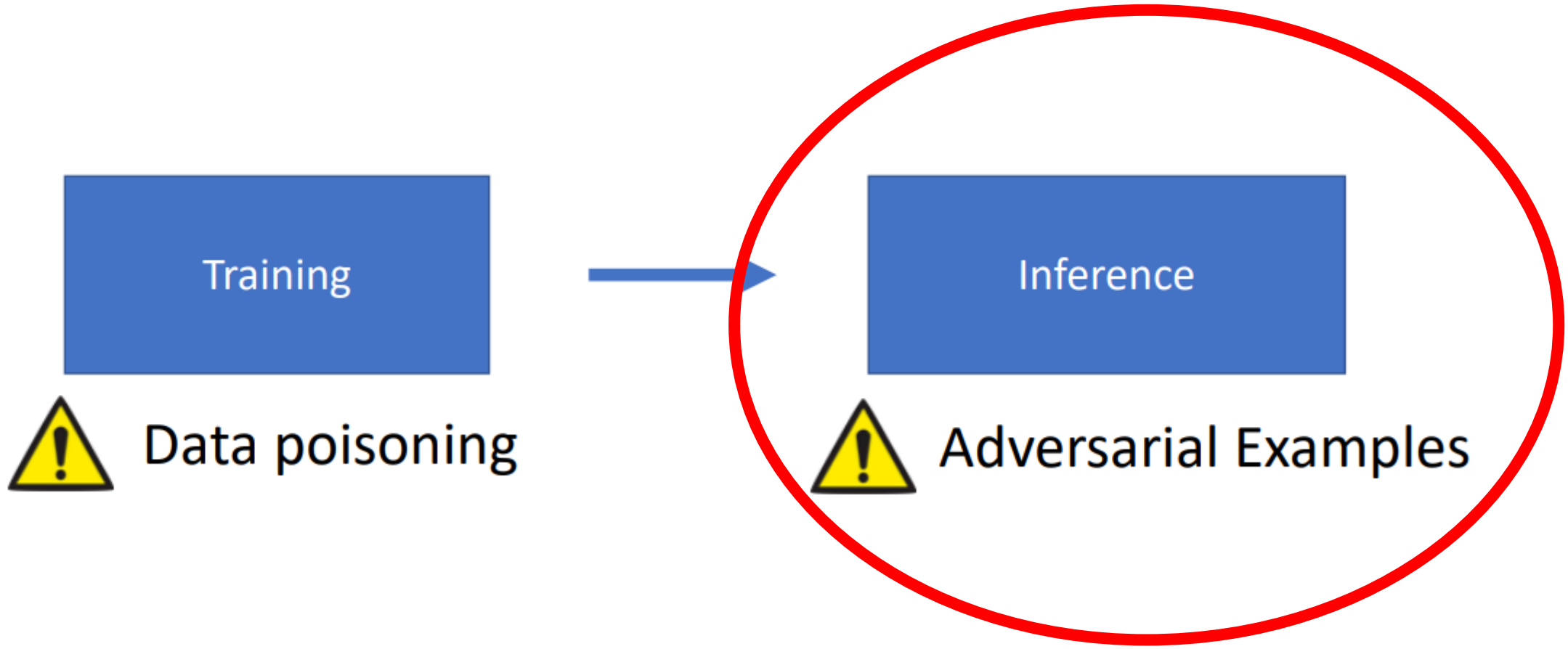
Exercise 1: Train your own model

Complete Babby's First Classification Model on the provided notebook.


BREAK
TIME!!!

Threat Model

Overview



Threat Models

- Attacker's Goals
  - Security Violation
    - **Integrity**: can we make our malicious activities evade detection without compromising system operation?
    - **Availability**: can we make the system perform badly (or not performant at all?)
    - **Privacy**: can we steal private data from the model or reverse engineer the algorithm?
  - Attack Specificity
    - **Targeted:** Misclassify a set of specific examples to a specific user
    - **Indiscriminate:** Misclassify any sample to any user
  - Error Specificity
    - **Specific**: Misclassify to a specific class
    - **Generic:** Misclassify to any class

Attacker's Knowledge

- **Varying assumptions on the knowledge of the attacker under these parameters**
  - The training data
  - The feature set (how the training data is represented)
  - The learning algorithm (together with the objective function learnt by the defender)
  - The trained parameters **w**
- **Can be classified into**
  - **Perfect Knowledge Attacks** – worst case behavior of the system (upper bounds on performance degradation)
  - **Limited Knowledge Attacks –** practical attacks are studied, including limited knowledge of training data (Universal Perturbations), or black box models (transferability of adversarial examples)

Attacker’s Capability

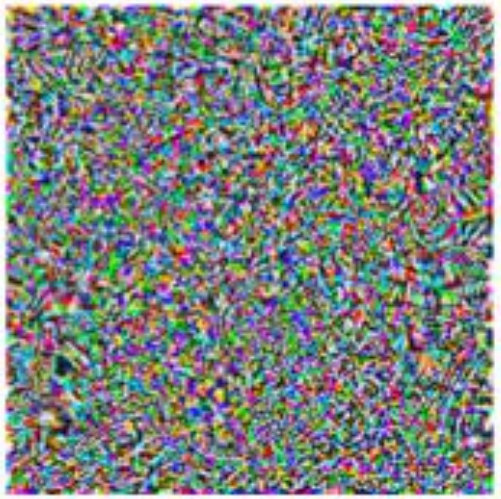
- Attack Influence
  - Training Data: Data Poisoning Attacks
  - Test Data: Evasion Attacks
- Data Manipulation Constraints
  - Constraints on percentage of data controlled
  - Constraints on perturbations to test time data

Evasion Attacks

(Adversarial Examples)

The big picture

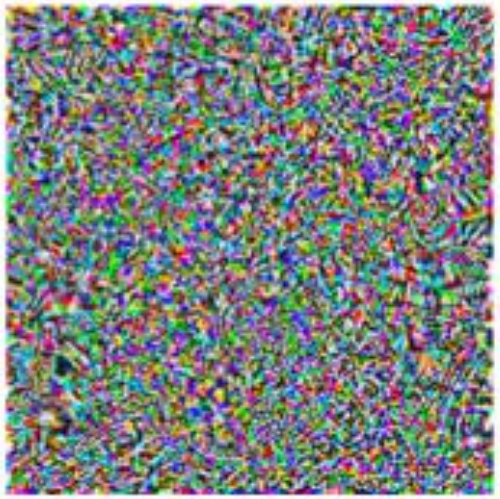
$y =$ “pig” “airliner” $= y'$

$x =$  + 0.005 x  =  $= x'$

δ

The big picture

$y =$ "pig" $= y'$

$x =$  + 0.005 x  =  $= x'$

δ

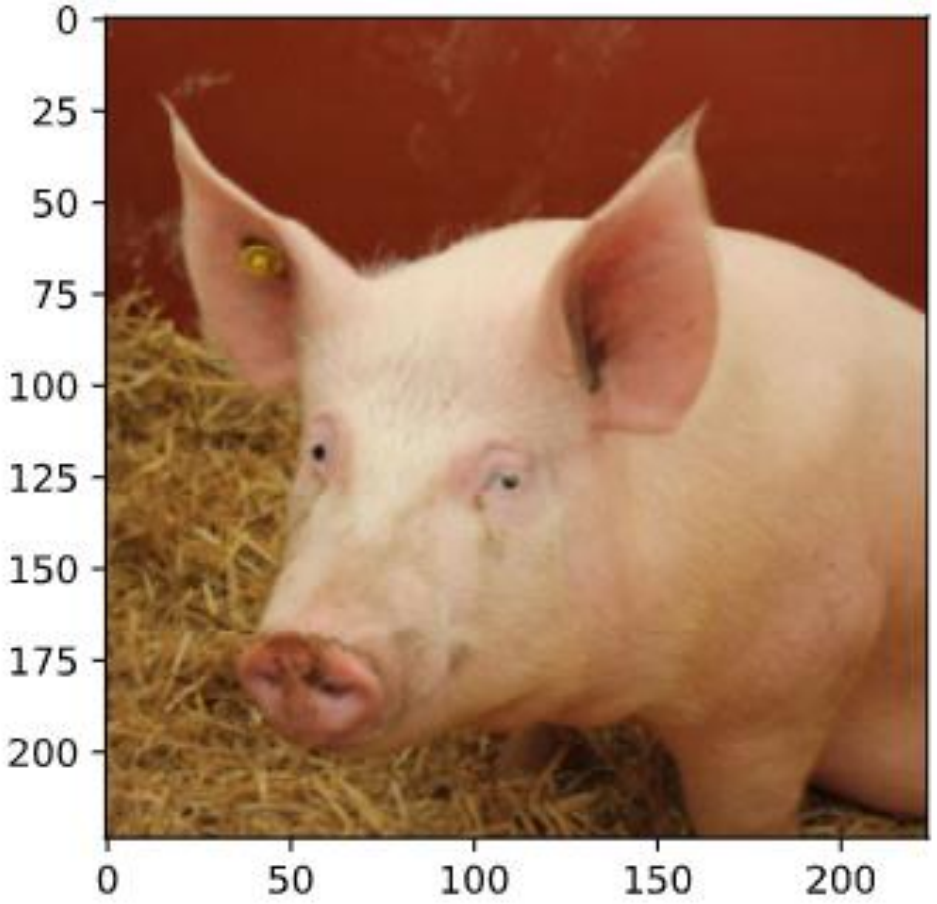
Some noise that we add to the original image

$$\max_{\delta \in \Delta} \text{Loss}\left(x + \delta, y; \theta\right)$$

The noise must be small

Such that the model performs very poorly (have a **very high/max loss**)

Our data is just a big ol’ tensor

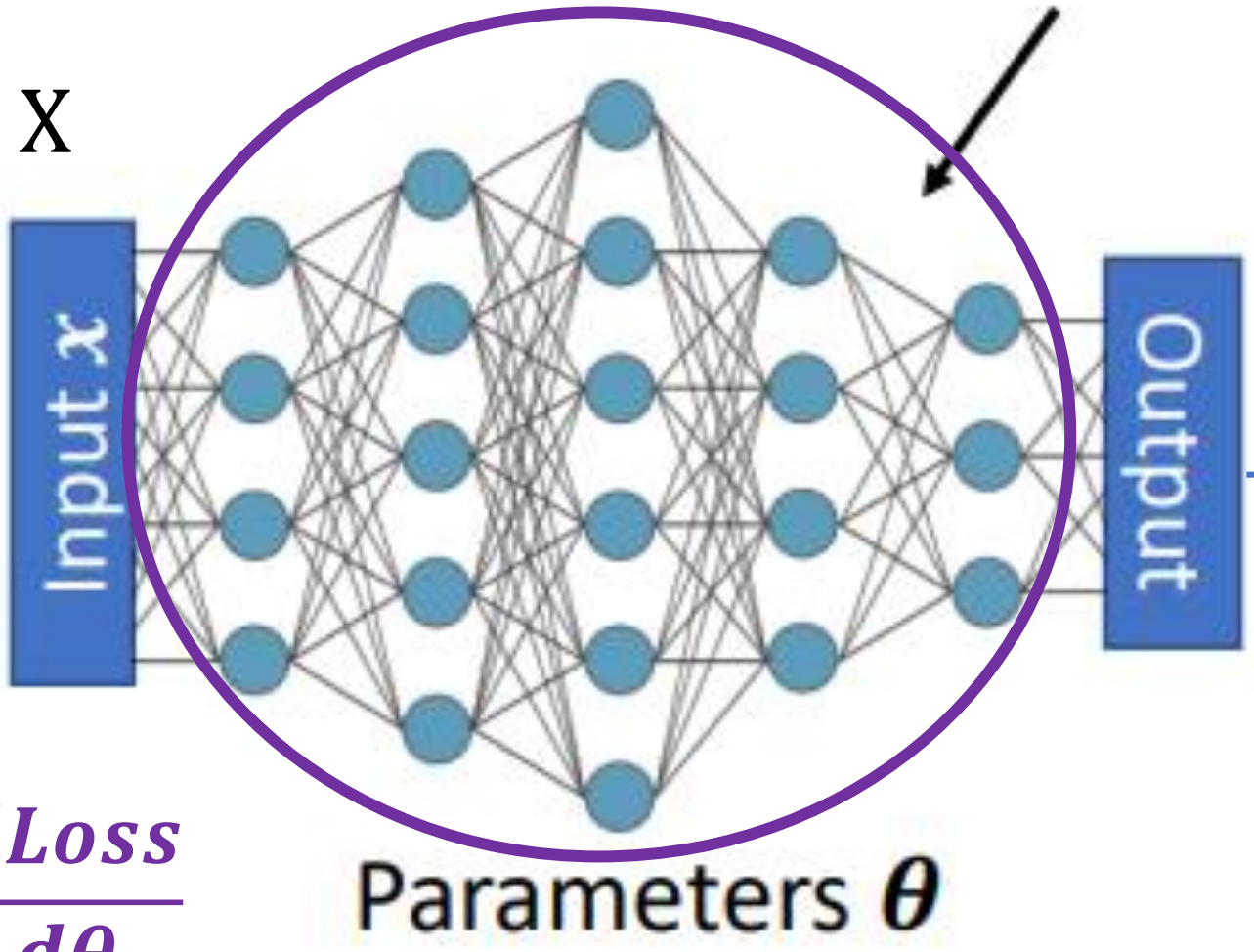


$x \in \mathrm{X}$

Some practice is available at the start of the notebook!

