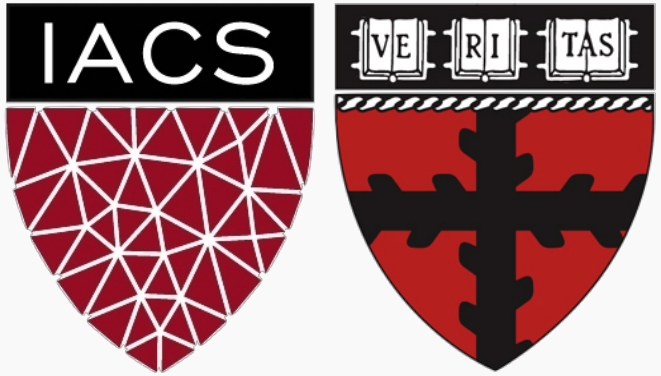
Lecture 6: GANS and Adversarial Networks

Pavlos Protopapas
Institute for Applied Computational Science
Harvard



Outline

Generative Models

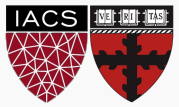
GANS

- Motivation
- Formalism
- Examples
- Training

Game Theory, minmax

Challenges:

- Big Samples
- Modal collapse



Generating Data (is exciting)

Figure 7: Generated samples

https://arxiv.org/pdf/1708.05509.pdf

Generating Data (is exciting)

Ground truth

Generated

Generating Data (is exciting)

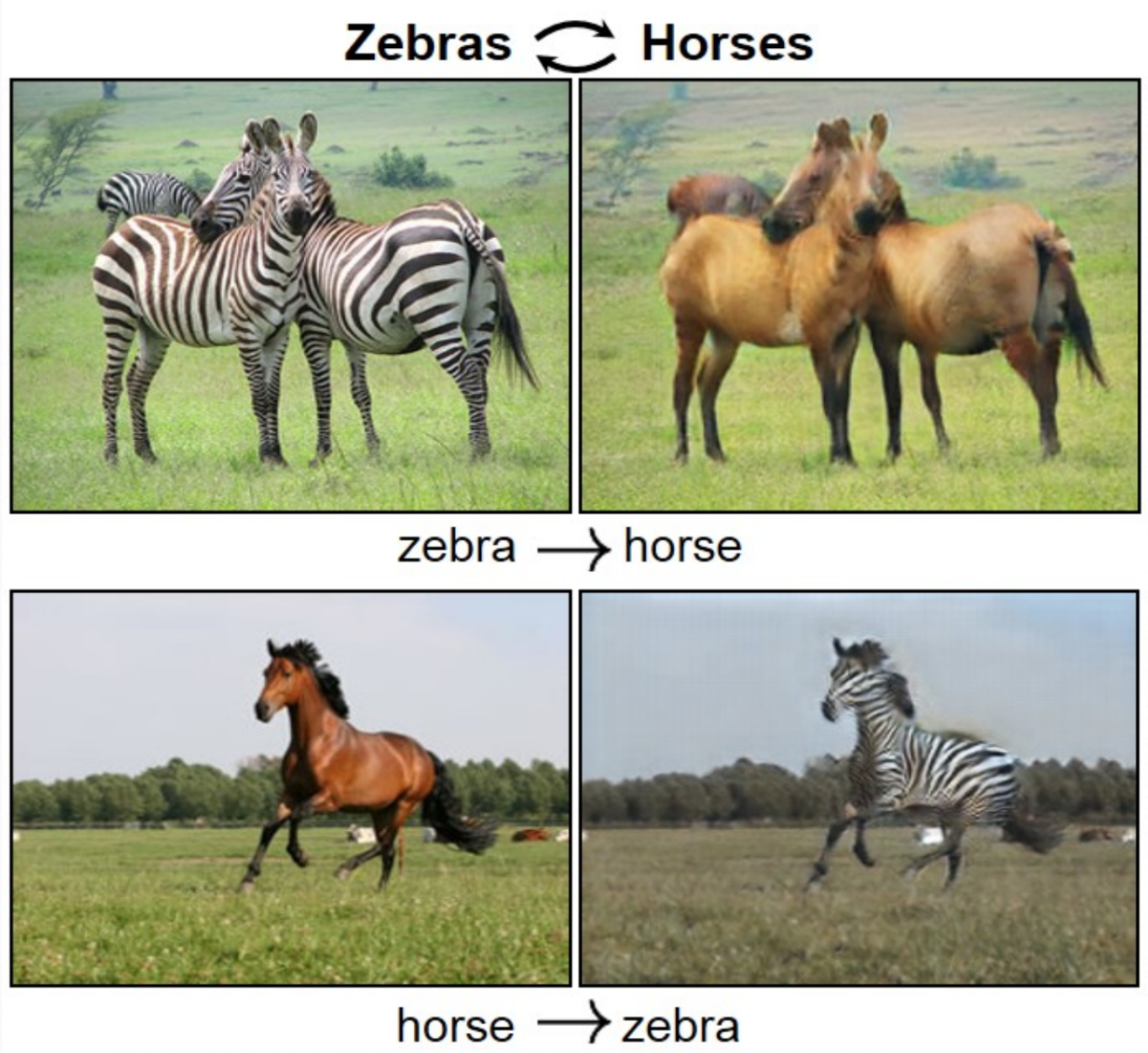
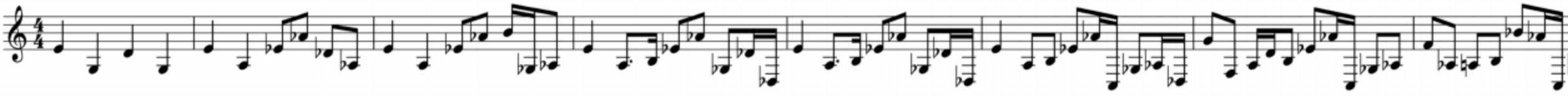
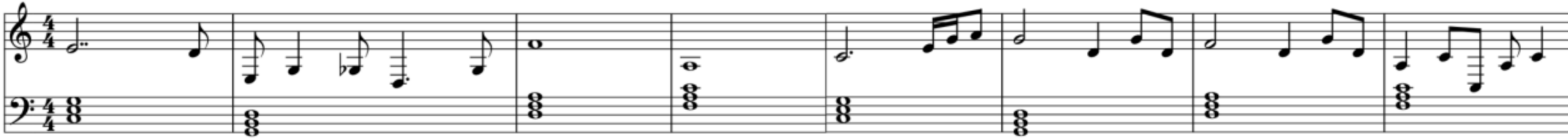


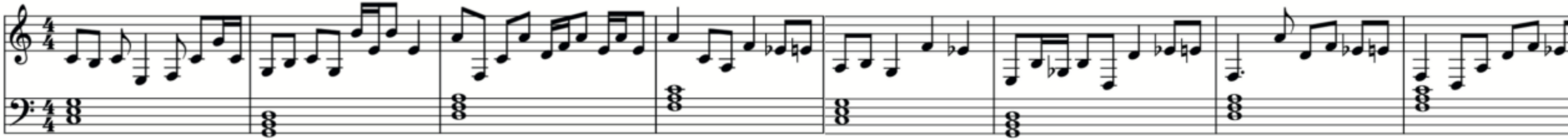
Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.