Normal Training

$x \in X$



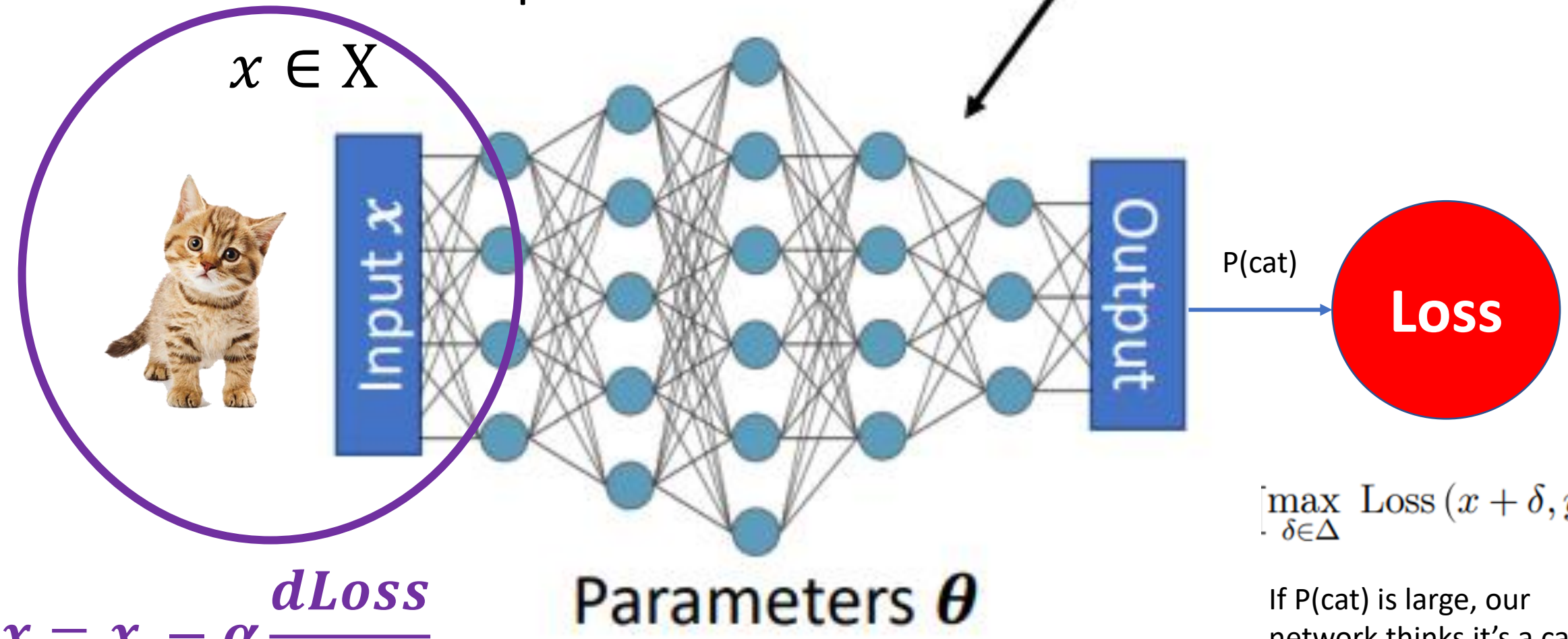
P(cat)

Loss

Penalize misclassification (CrossEntropyLoss)

$$\theta = \theta - \alpha \frac{dLoss}{d\theta}$$

Finding Adversarial Input



$$x = x - \alpha \frac{dLoss}{dx}$$

$$\max_{\delta \in \Delta} \text{Loss}\,(x + \delta, y; \theta)$$

If P(cat) is large, our network thinks it's a cat, so we make the Loss **BIG** to turn it into a dog

It’s Gonna Get Mathy from Here On Out!

What the model designer wants

Objective function

$$\underset{\theta}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} \ell(h_\theta(x_i), y_i)$$

Loss of model output compared to true labels y_i

Parameter update

$$\theta := \theta - \frac{\alpha}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \ell(h_\theta(x_i), y_i)$$

Gradient of loss function with respect to θ

End result: Defender has some optimal θ based on the empirical samples

What the attacker wants

Objective function

$$\underset{\hat{x}}{\text{maximize}}\ \ell(h_\theta(\hat{x}), y)$$

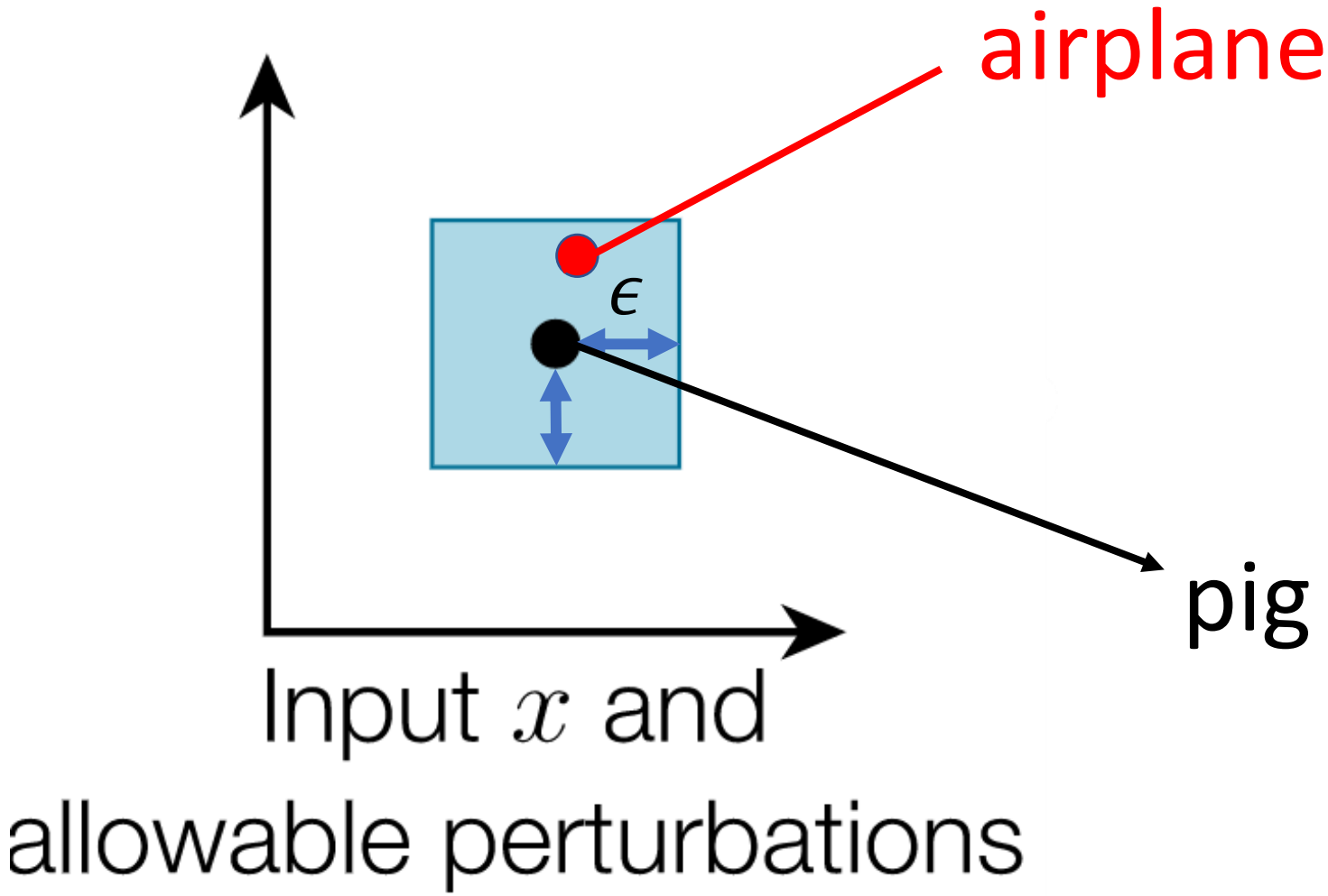
Find adversarial example $\hat{x}$ that maximizes the loss compared to the true label y

But we want $\hat{x}$ to be 'imperceptibly different'

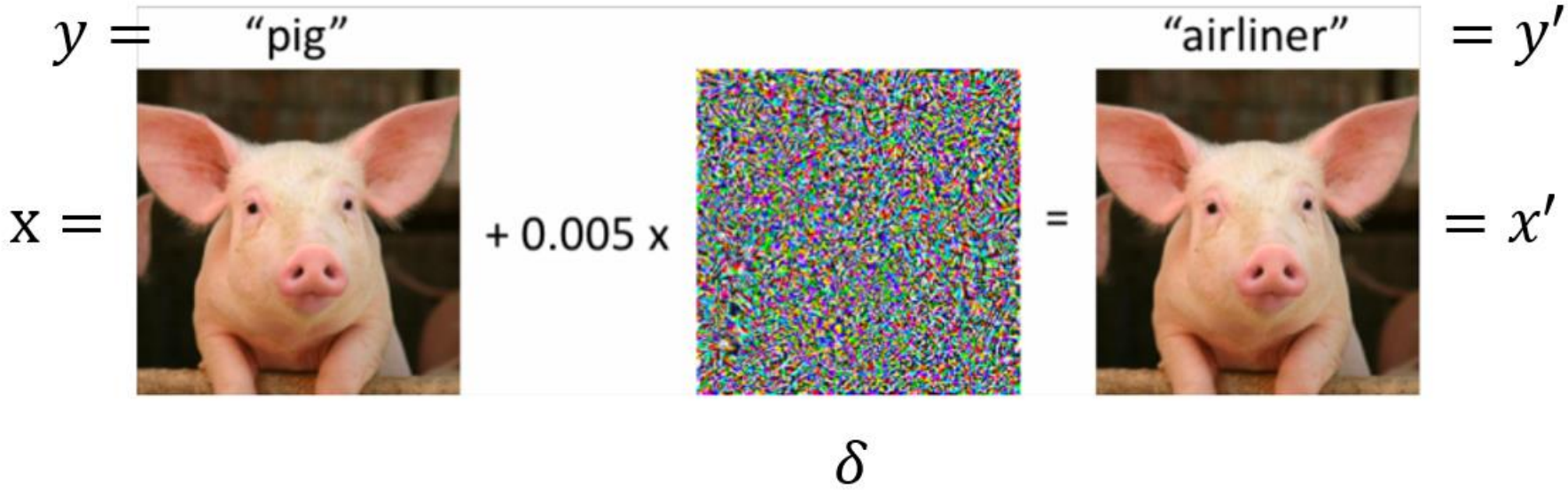
$$\underset{\delta \in \Delta}{\text{maximize}}\ \ell(h_\theta(x + \delta), y)$$

e.g. $\Delta = \{\delta : \|\delta\|_\infty \leq \epsilon\}$ $\quad \|z\|_\infty = \max_i |z_i|$

What's l-inf norm?

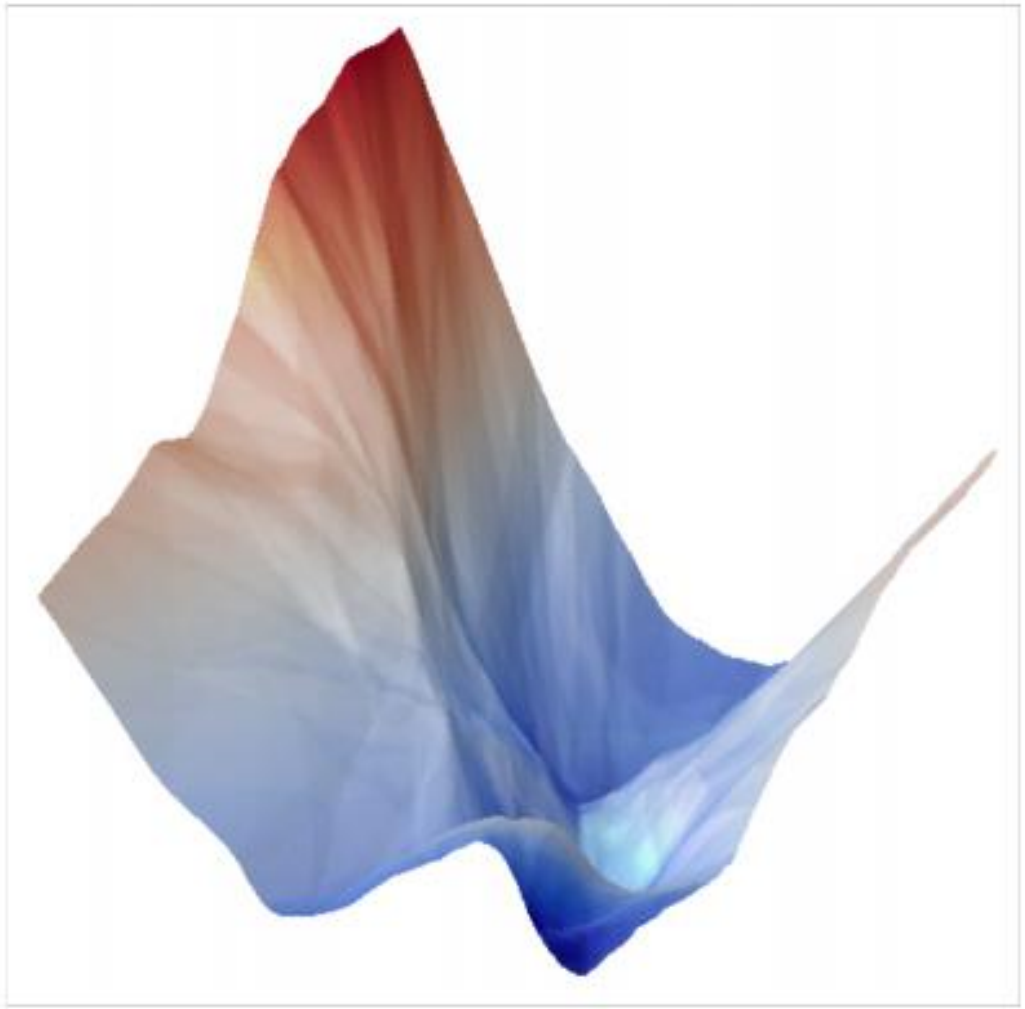


Why does finding an adversarial input in some restricted space matter?



- We don't really understand **why** the model is failing
  - Picture still looks visually like a pig
  - Seemingly unexpected "*random*" failure
  - We need to understand how this failure arises so that vital applications of ML **works as expected**

Local search



For neural networks, the loss landscape is non-convex, inner maximization problem is difficult to solve exactly

This has never stopped us before in deep learning … let's just find an approximate solution using gradient-based methods

Finding Adversarial Input

$x \in X$

Differentiable

Input x

Output

P(cat)

Loss

$$\max_{\delta \in \Delta} \text{Loss}\,(x + \delta, y; \theta)$$

Parameters $\boldsymbol{\theta}$

$$x = x - \alpha \frac{dLoss}{dx}$$

Backpropagate Loss

If P(cat) is large, our network thinks it's a cat, so we make the Loss **BIG** to turn it into a dog

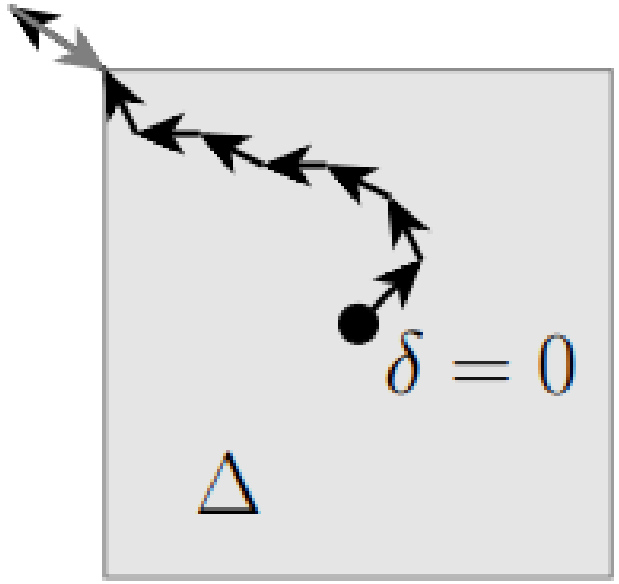
However, change to x needs to be small!

Projected gradient descent

Recall we are optimizing

$$\max_{\delta \in \Delta} \text{Loss}\,(x + \delta, y; \theta)$$

$\alpha \nabla_\delta \text{Loss}(x + \delta, y; \theta)$



We can employ a projected gradient descent method, take gradient step and project back into feasible set Δ

$$\delta := \mathcal{P}_\Delta[\delta + \nabla_\delta \text{Loss}(x + \delta, y; \theta)]$$

e.g. $\Delta = \{\delta : \|\delta\|_\infty \leq \epsilon\}$ $\quad \delta := \text{Clip}_\epsilon[\delta + \nabla_\delta \text{Loss}(x + \delta, y; \theta)]$

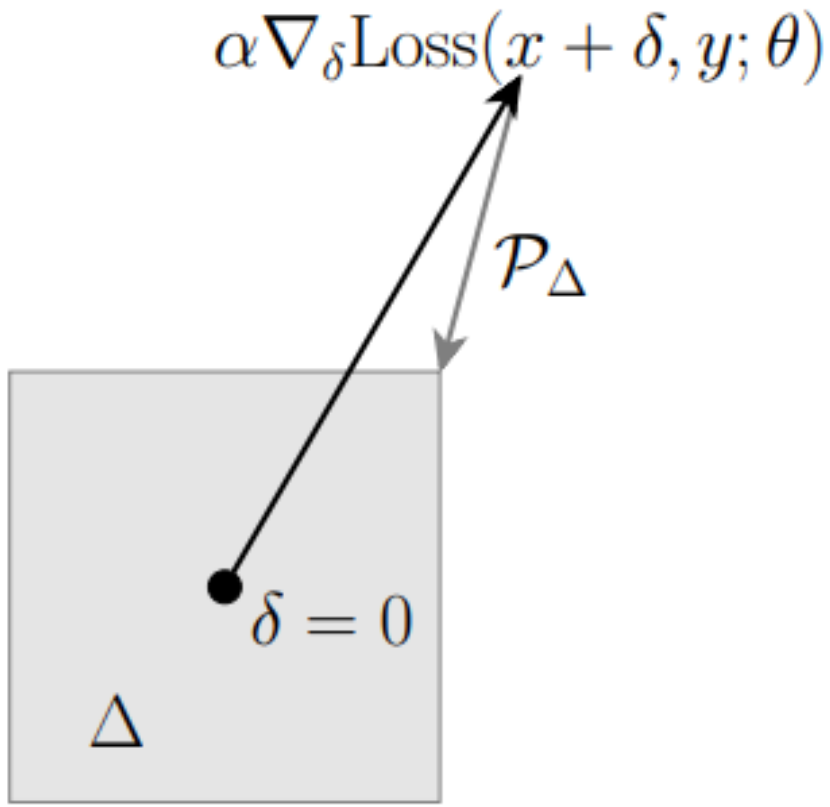
The Fast Gradient Sign Method (FGSM)

To be more concrete, take Δ to be the ℓ_∞ ball, $\Delta = \{\delta : \|\delta\|_\infty \leq \epsilon\}$, so projection takes the form

$$P_\Delta(\delta) = \text{Clip}(\delta, [-\epsilon, \epsilon])$$

As $\alpha \to \infty$, we always reach "corner" of the box, called fast gradient sign method (FGSM) [Goodfellow et al., 2014]

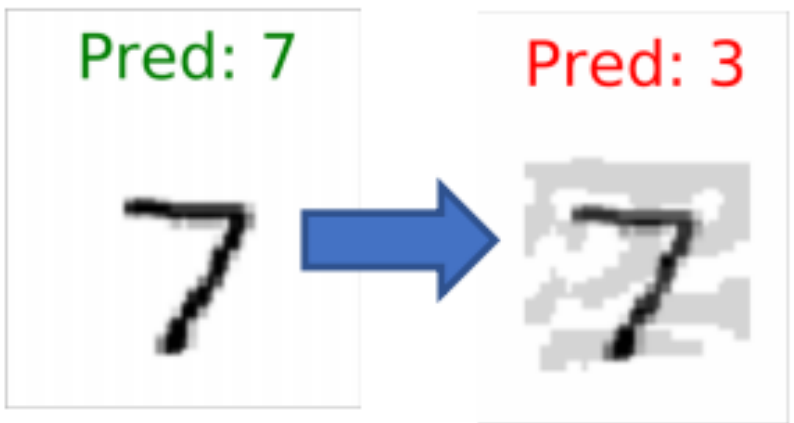
$$\delta = \epsilon \cdot \text{sign}\big(\nabla_\delta \text{Loss}(x + \delta, y; \theta)\big)$$

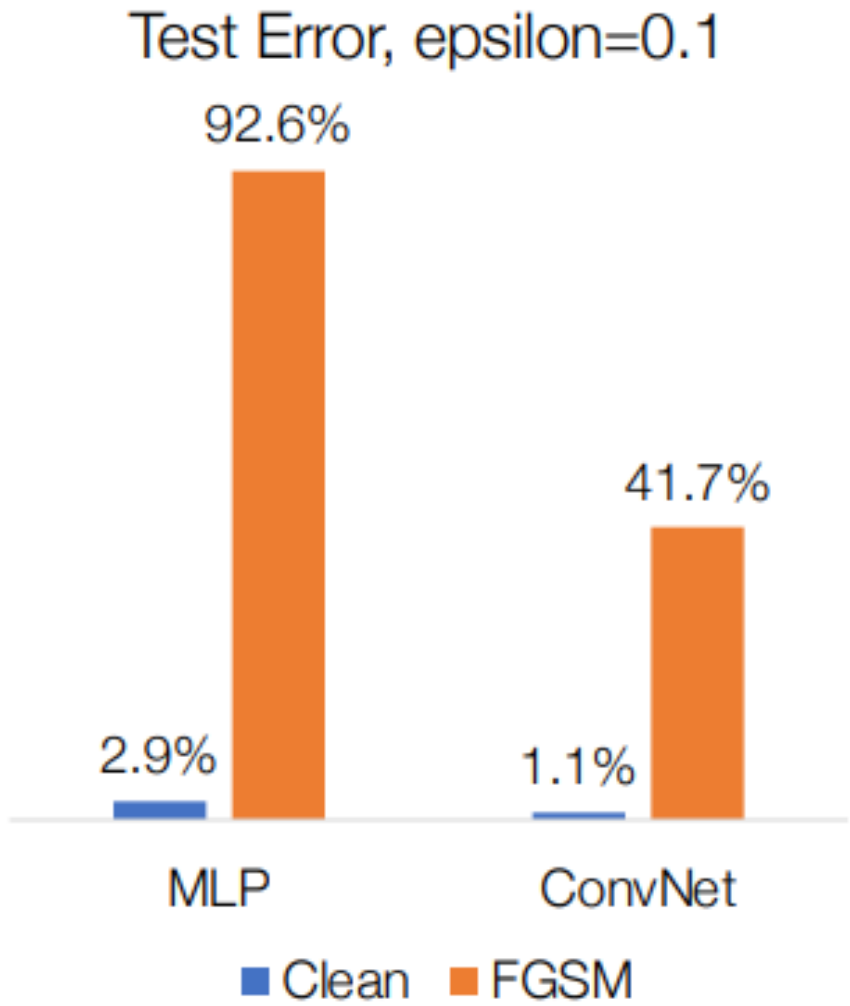


Illustrations of FGSM

MLP:

ConvNet:





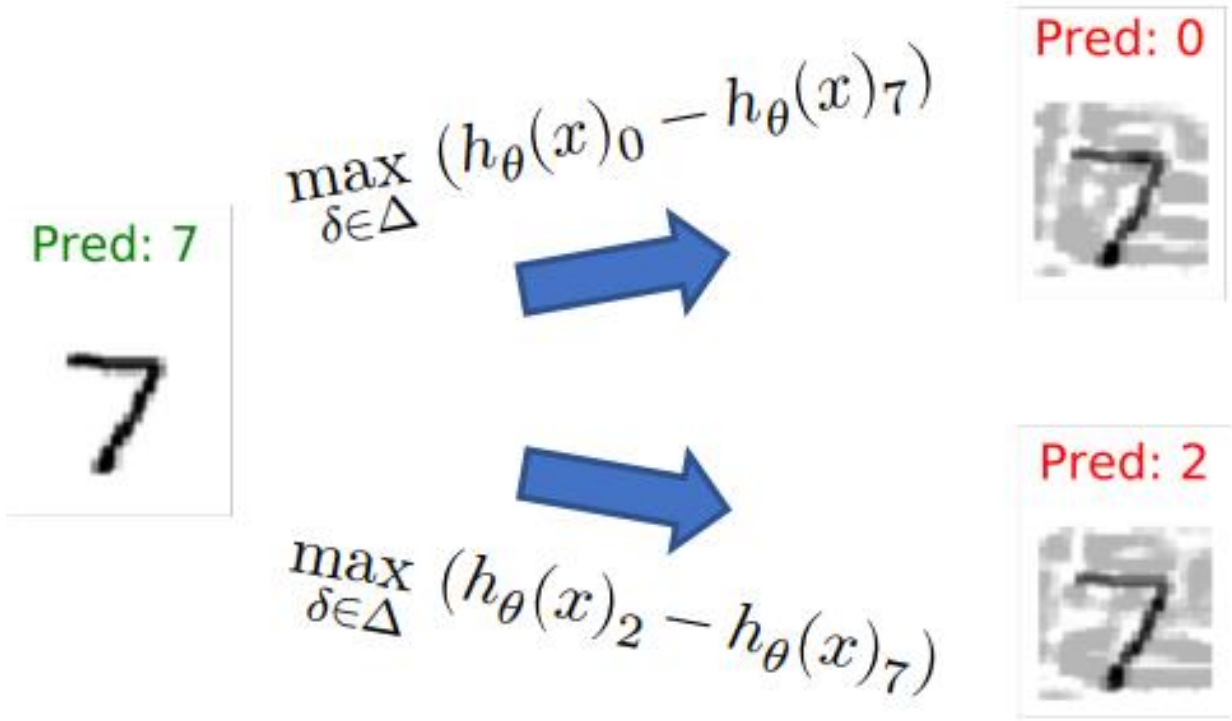
Targeted Attacks

Also possible to explicitly try to change label to a *particular* class

$$\max_{\delta \in \Delta} \left(\text{Loss}(x + \delta, y; \theta) - \text{Loss}(x + \delta, y_{\text{targ}}; \theta) \right)$$

Idea: make it perform "better" on the target class!

Targeted attack examples



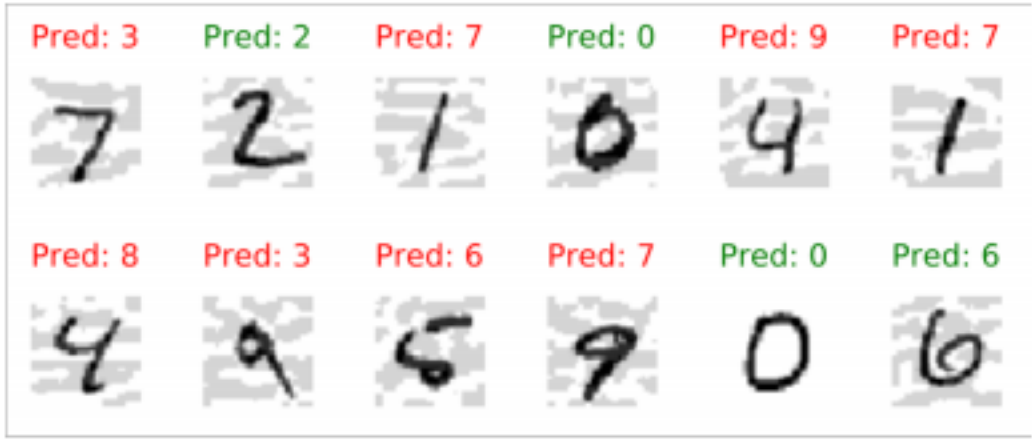
Note: A targeted attack can succeed in "fooling" the classifier, but change to a different label than target

Non-ℓ_∞ norms

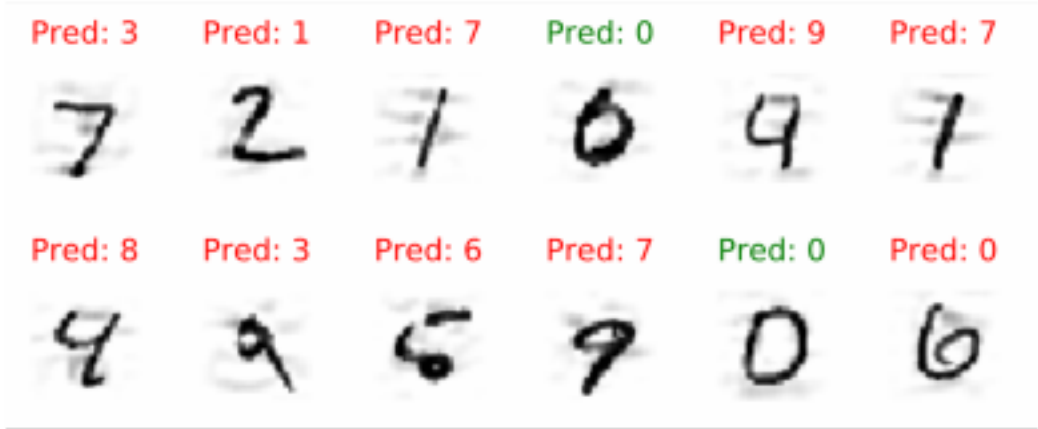
Everything we have done with ℓ_∞ norm possible with other ℓ_p norms

E.g., derive steepest descent for ℓ_2 ball (i.e., normalized gradient descent), projection onto ball

ℓ_∞:



ℓ_2:



Other Methods of Finding Adversarial Examples: **Combinatorial Optimization**

To describe exact combinatorial optimization procedure, we need to more explicitly describe the network; consider a ReLU-based feedforward net

$$
\begin{aligned}
z_1 &= x \\
z_{i+1} &= \mathrm{ReLU}(W_i z_i + b_i), \qquad i = 1, \ldots, d-1 \\
h_\theta(x) &= W_d z_d + b_d
\end{aligned}
$$

Targeted attack in ℓ_∞ norm can be written as the optimization problem

$$
\begin{aligned}
\underset{z_{1:d}}{\text{minimize}} \quad & \left(e_y - e_{y_{\text{targ}}}\right)^T (W_d z_d + b_d) \\
\text{subject to} \quad & z_{i+1} = \mathrm{ReLU}(W_i z_i + b_i), \qquad i = 1, \ldots d-1 \\
& \|z_1 - x\|_\infty \leq \epsilon
\end{aligned}
$$

Solving combinatorial problem

The optimization formulation of an adversarial attack can be written as an binary mixed integer program [e.g., Tjeng et al., 2018, Wong and Kolter, 2018] or as a satisfisiability modulo theories (SMT) problem [e.g. Katz et al., 2017]

In practice, off-the-shelf solvers (CPLEX, Gurobi, etc) can scale to ~100 hidden units, but size depends heavily on problem structure (including, e.g, the size of ϵ)

Certifying robustness

Consider the objective or our optimization problem

$$\left(e_y - e_{y_{\text{targ}}}\right)^T (W_d z_d + b_d)$$

If we solve our the integer program for some y_{targ} and the objective is *positive,* then this is a *certificate* that there exists *no* adversarial example for that target class

If objective is positive for all $y_{\text{targ}} \neq y$, this is a *verified proof* that there exists no adversarial example at all

Certifying examples

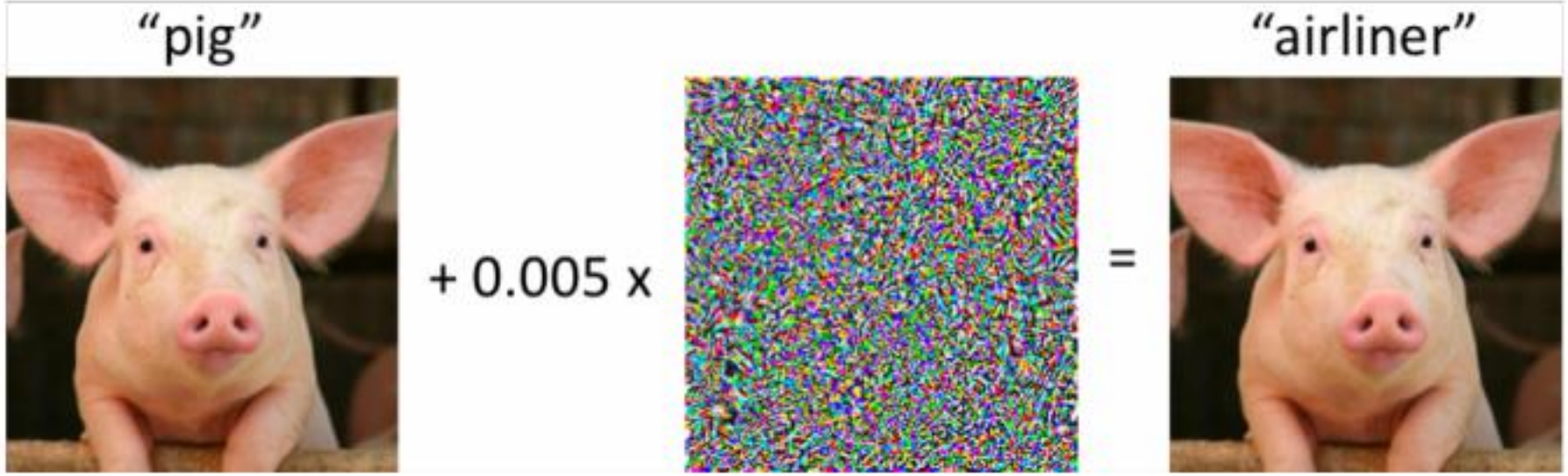
$$\min\ (e_7 - e_0)^T(W_d z_d + b_d)$$
$$\text{s.t.}\ \ \ldots$$

= -2.54 (exists adversarial example for target class zero or another class)

$$\min\ (e_7 - e_1)^T(W_d z_d + b_d)$$
$$\text{s.t.}\ \ \ldots$$

= 3.04 (there is *no* adversarial example to make classifier predict class 1)

The big picture



Part II: training a robust classifier

$$\min_{\theta} \sum_{x,y \in S} \max_{\delta \in \Delta} \text{Loss}\left(x + \delta, y; \theta\right)$$

Part I: creating an adversarial example
(or ensuring one does not exist)

Adversarial training

How do we optimize the objective

$$\min_{\theta} \sum_{x,y \in S} \max_{\delta \in \Delta} \text{Loss}\left(x + \delta, y; \theta\right)$$

We would *like* to solve it with gradient descent, but how do we compute the gradient of the objective with the max term inside?

Danskin's Theorem

A fundamental result in optimization:

$$\nabla_\theta \max_{\delta \in \Delta} \text{Loss}\,(x+\delta, y; \theta) = \nabla_\theta \text{Loss}(x+\delta^\star, y; \theta)$$

where $\delta^\star = \max_{\delta \in \Delta} \text{Loss}\,(x+\delta, y; \theta)$

Seems "obvious," but it is a very subtle result; means we can optimize through the max by just finding it's maximizing value

Note however, it *only* applies when max is performed exactly

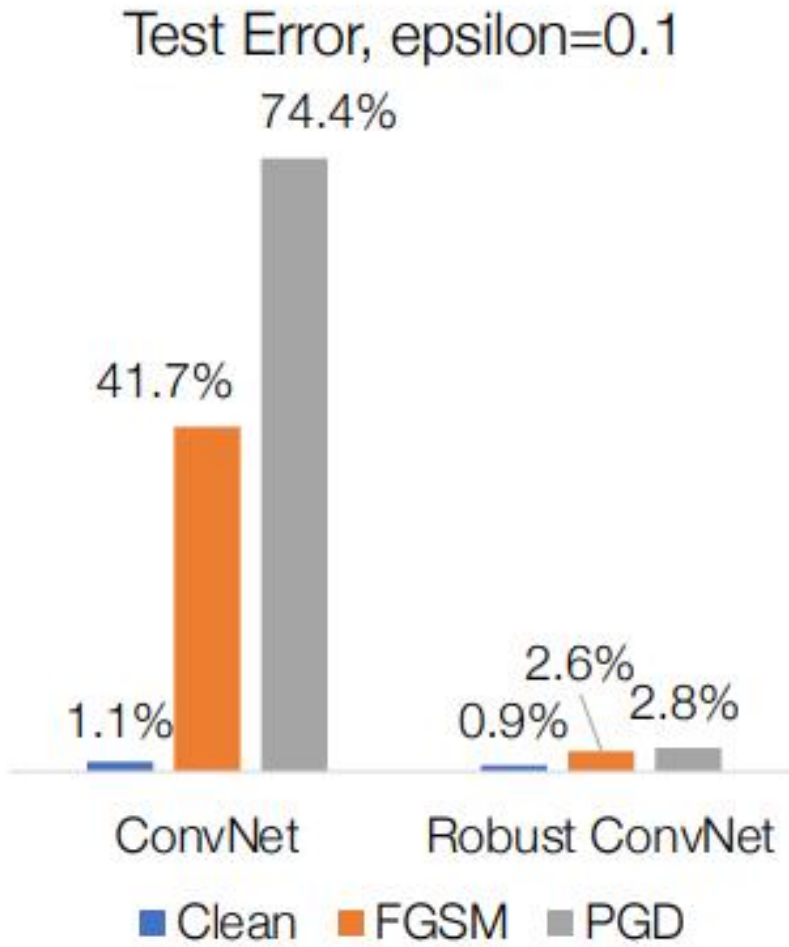
Adversarial training [Goodfellow et al., 2014]