(a) MidiNet model 1



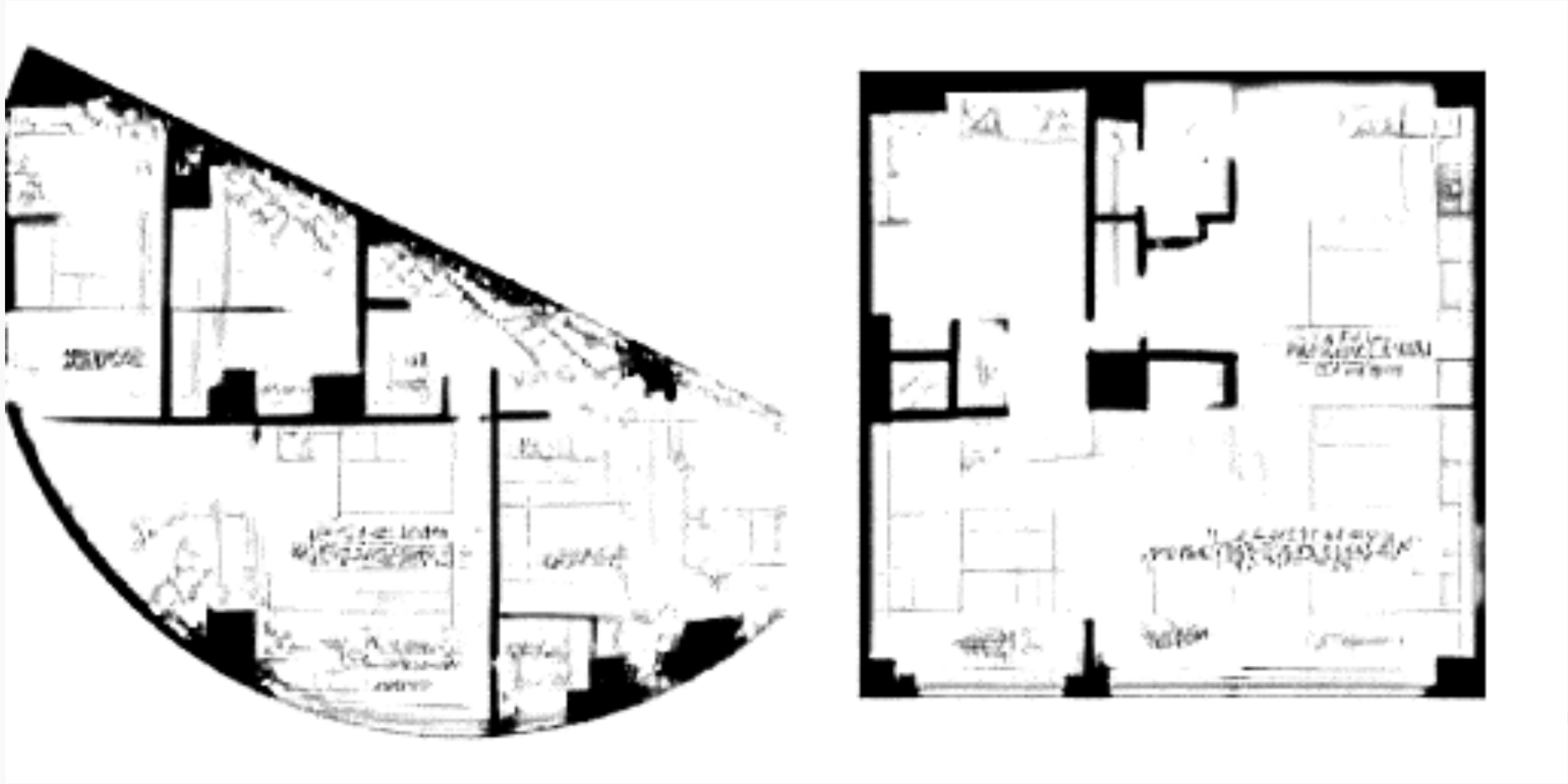
(b) MidiNet model 2



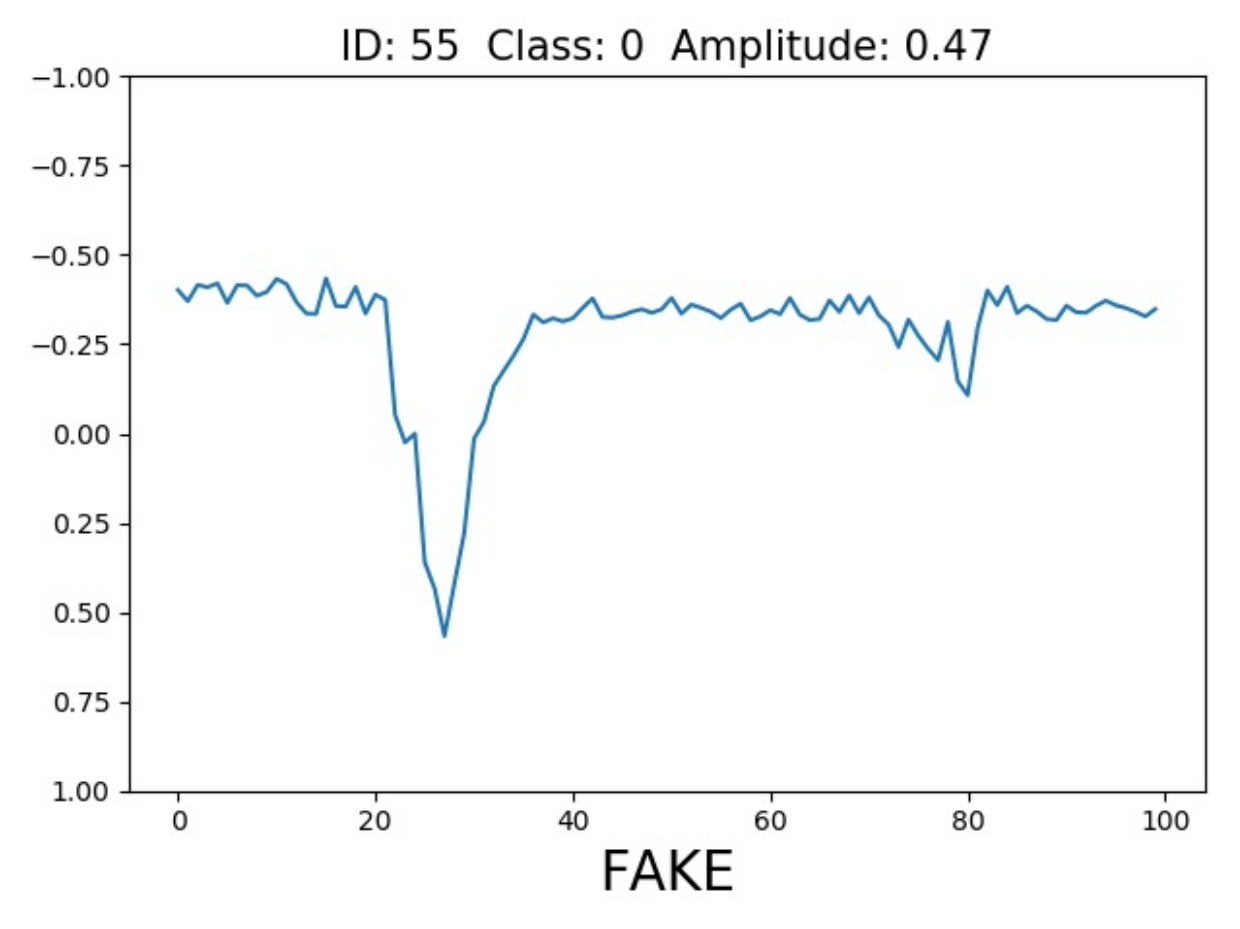
(c) MidiNet model 3

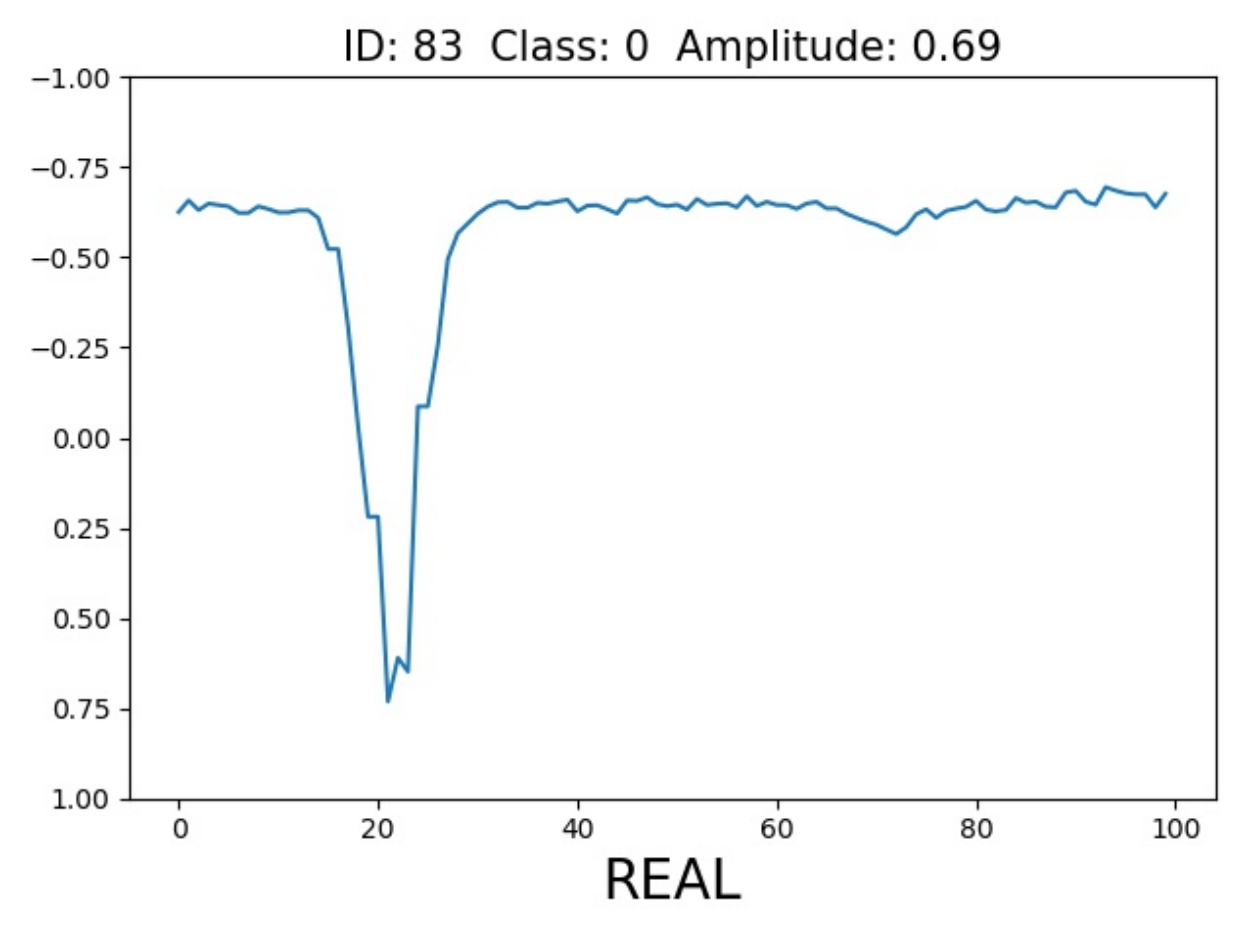
Figure 3. Example result of the melodies (of 8 bars) generated by different implementations of MidiNet.

Another use of generating new data is to give us ideas and options. Suppose we're planning a house. We can give the computer the space we have available, and its location. From this, the computer can can give us some ideas.



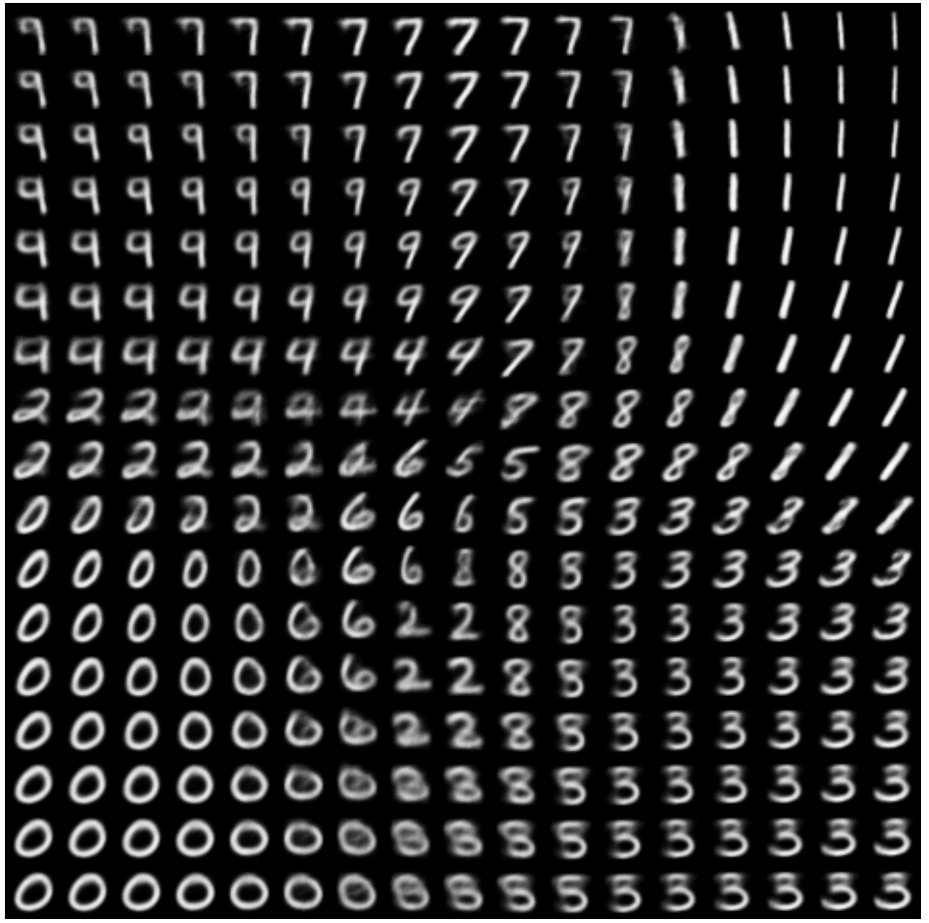
Big networks require big data, and getting high-quality, labeled data is difficult. If we're generating that data our selves, we can make as much of it as we like.