Repeat

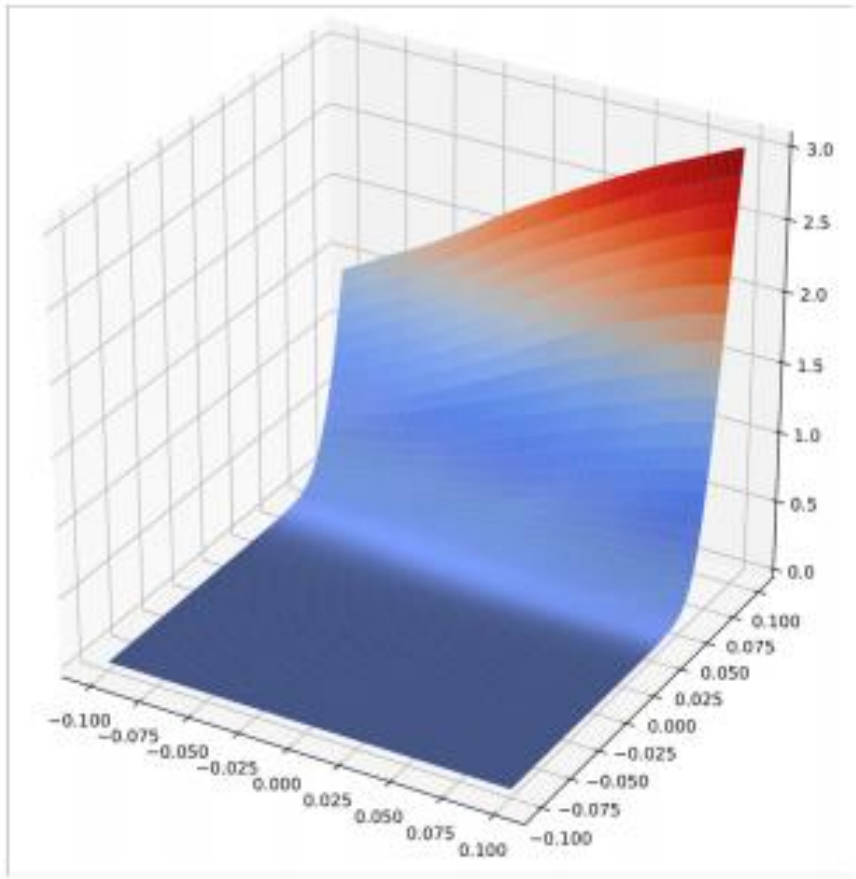
1. Select minibatch B
2. For each $(x, y) \in B$, compute adversarial example $\delta^\star(x)$
3. Update parameters

$$\theta := \theta - \frac{\alpha}{|B|} \sum_{x,y \in B} \nabla_\theta \text{Loss}(x + \delta^\star(x), y; \theta)$$

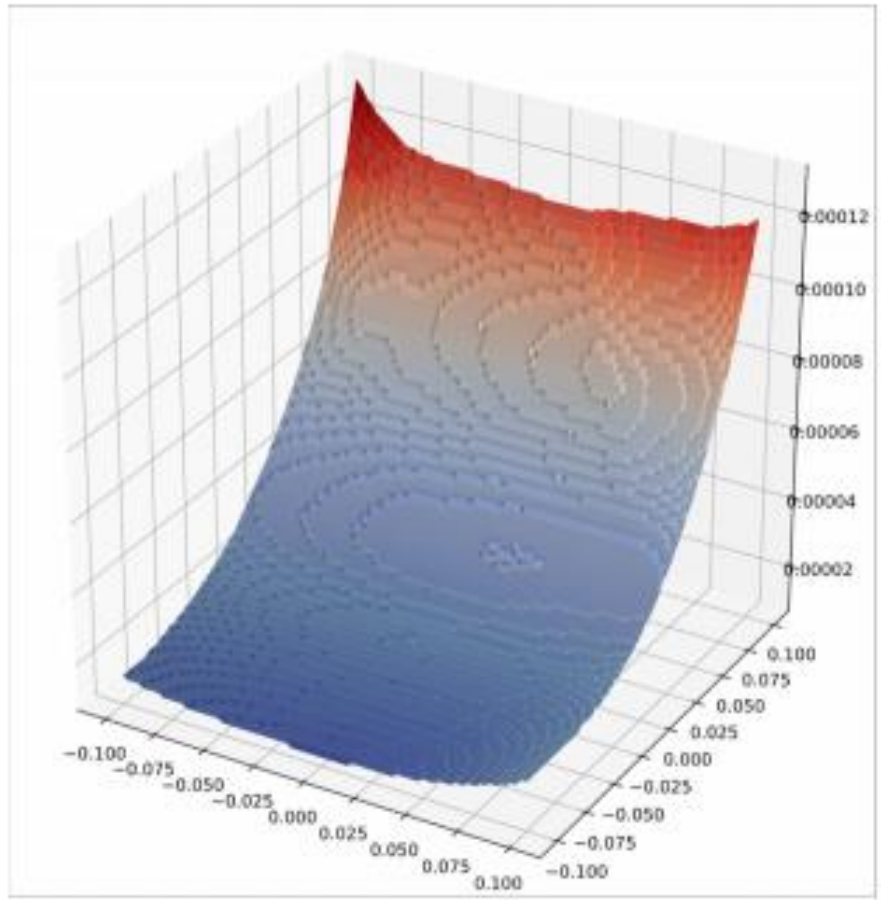
Common to also mix robust/standard updates (not done in our case)



What makes the models robust?



Loss surface:
standard training



Loss surface:
robust training

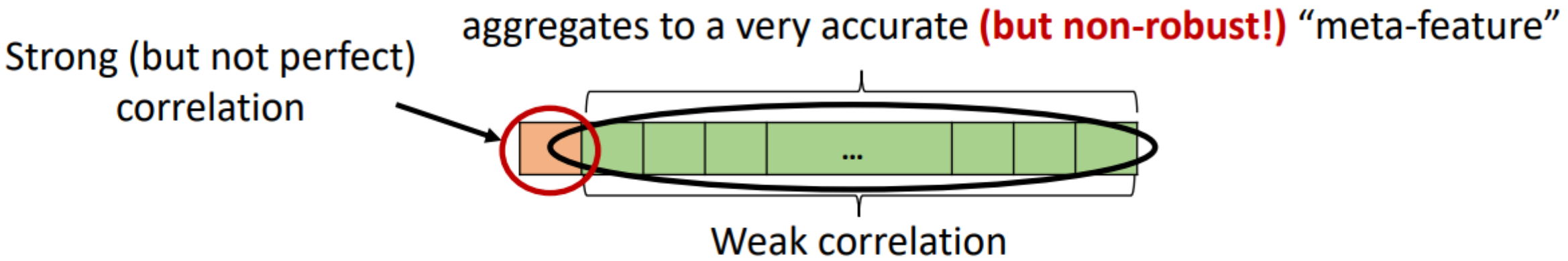
Does Being Robust Help "Standard" Generalization?

Theorem [Tsipras Santurkar Engstrom Turner M 2018]:
No "free lunch": can exist a trade-off between accuracy and robustness

Basic intuition:

→ In standard training, **all correlation is good correlation**

→ If we want robustness, **must avoid** weakly correlated features



Standard training: use all of features, maximize accuracy

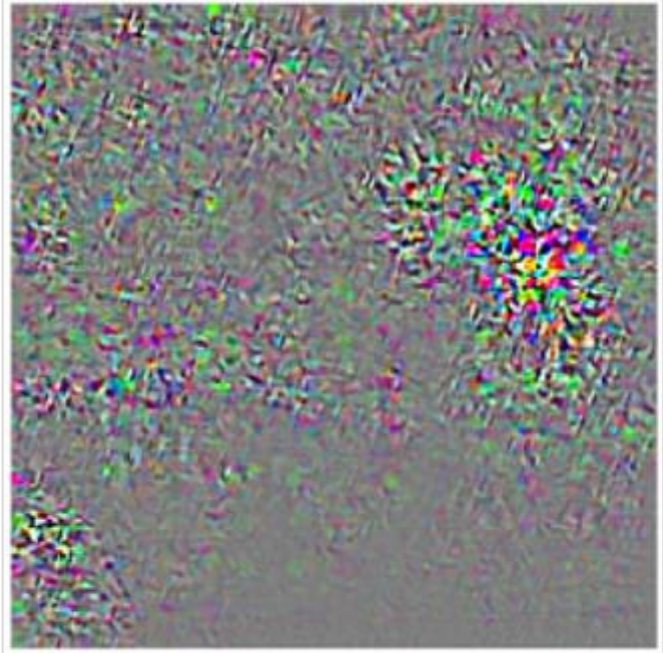
Adversarial training: use only single robust feature (**at the expense of accuracy**)

But There Are (Unexpected?) Benefits Too

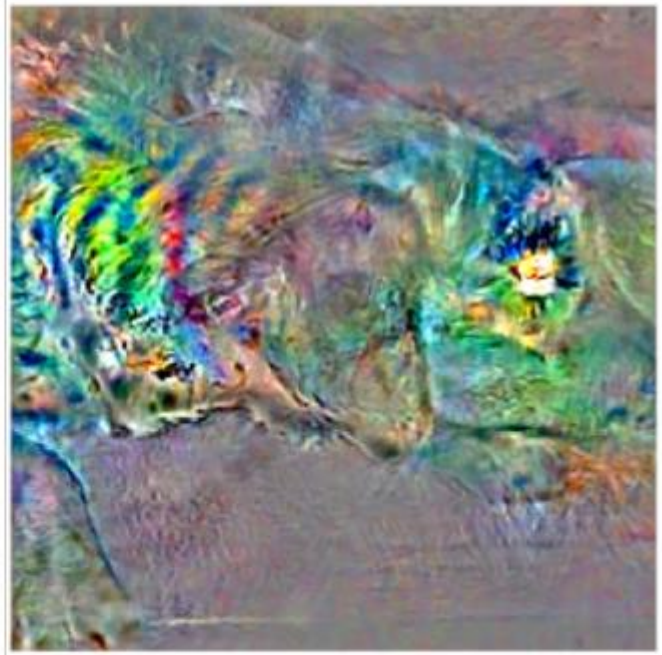
[Tsipras Santurkar Engstrom Turner M 2018]

Models become more **semantically meaningful**

Input



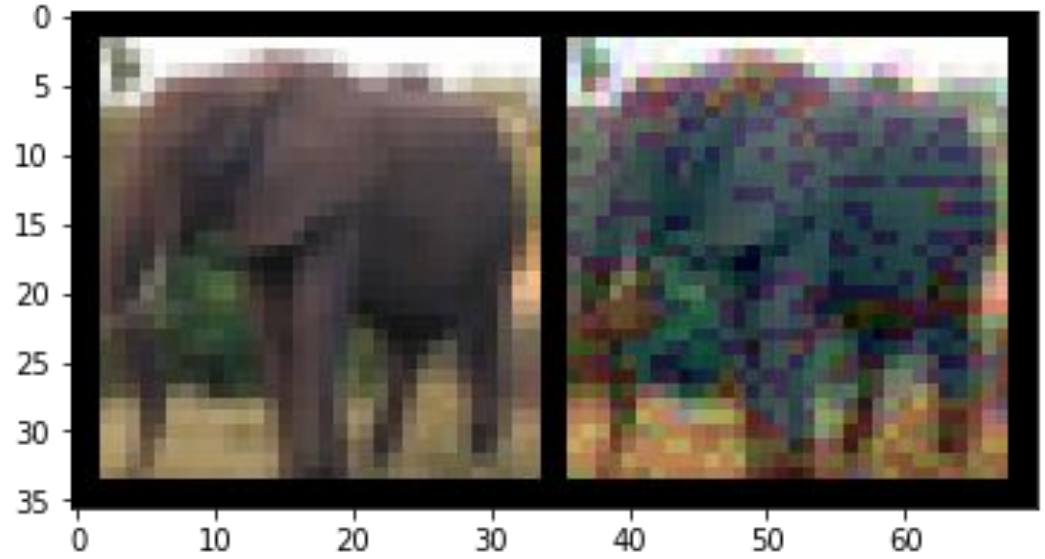
Gradient of
standard model



Gradient of
adv. robust model

Exercise 2: Attack Your Trained Model

We will use I-FGSM (a modified FGSM) to find the adversarial example in Babby's First Adversarial Attack




BREAK
TIME!!!