Generating Data

We saw how to generate new data with a VAE in Lecture 5.



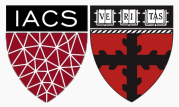
Generating Data

In this chapter we're going to look at a completely different approach to generating data that is like the training data.

This technique lets us generate types of data that go far beyond what a VAE offers.

Generative Adversarial Network, or **GAN**.

It's based on a smart idea where two different networks are put against one another, with the goal of getting one network to create new samples that are different from the training data, but are so close that the other network can't tell which are synthetic and which belong to the original training set.



Generative Adversarial Networks (GANs)

2 Athletes: Messi and Ronaldo. Adversaries!

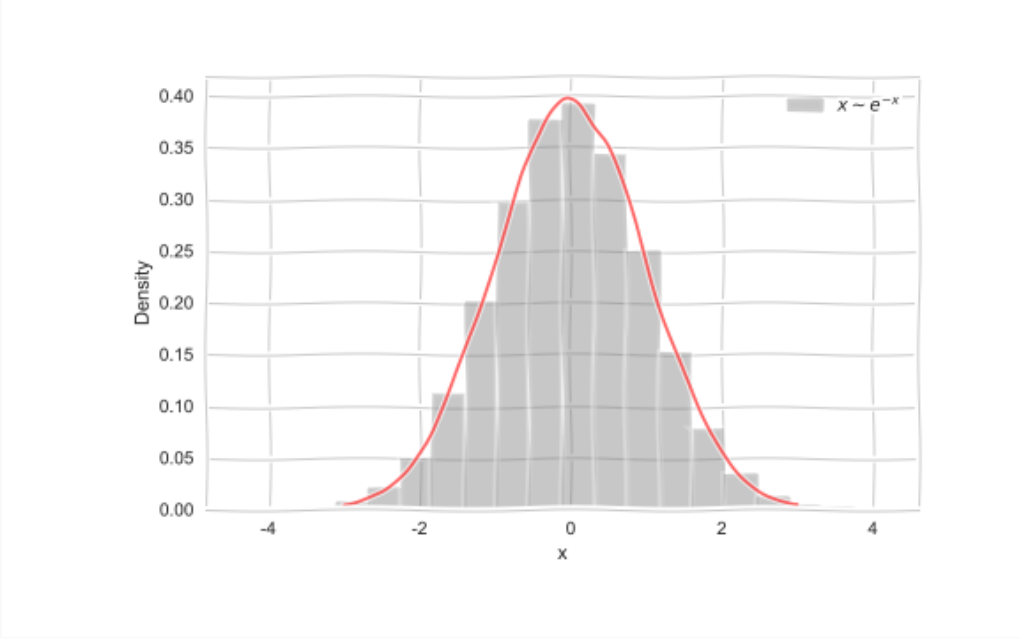
Generative models

Imagine we want to generate data from a distribution,

e.g.

$$x \sim p(x)$$

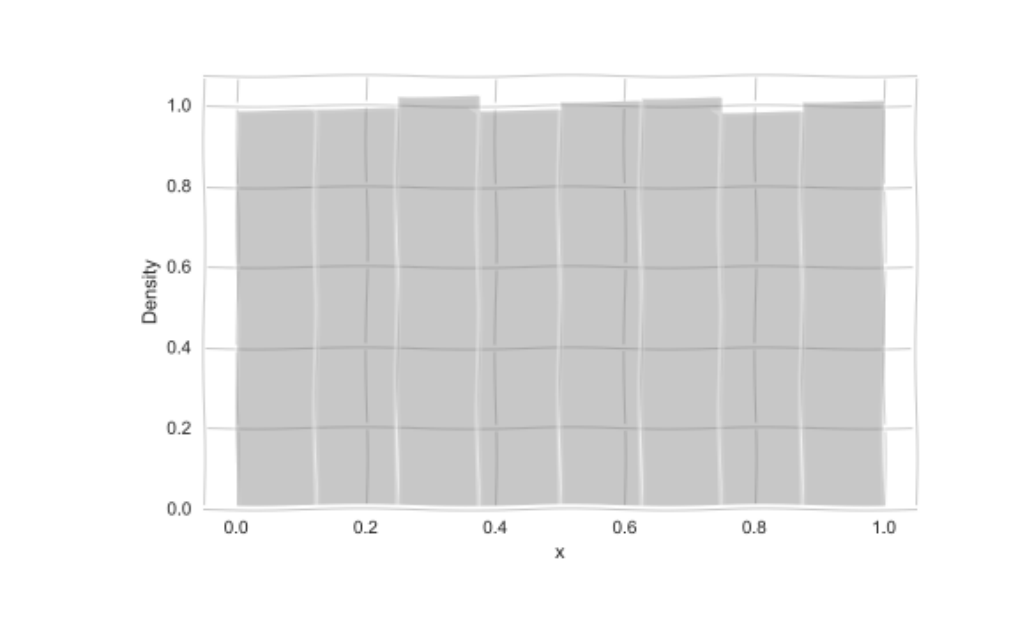
$$x \sim \mathcal{N}(\mu, \sigma)$$



Generative models

But how do we generate such samples?

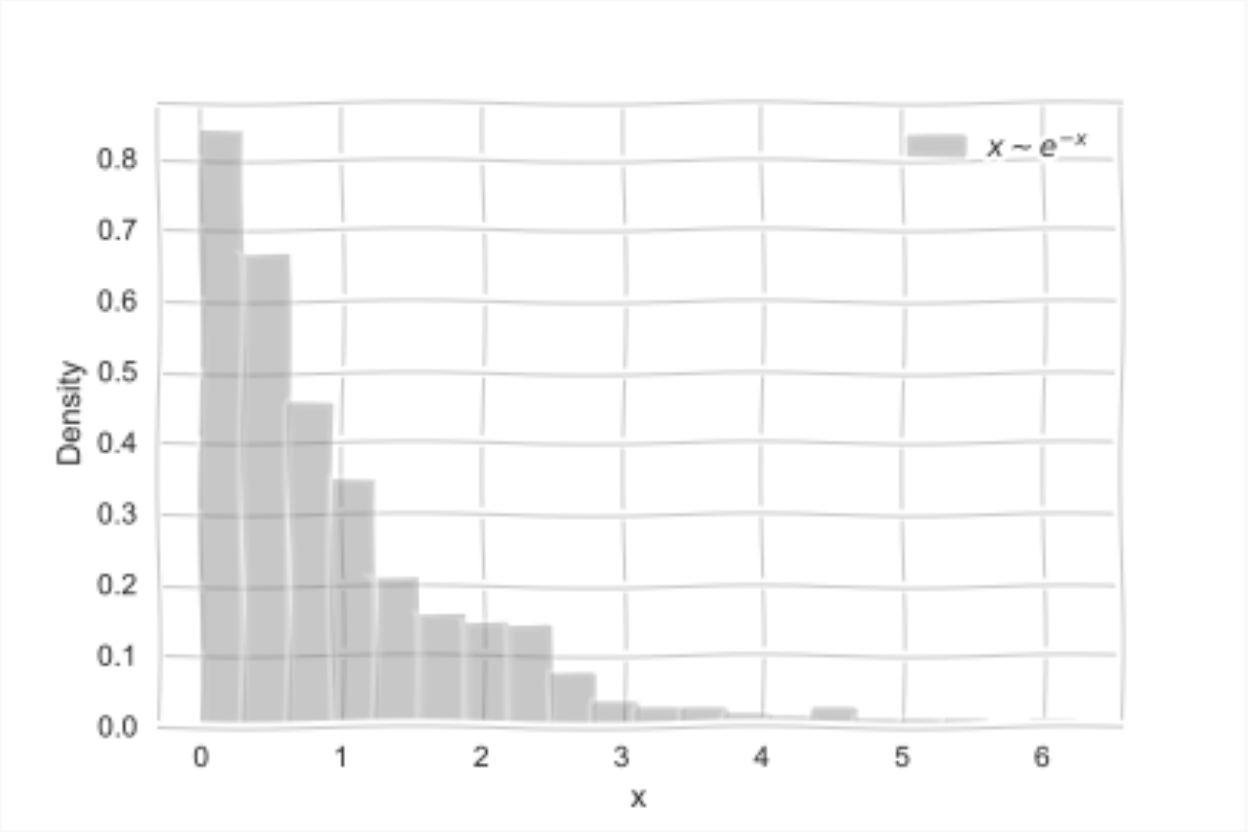
$$z \sim \text{Unif}(0, 1)$$



Generative models

But how do we generate such samples?

$$z \sim \text{Unif}(0,1) \quad \text{x} = \ln \text{z}$$



Generative models

In other words we can think that if we choose $\mathbf{z} \sim \boldsymbol{Uniform}$ then there is a mapping:

$$x = f(z)$$

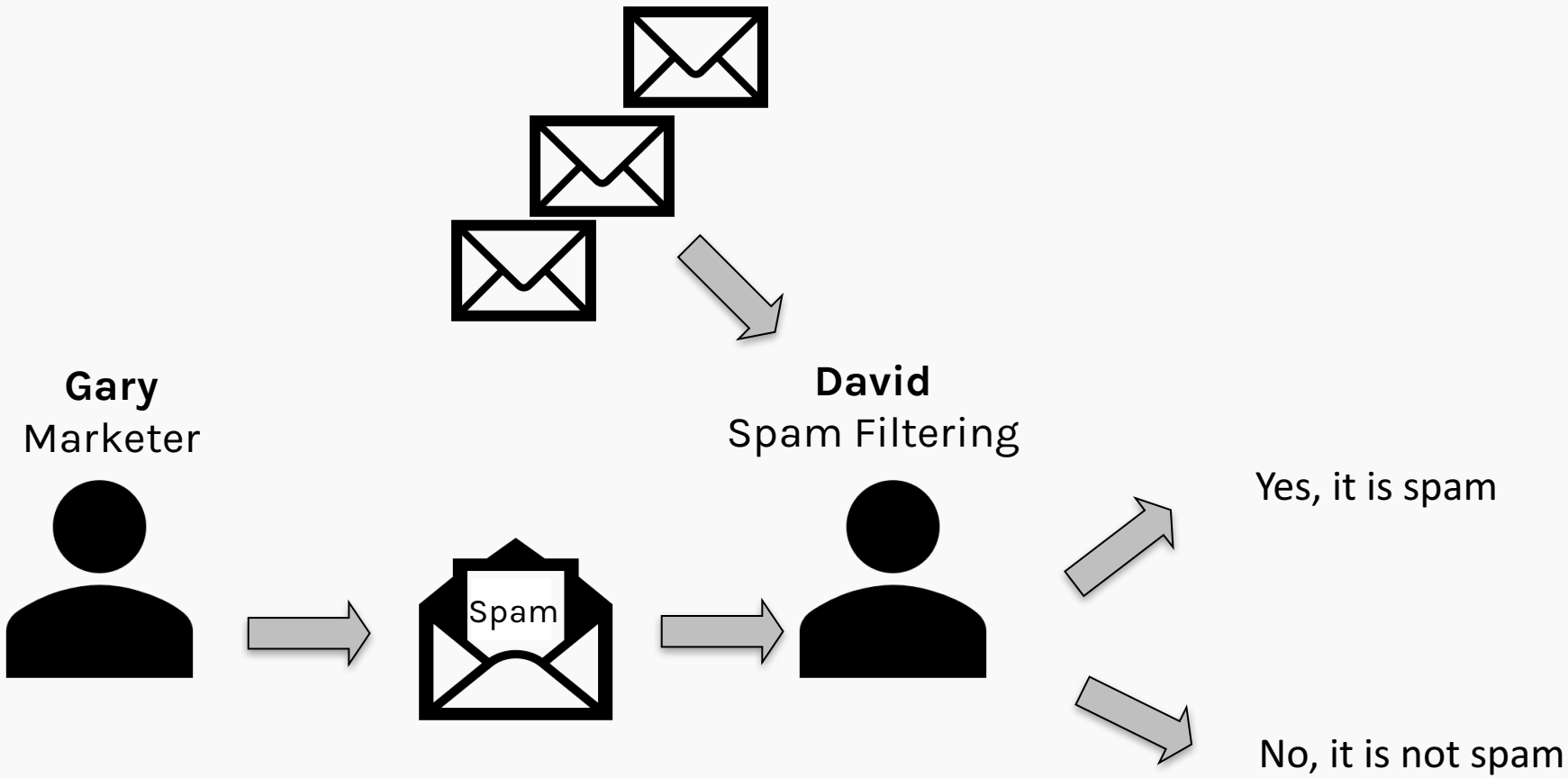
such as:

$$x \sim p(x)$$

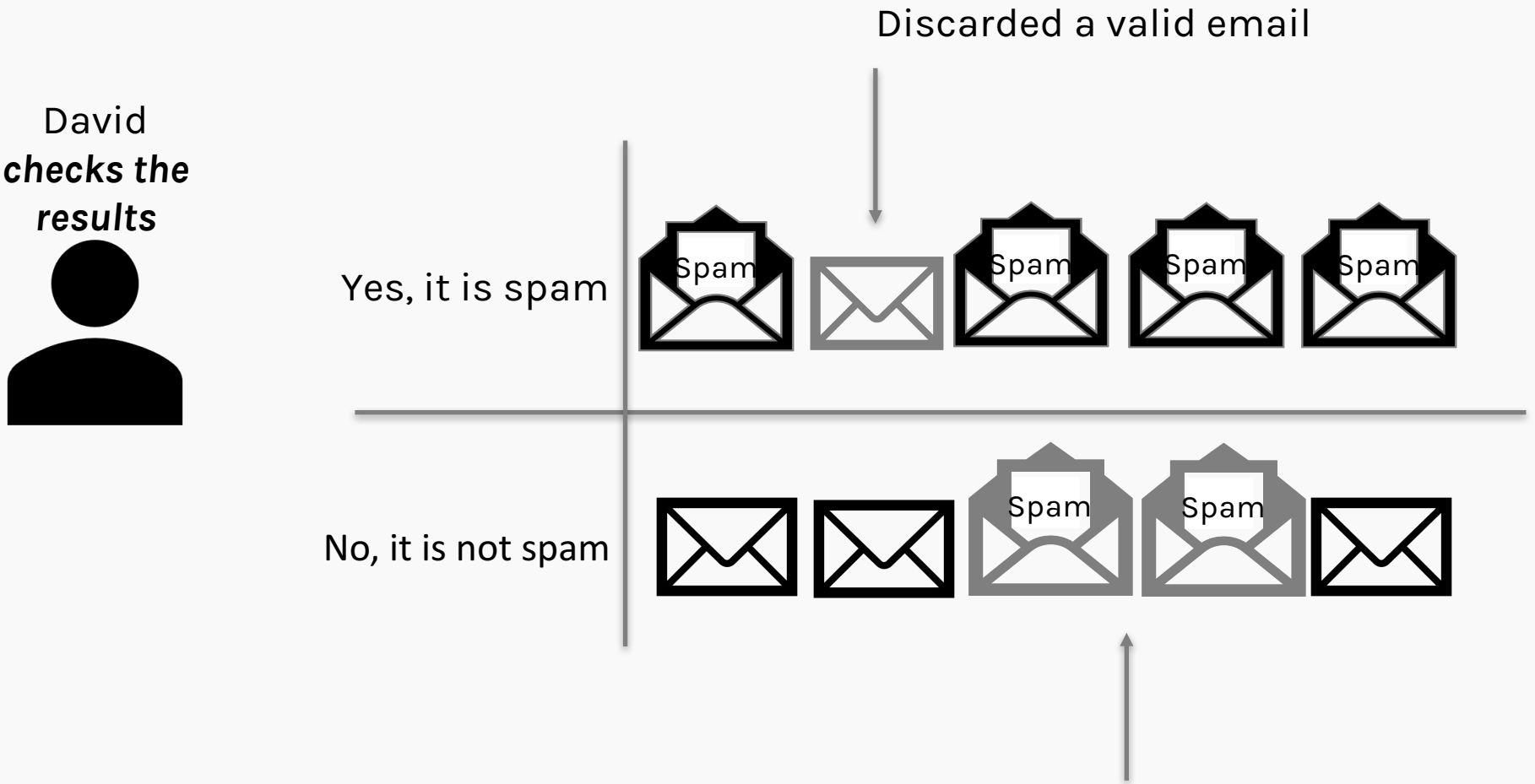
where in general f is some complicated function.

We already know that Neural Networks are great in learning complex functions.

Generative Adversarial Networks (GANs)

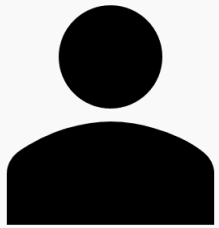


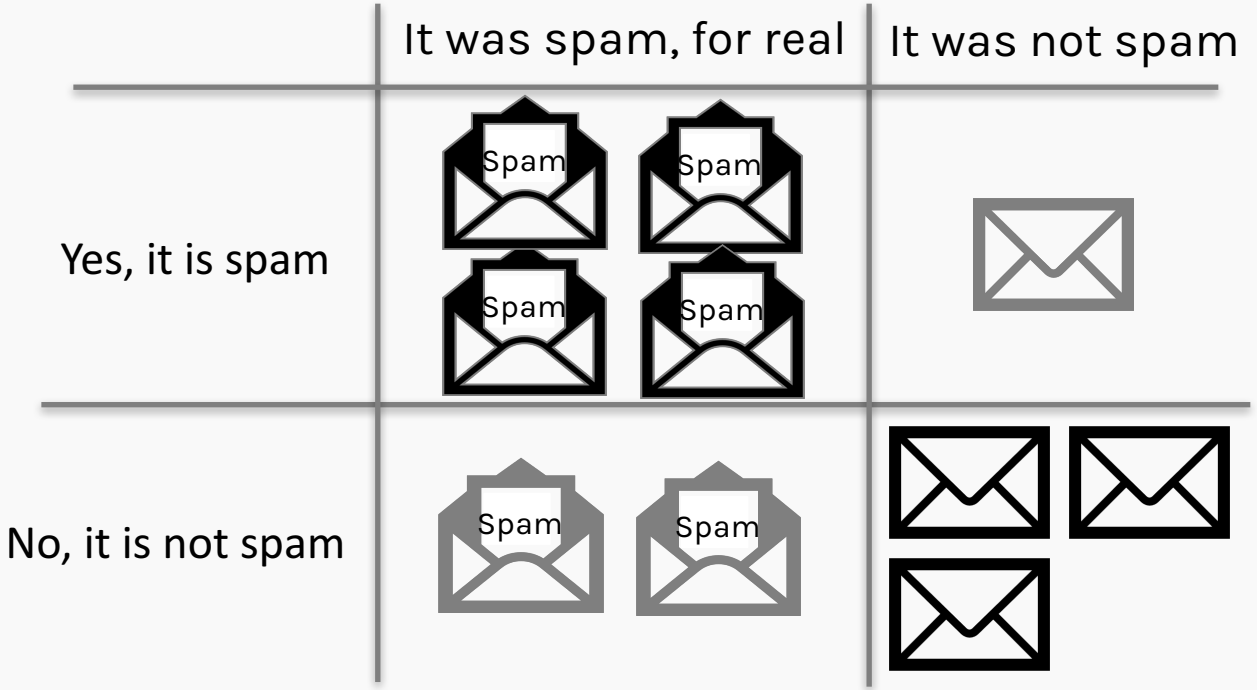
Generative Adversarial Networks (GANs)