Case Study: Neural Hash

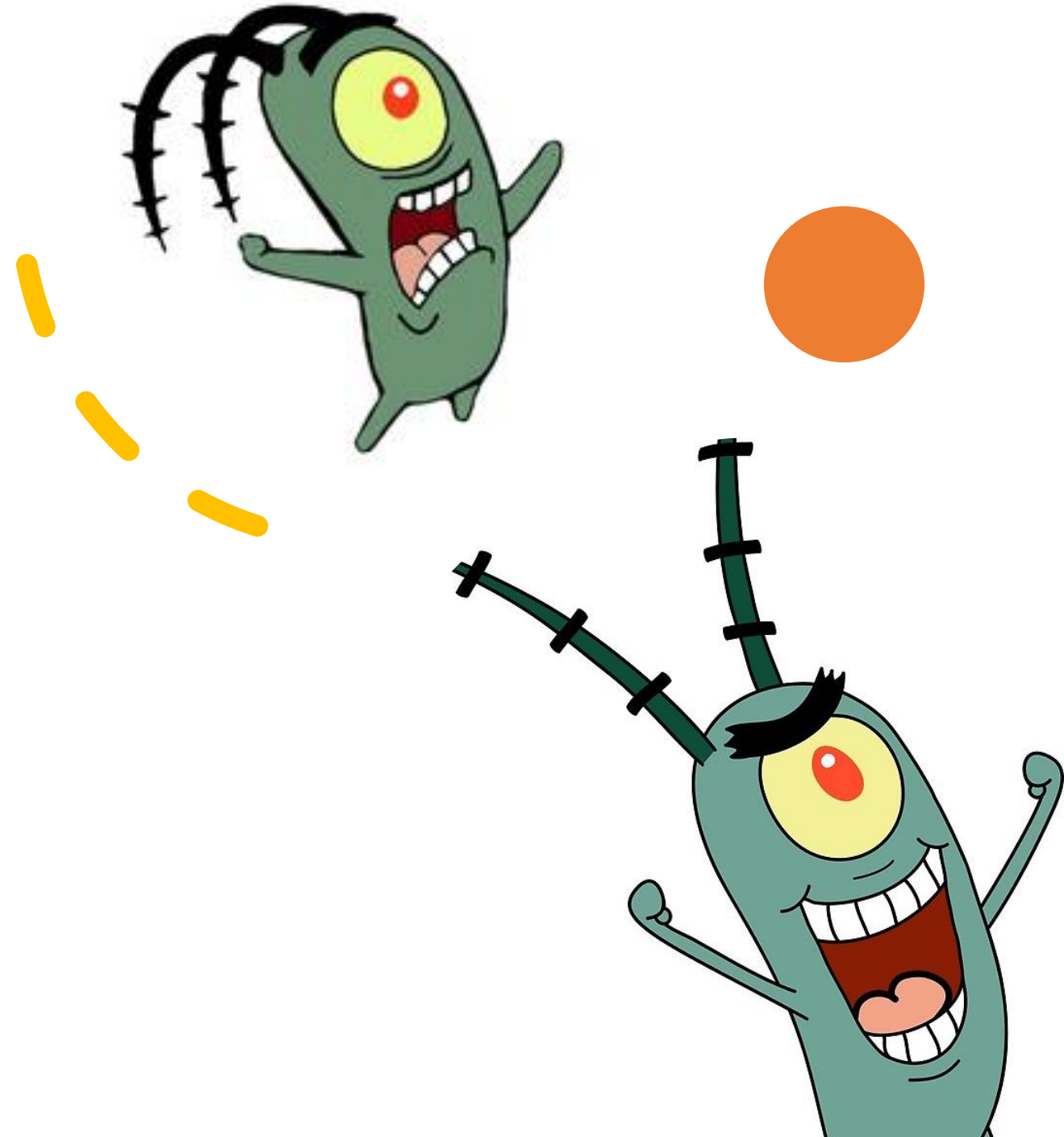

PineApple

Illegal To Store Plankton Pictures

- Pineapple does not allow users of its device any picture of the EVIL PLANKTON!
  - Examples to the right.

- To do this they employed the use of a **Neural Hash**

Note: any resemblance to any real life technology is purely coincidental



Neural Hash

NeuralHash: 01001110101001010 11...

NeuralHash: 01001110101001010 11...

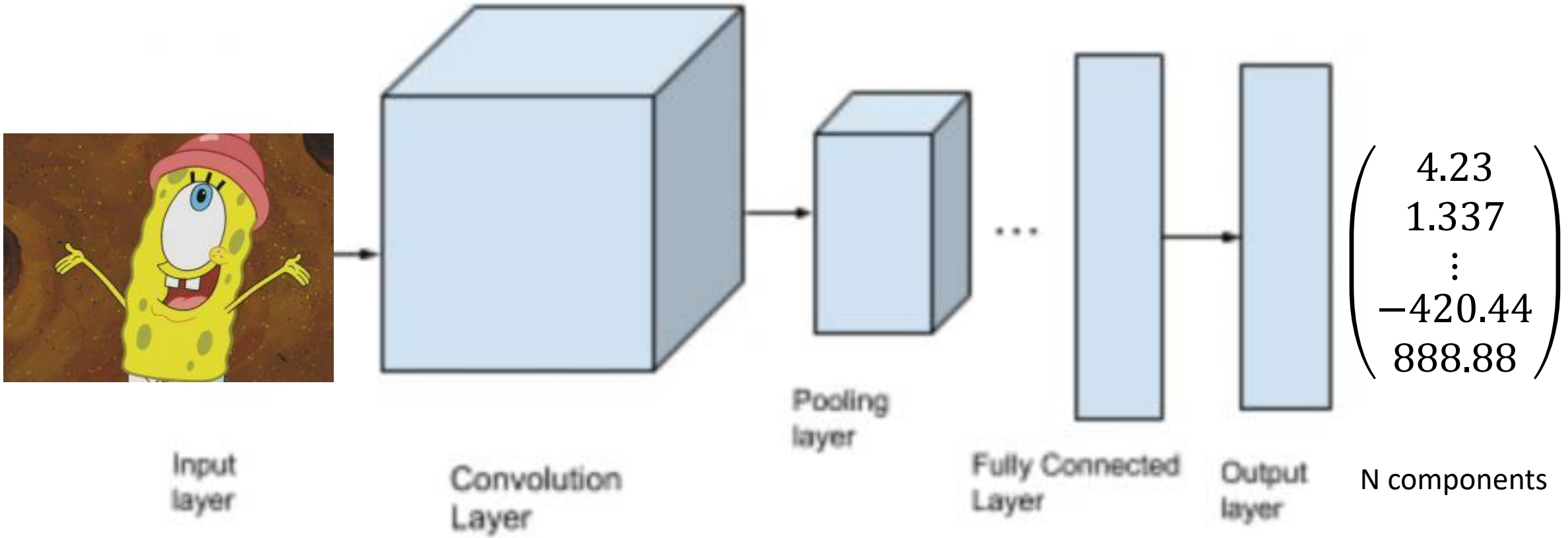
NeuralHash: 11110000010111110 11...

Original image on the left is nearly identical to the center image, therefore both images have the same NeuralHash. The image on the right has different content and hence a different hash.

Threat Model

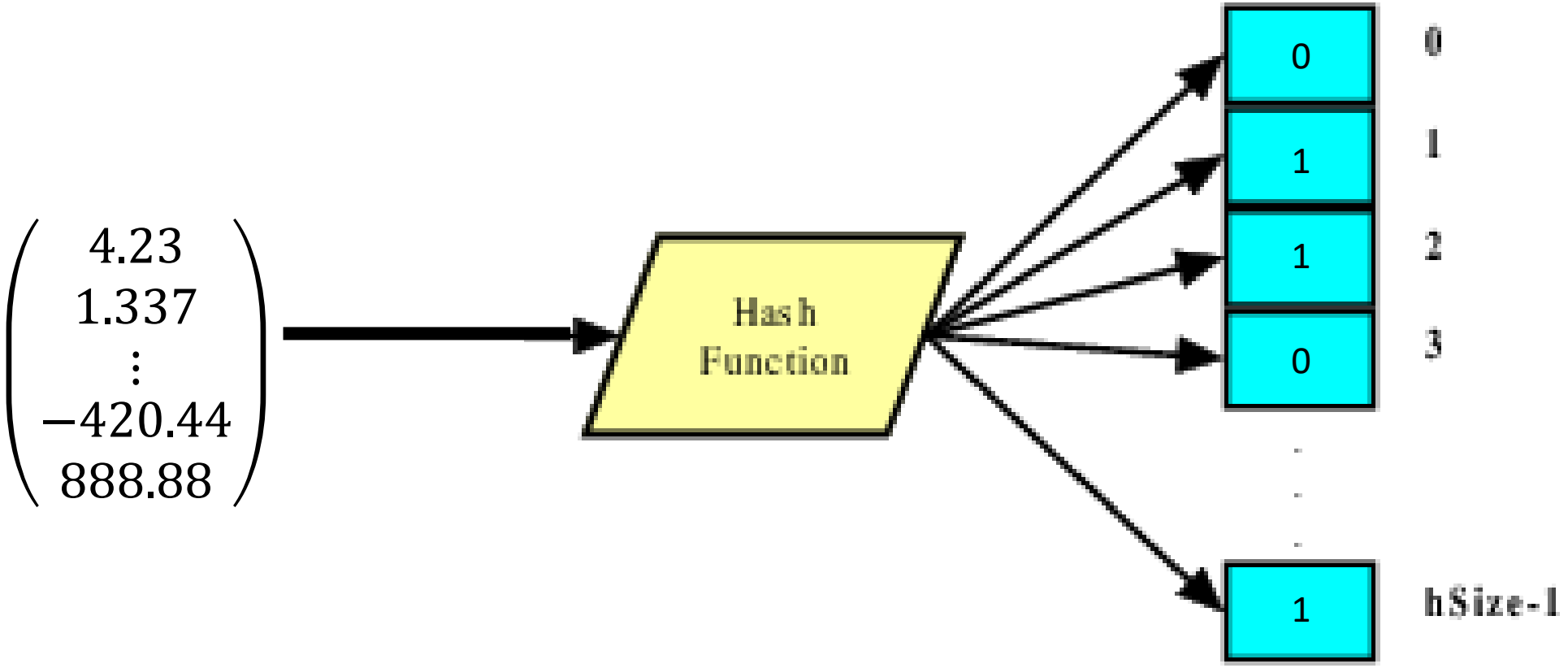
- We shouldn’t be able to recover pictures of Plankton from the neural network
  - Hashes are stored instead of the actual pictures
  - Hashes are stored blinded on the device
- Hashes generated from Plankton are done so with very high confidence
- We shouldn’t be able to peek at other pictures on the device
- There must be extremely high confidence that the neural network will not falsely identify a user as a Plankton lover
  - Threshold of **20 plankton images**
  - **But is this really true?**

Neural Hash Architecture (1)



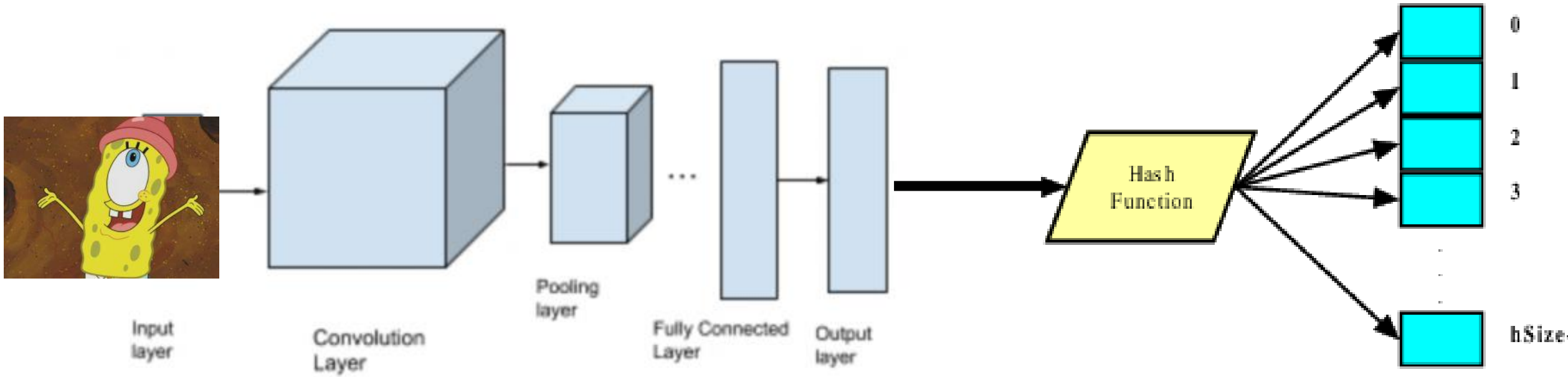
$$\begin{pmatrix} 4.23 \\ 1.337 \\ \vdots \\ -420.44 \\ 888.88 \end{pmatrix}$$