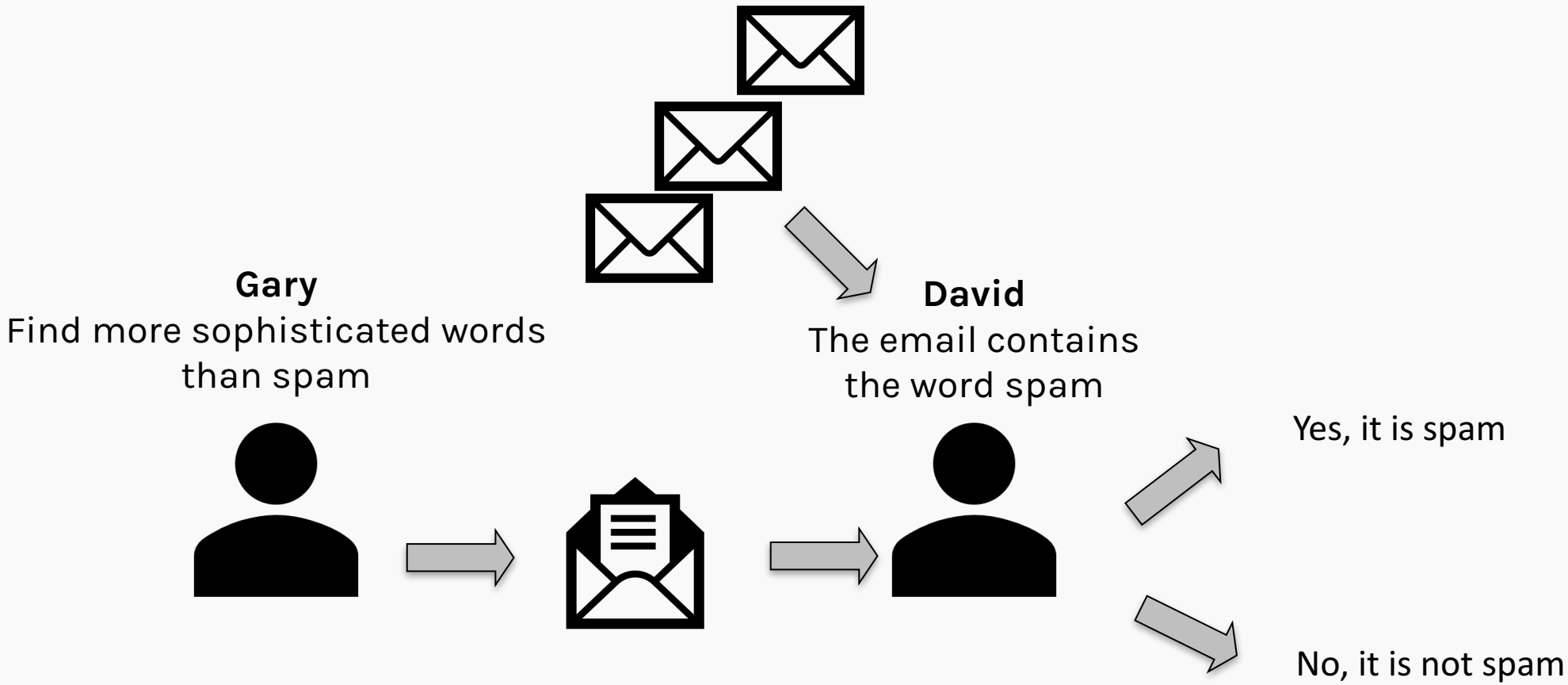
Generative Adversarial Networks (GANs)

David
learned what went wrong

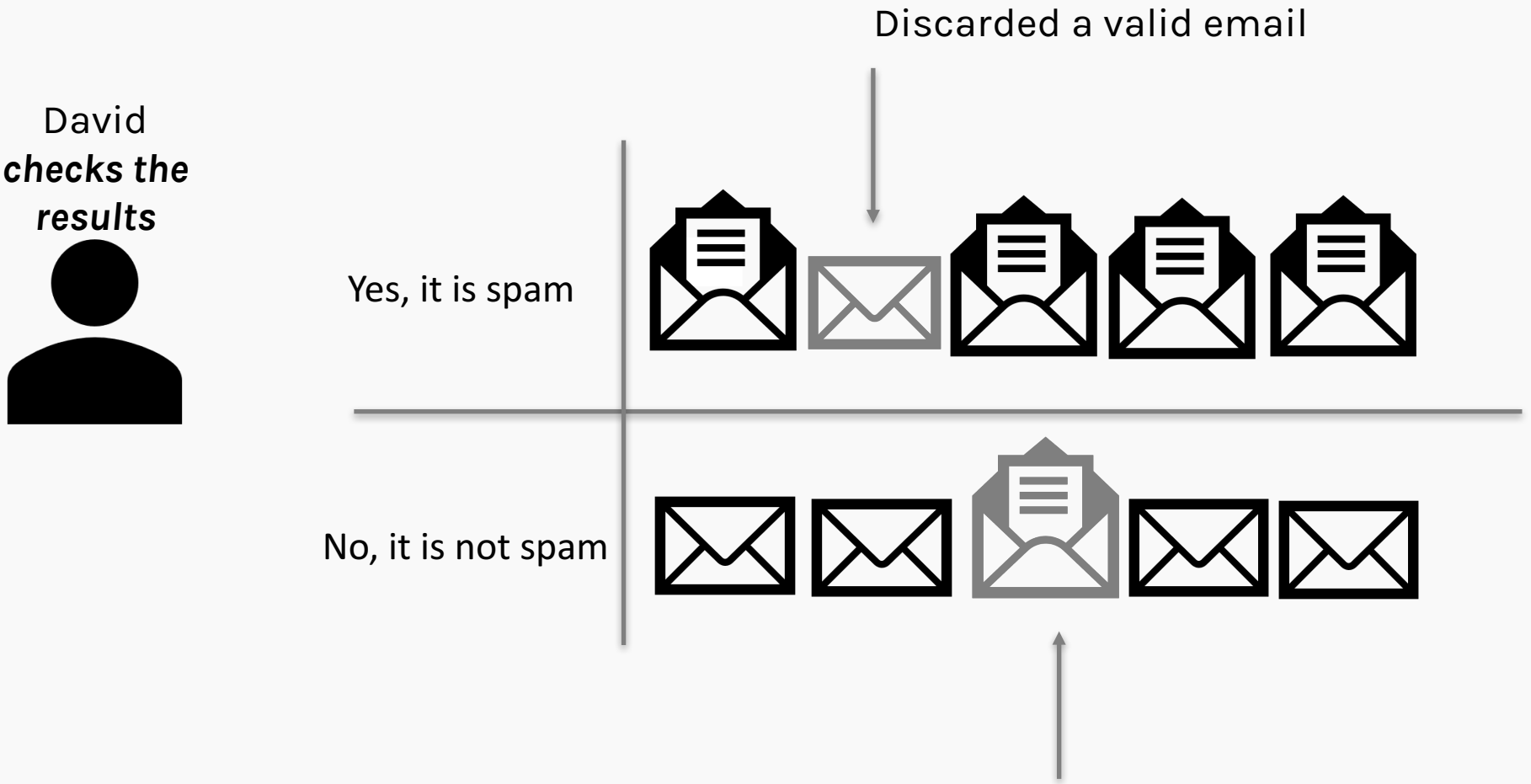




Generative Adversarial Networks (GANs)

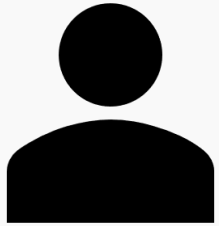


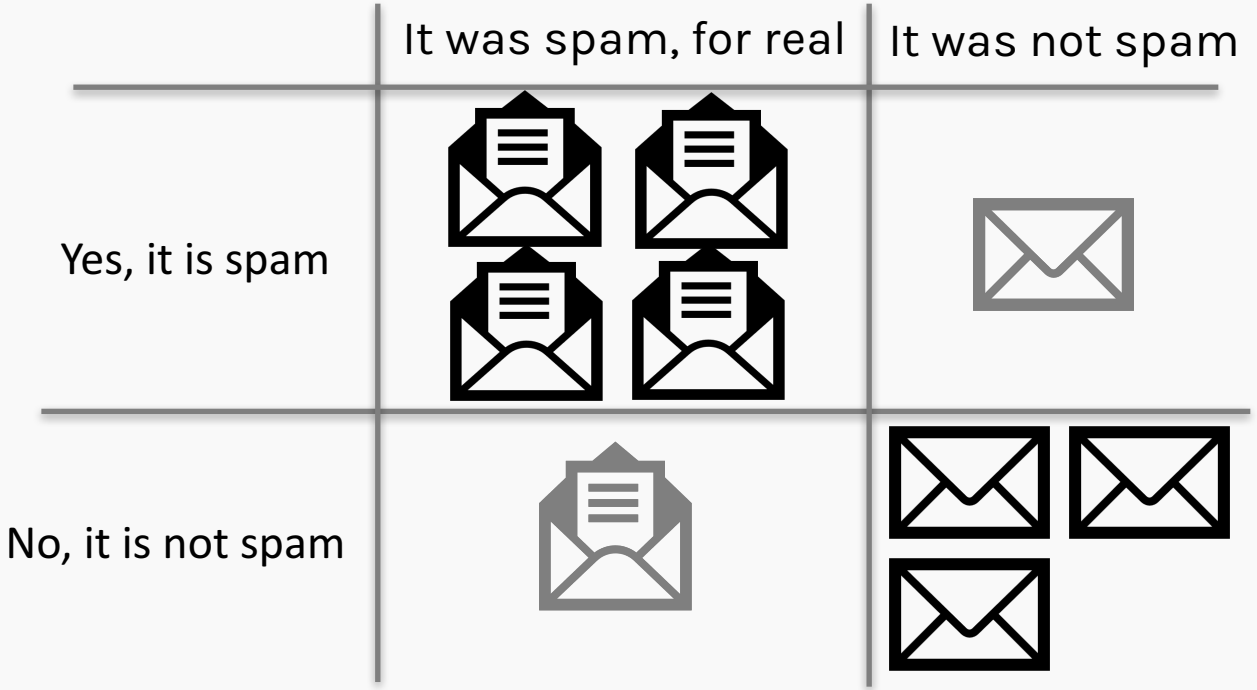
Generative Adversarial Networks (GANs)



Generative Adversarial Networks (GANs)

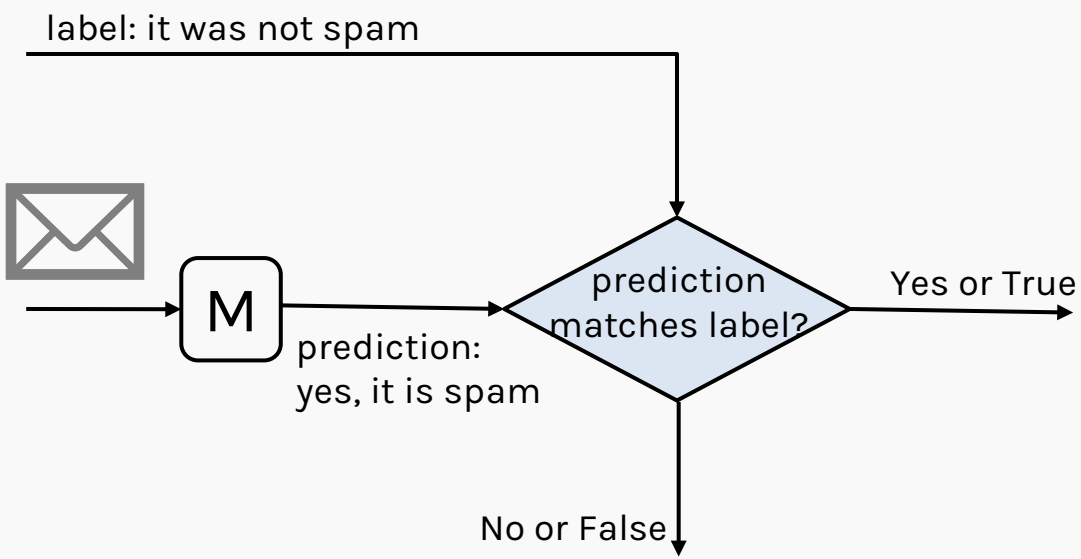
David
learned what went wrong

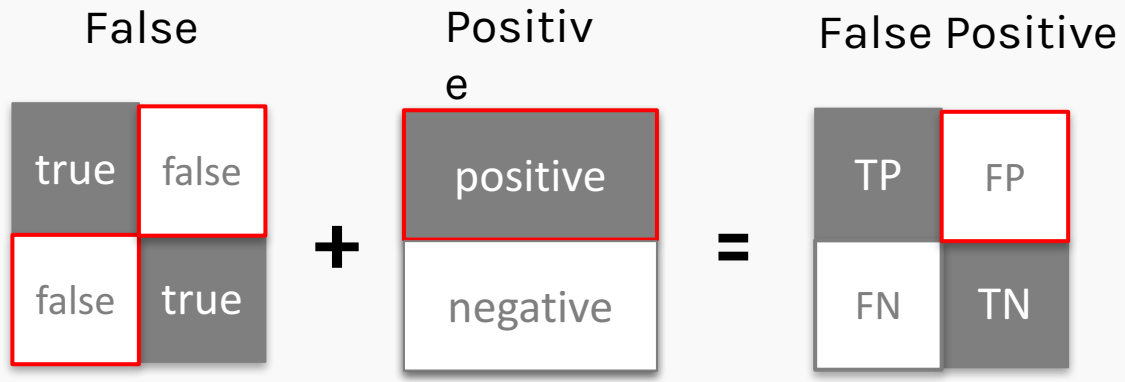




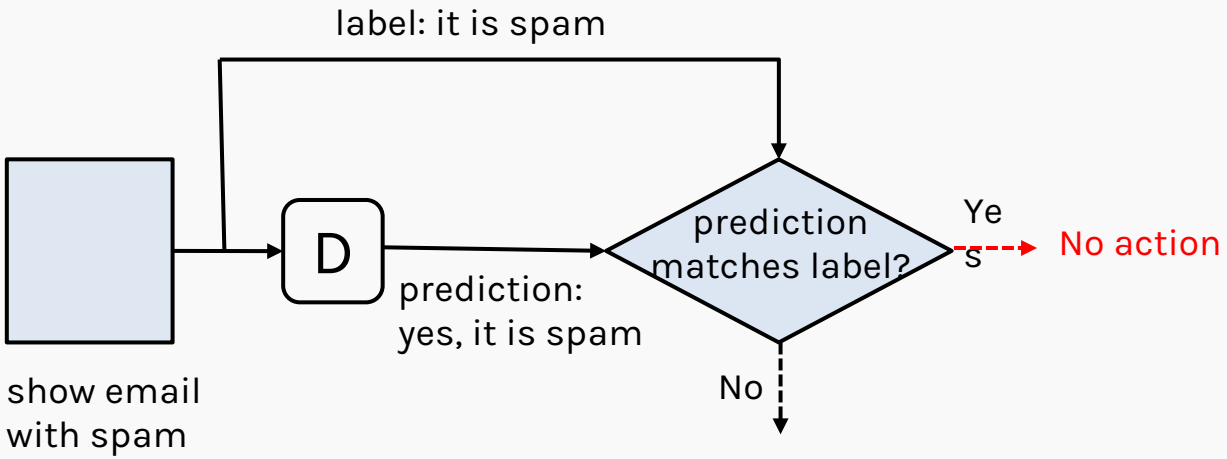
Generative Adversarial Networks (GANs)

Understand confusion matrix through an example

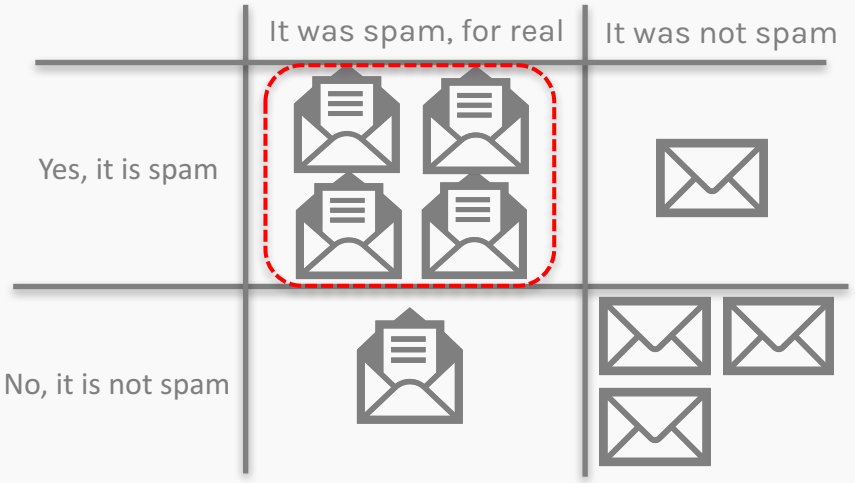




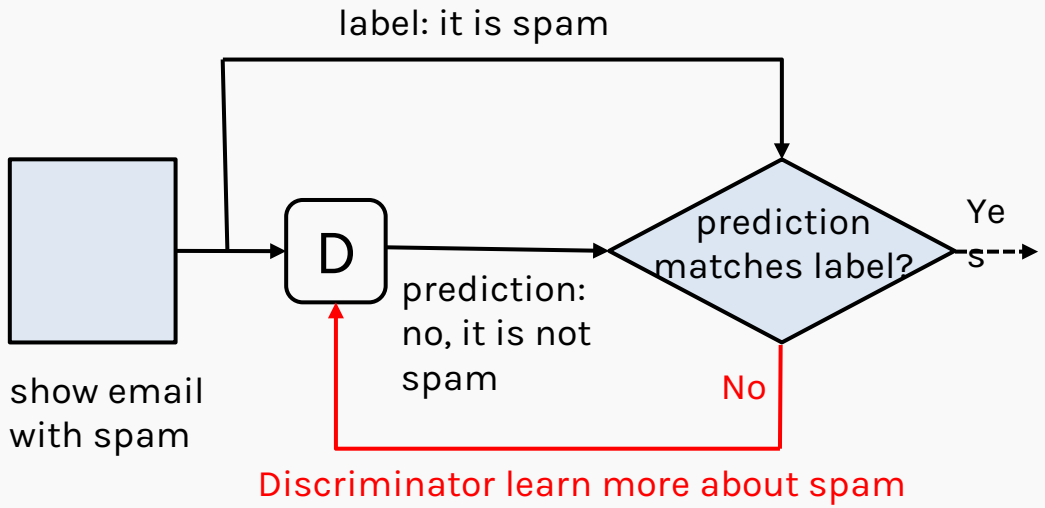
Generative Adversarial Networks (GAN)



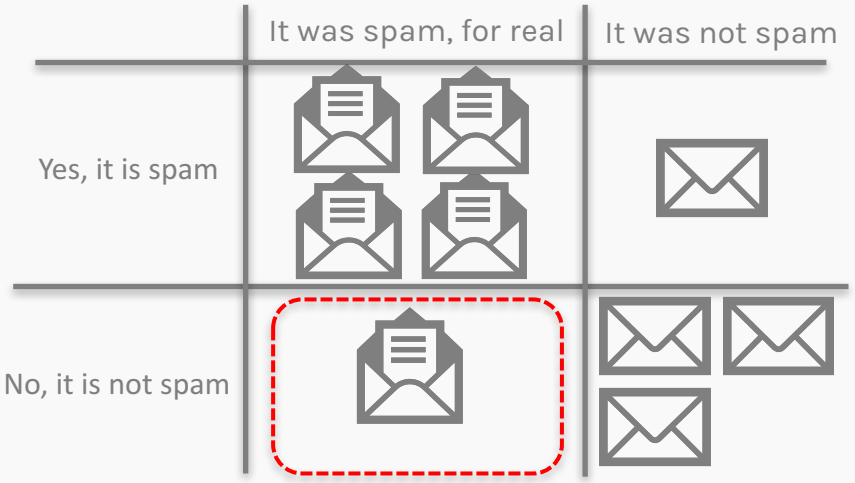
True positive (TP): the discriminator see a spam and predicts correctly. No need for further actions.