Neural Hash Architecture (2)



0110….01 ➔ 69a34eabe31910abfb06f301

Overall Architecture



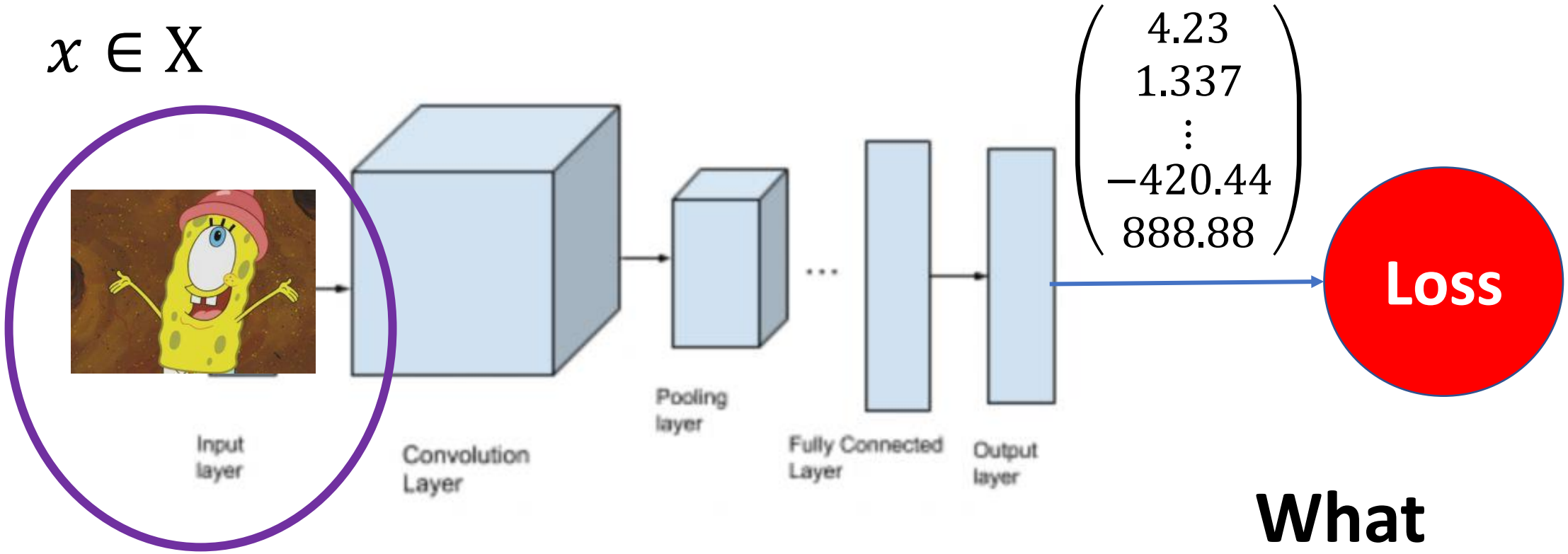
0110….01 ➔ 69a34eabe31910abfb06f301

So How Do We Attack This?

- Brute force attack (upload every single kind of picture)
  - Slow – the space of images are huge!
  - Images are 3x360x360=388800 and each 388800 pixel value can take on 256 possible values
  - Space of values: 2^{4753} ☹
- Can we make use of the fact that it's a neural network somehow?
  - **Differentiable!**

Finding an Adversarial Input

$x \in X$



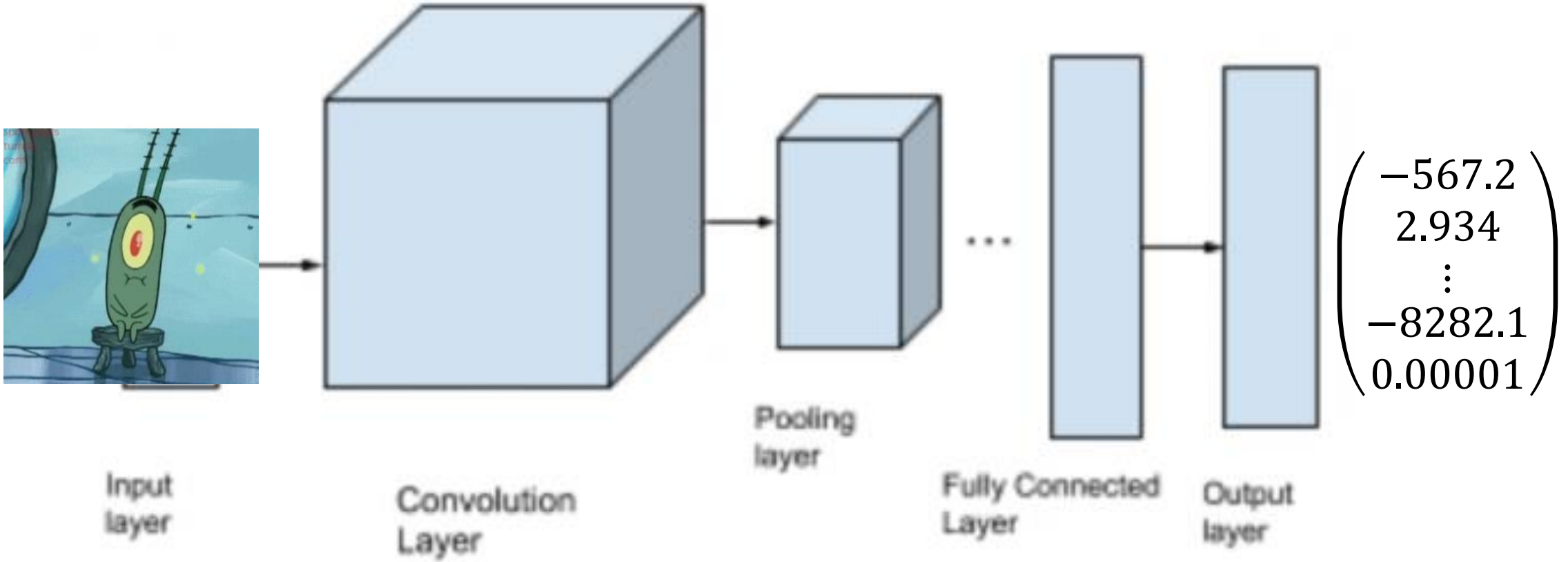
$$\begin{pmatrix} 4.23 \\ 1.337 \\ \vdots \\ -420.44 \\ 888.88 \end{pmatrix}$$

$$x = x - \alpha \frac{dLoss}{dx}$$

Backpropagate Loss

What about the loss?

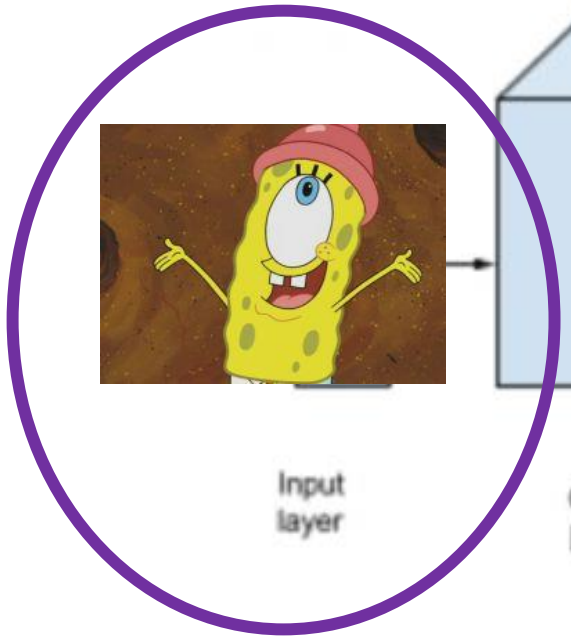
Planktons have a “float representation” too

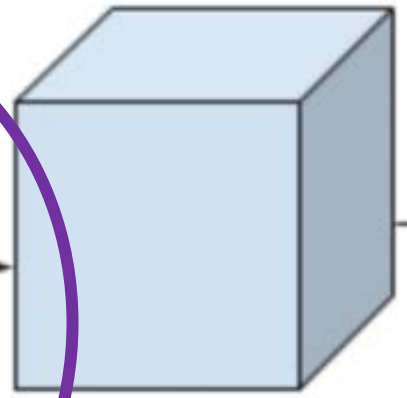


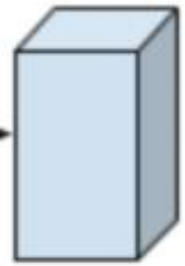
Finding an Adversarial Input

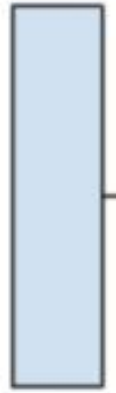
$x \in X$

f









$$\begin{pmatrix} 4.23 \\ 1.337 \\ \vdots \\ -420.44 \\ 888.88 \end{pmatrix}$$

Loss

| f() – f() |

L1 Loss

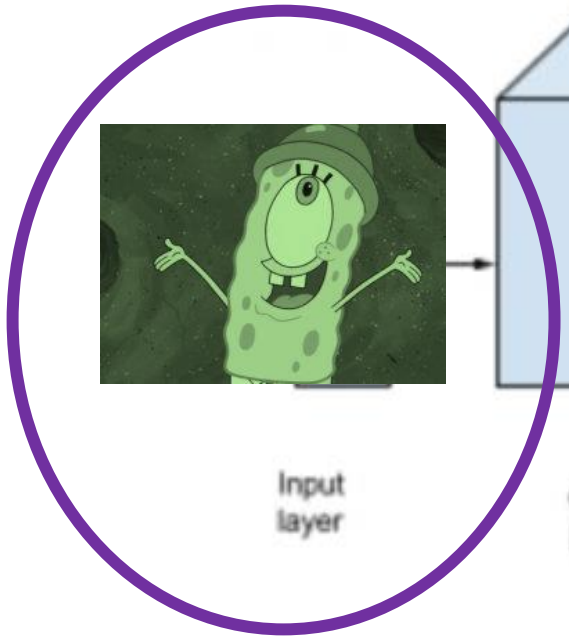
Backpropagate Loss

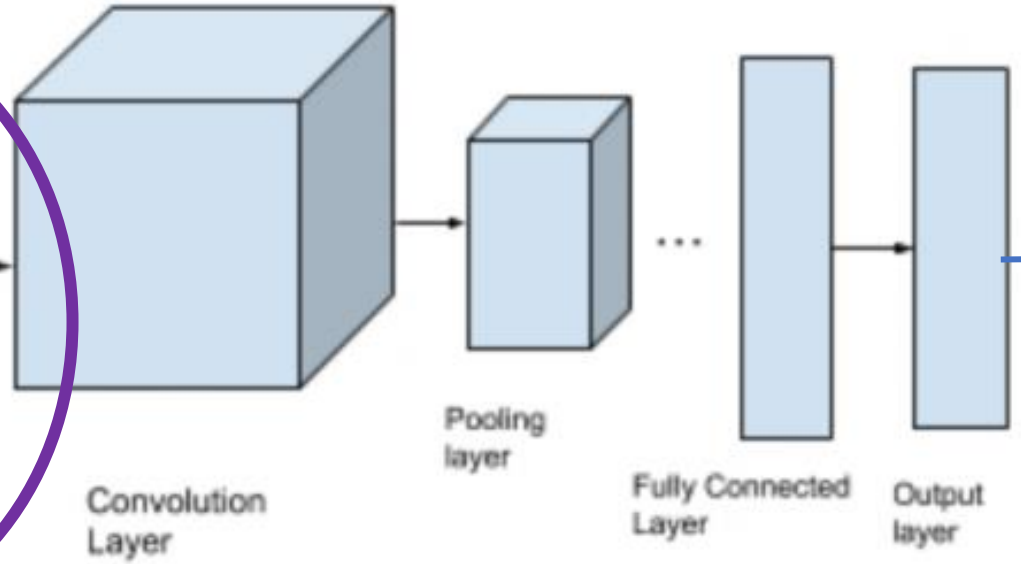
$$x = x - \alpha \frac{dLoss}{dx}$$

Finding an Adversarial Input

$x \in X$

f





$$\begin{pmatrix} -567.2 \\ 2.934 \\ \vdots \\ -8282.1 \\ 0.00001 \end{pmatrix}$$