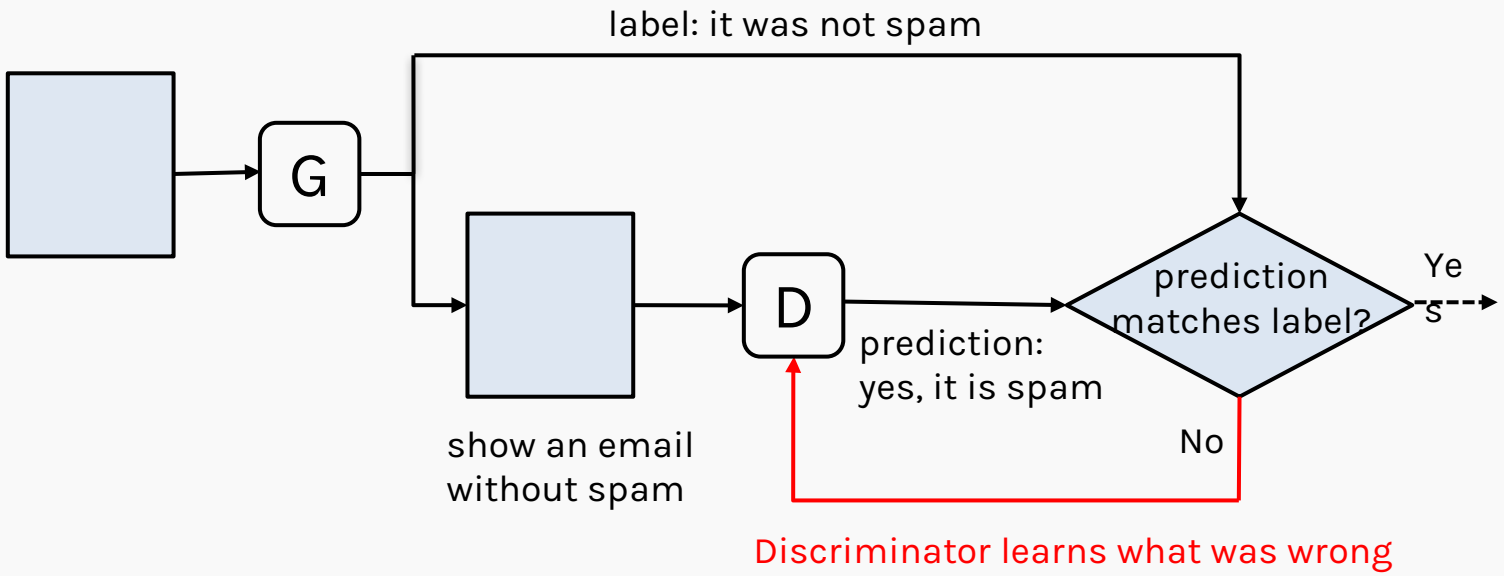
Generative Adversarial Networks (GANs)

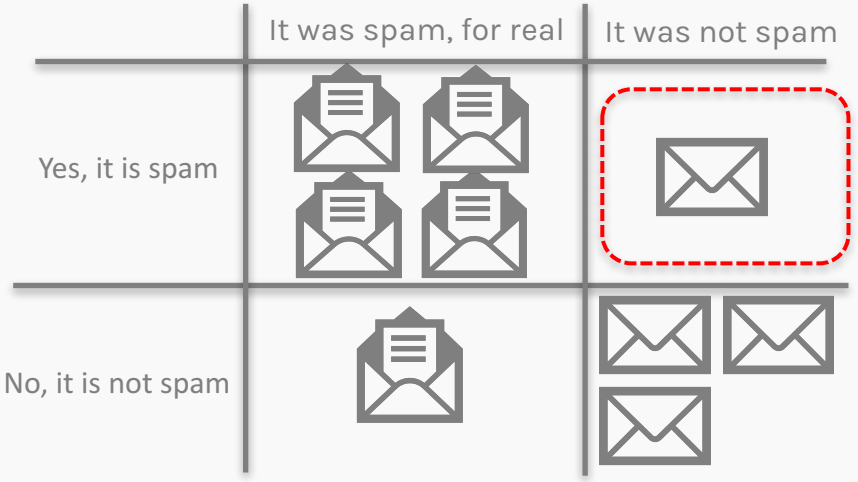


False Negative (FN): the discriminator see an email and predict it as spam even though it is not. The discriminator learn more



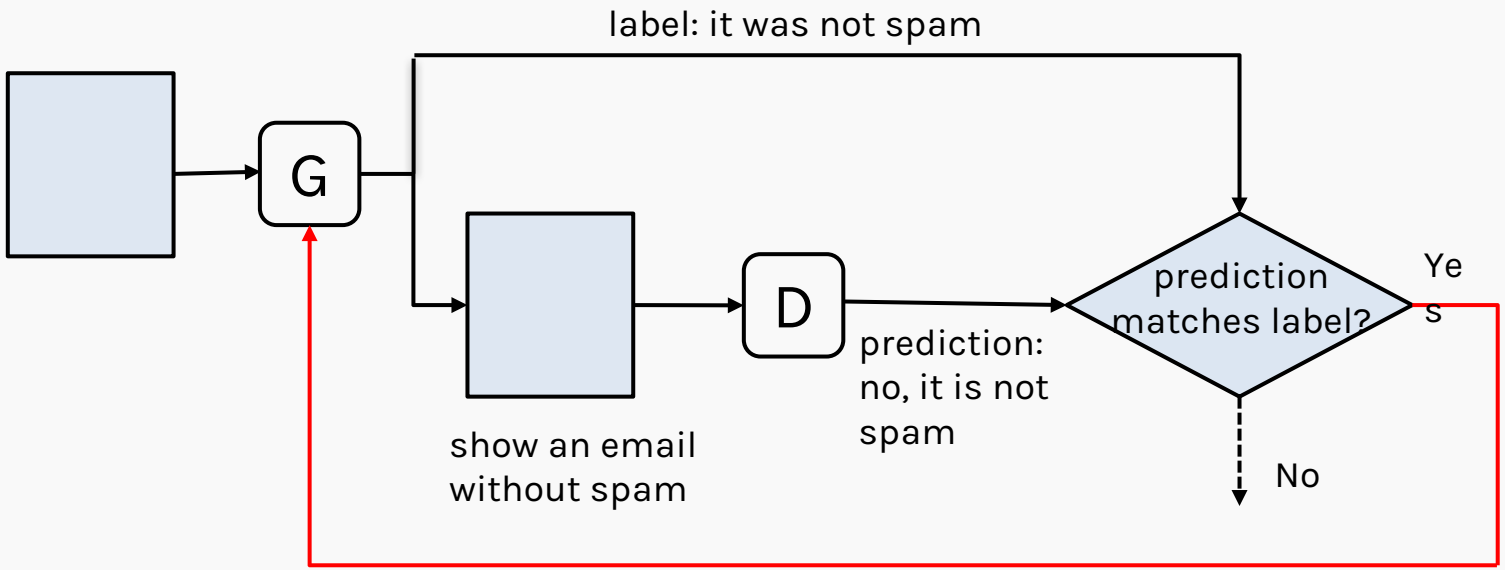
Generative Adversarial Networks (GANs)

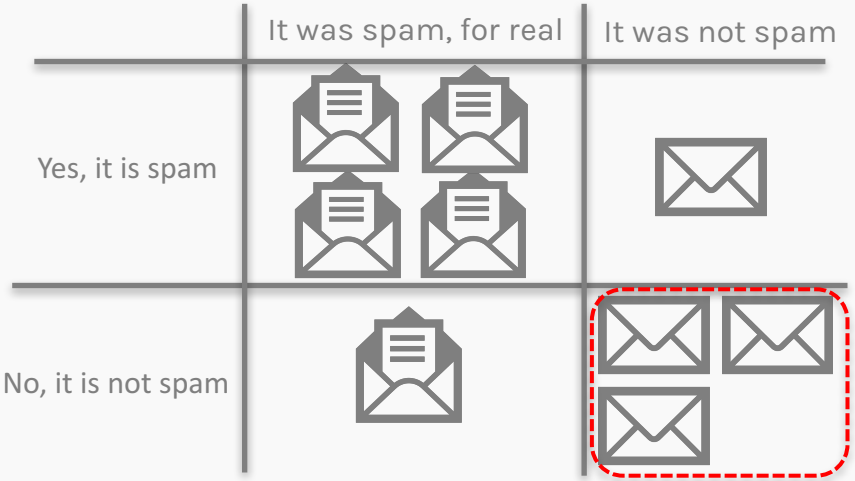




False positive (FP): generator try to fool discriminator and the discriminator fails. The generator succeeded and the generator is forced to improve.

Generative Adversarial Networks (GANs)





True negative (TN): generator try to fool discriminator, however the discriminator predict correctly. The generator learn what was wrong and try something else.

Generative Adversarial Networks (GANs)

The process – known as **Learning Round** - accomplishes three jobs:

1. The discriminator learns to identify features that characterize a real sample
2. The discriminator learns to identify features that reveal a fake sample
3. The generator learns how to avoid including the features that the discriminator has learned to spot

Min-max Cost

Discriminator seeks to predict 1 on training samples

Discriminator wants to predict 0 on samples generated by *G*

$$V(\theta^D, \theta^G) = \mathbf{E}_{x \sim p_{data}} \log D(x) + \mathbf{E}_z \log(1 - D(G(z)))$$

Generator wants *D* to not distinguish between original and generated samples!