Loss

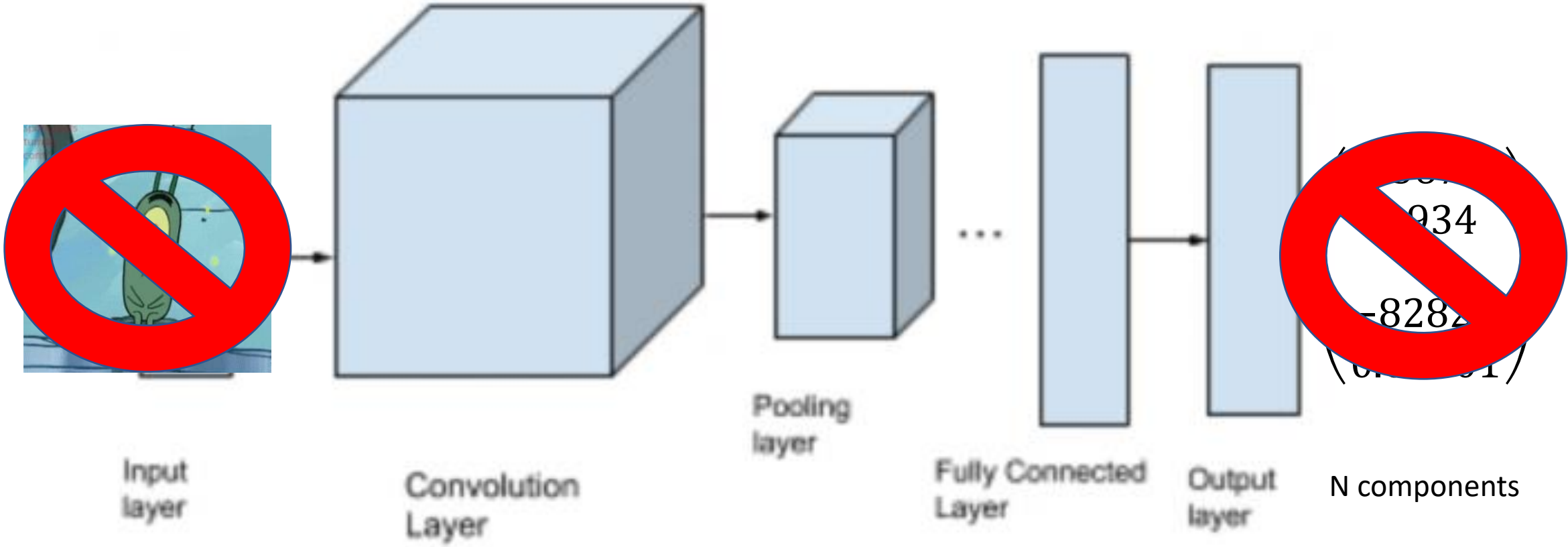
| f() – f() |

L1 Loss

Backpropagate Loss

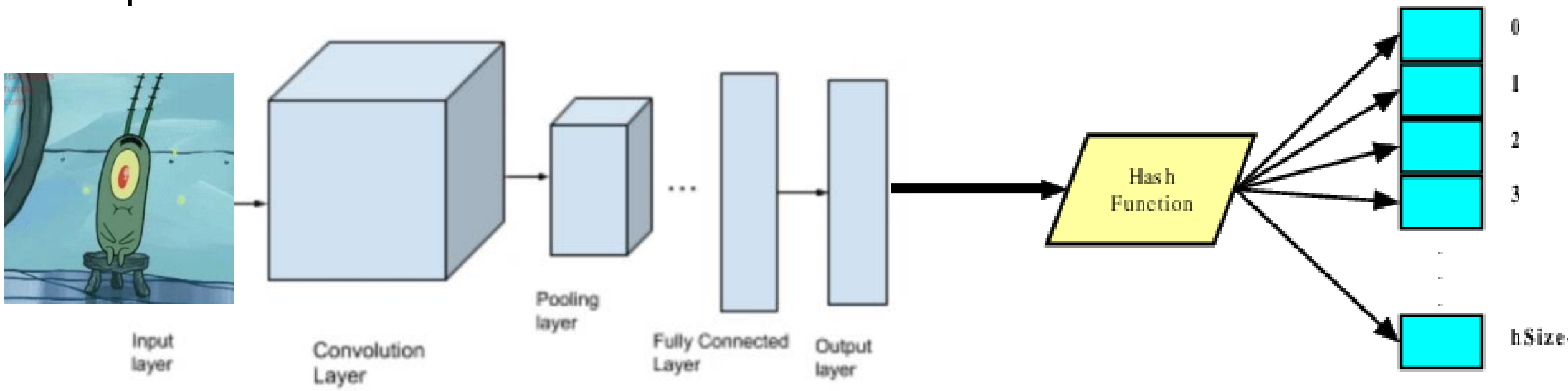
$$x = x - \alpha \frac{dLoss}{dx}$$

But we don’t actually have plankton **pictures**…



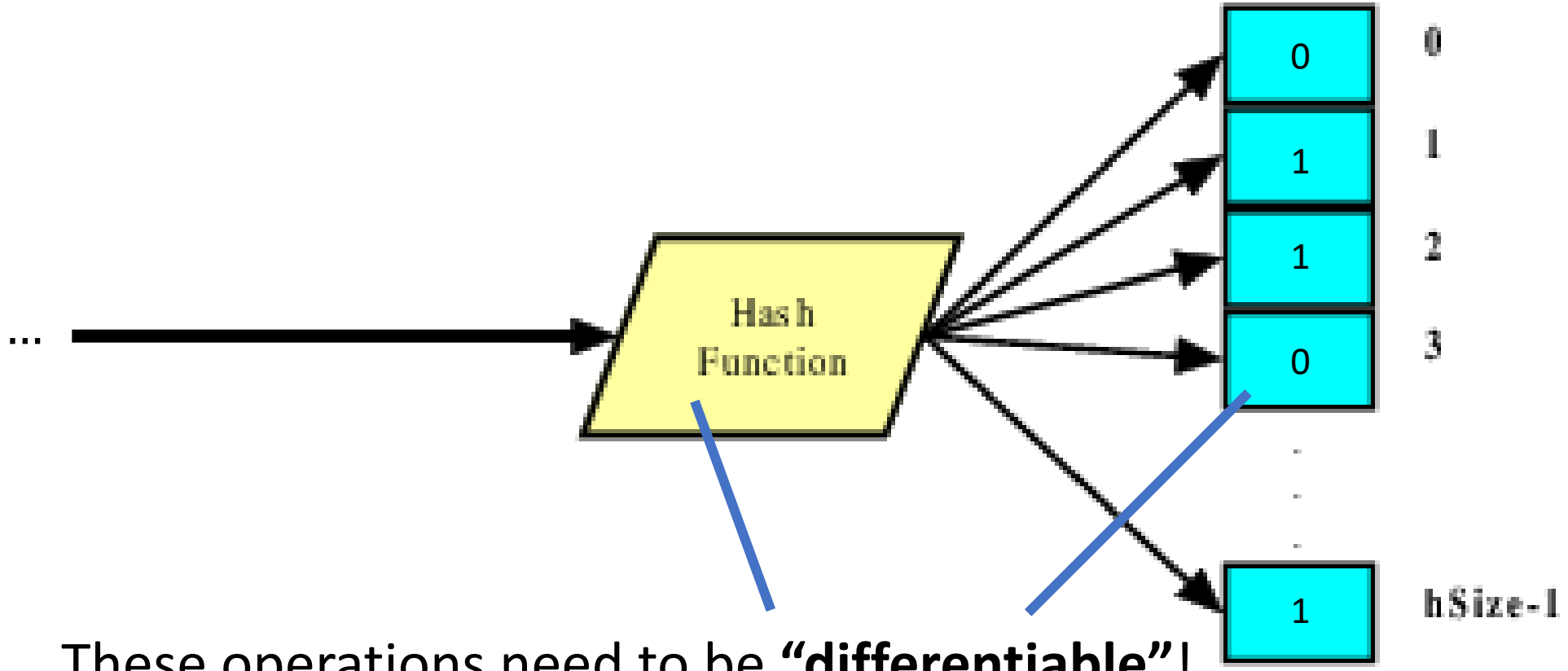
But We Do Have Plankton **Hashes**…

- The hashes are matched against all images on our device, we can just pull them out!



63a12effe31910abfb06f301

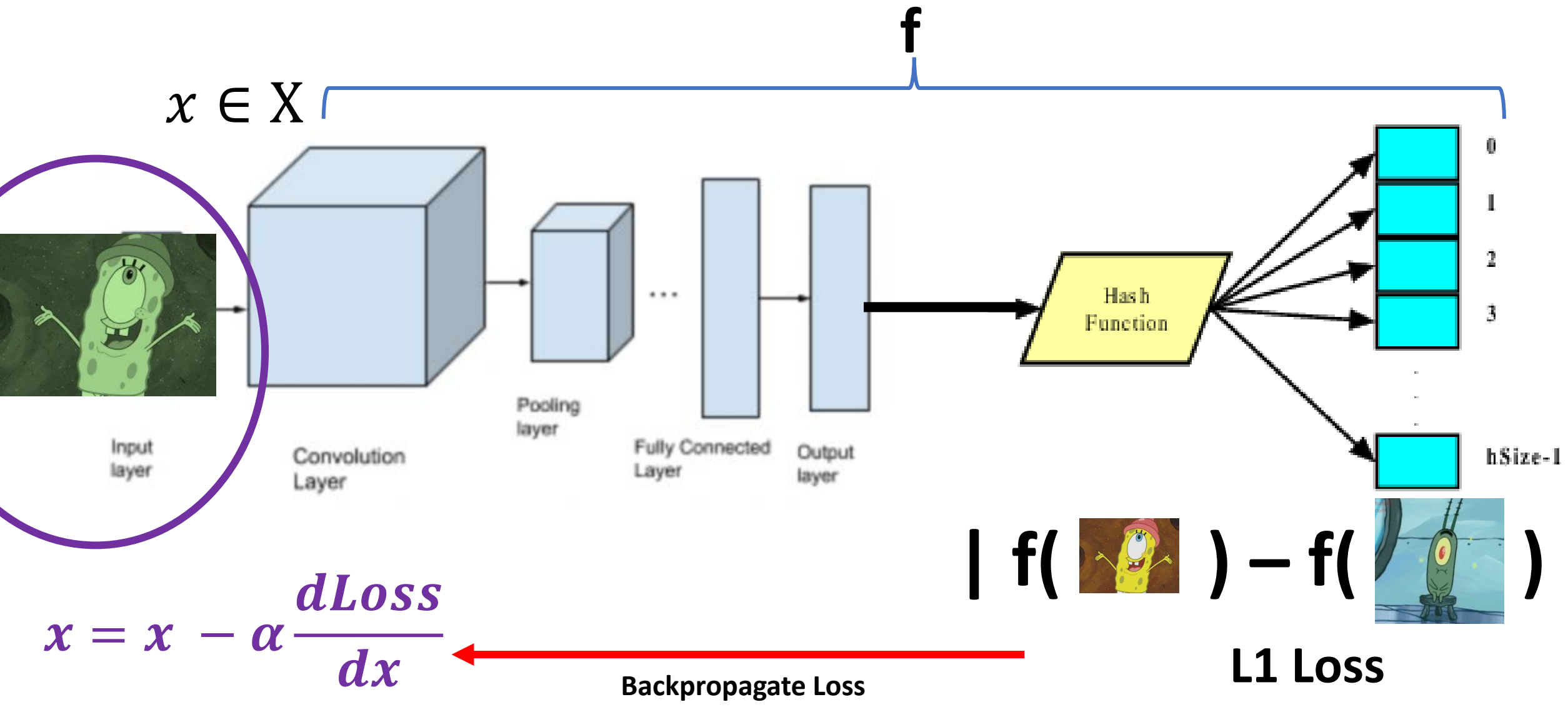
Finding an Adversarial Input (Take 2)



These operations need to be **“differentiable”**!

0110….01 ➔ 63a12effe31910abfb06f301

Finding an Adversarial Input - Final



What Could Pineapple Have Done?

Hide the hashes (maybe)

But where?

Not used **Neural Networks**

Too brittle – you can't make any certain statements trivially!

Interested to hear your thoughts!

Useful Tooling

- https://github.com/DSE-MSU/DeepRobust
  - Adversarial Library for attack and defense methods
- https://www.robust-ml.org/defenses/
  - Published list of defenses and benchmarks
- https://github.com/cleverhans-lab/cleverhans
  - Another library for attack and defense methods
- https://github.com/prolearner/hypertorch
  - Get hypergradient
- https://github.com/facebookresearch/higher
  - Higher order optimization of neural networks

Questions?