Min-max Cost

$$\min_{\theta^G} \max_{\theta^D} J(\theta^G, \theta^D)$$

Equilibrium is a saddle-point

Training is difficult in practice

Training GANs

Sample mini-batch of training images x, and generator codes z

Update G using back-prop

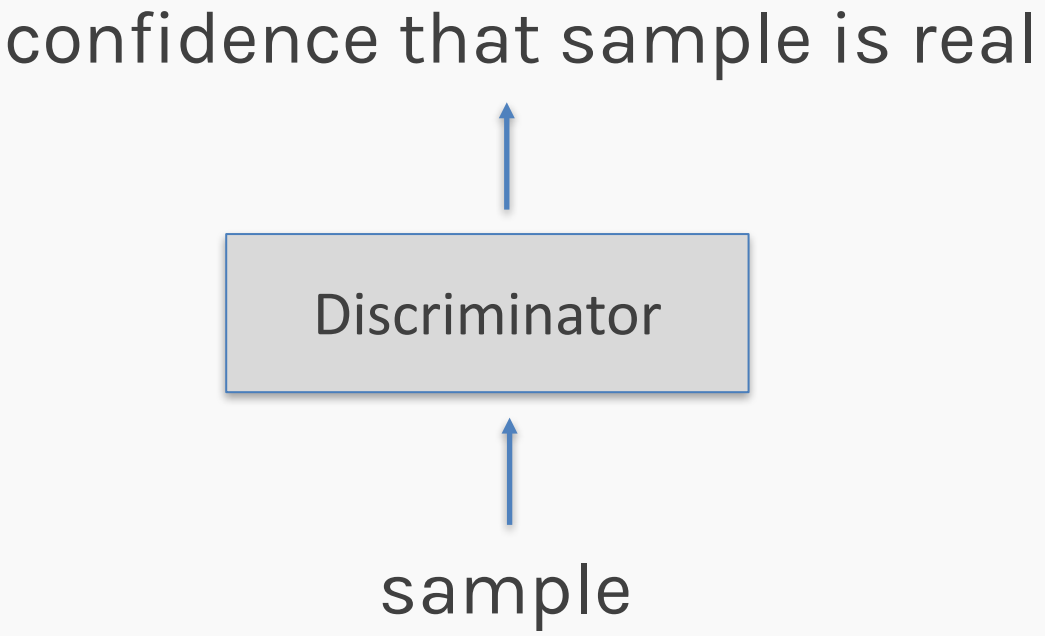
Update D using back-prop

Optional: Run k steps of one player for every step of the other player

The Discriminator

The discriminator is very simple. It takes a sample as input, and its output is a single value that reports the network's confidence that the input is from the training set, rather than being a fake.

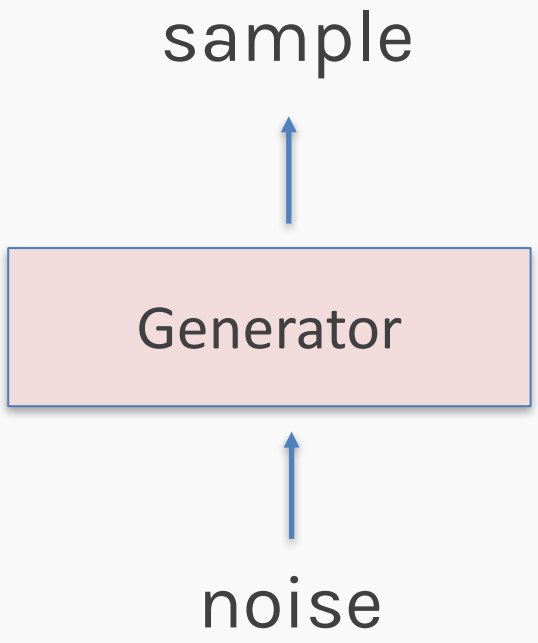
There are not many restrictions on what the discriminator is.



The Generator

The generator takes as input a bunch of **random numbers** .

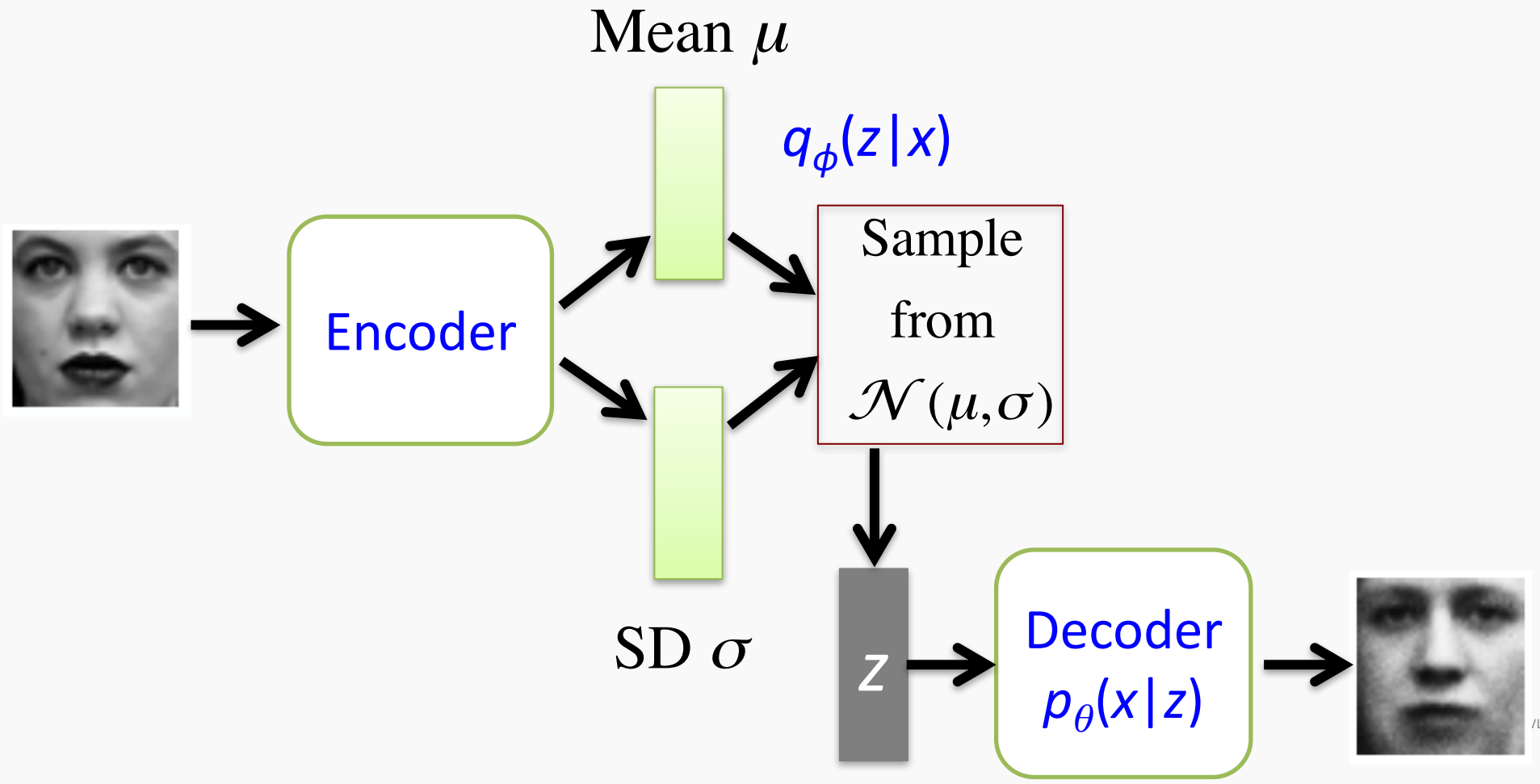
If we build our generator to be deterministic, then the same input will always produce the same output. In that sense we can think of the input values as latent variables. But here the latent variables weren't discovered by analyzing the input, as they were for the VAE.



The Generator

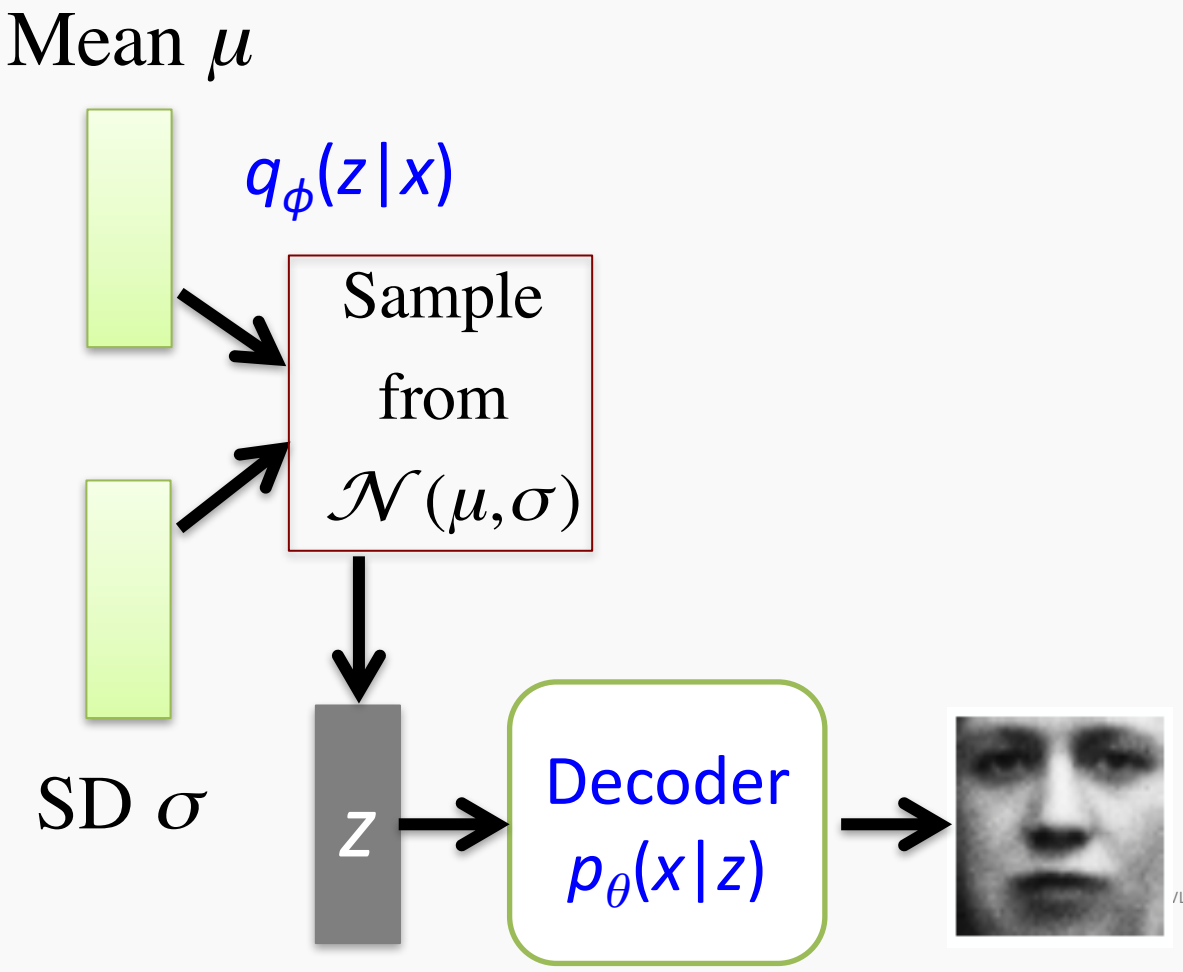
Is it really noise?

Remember our VAE



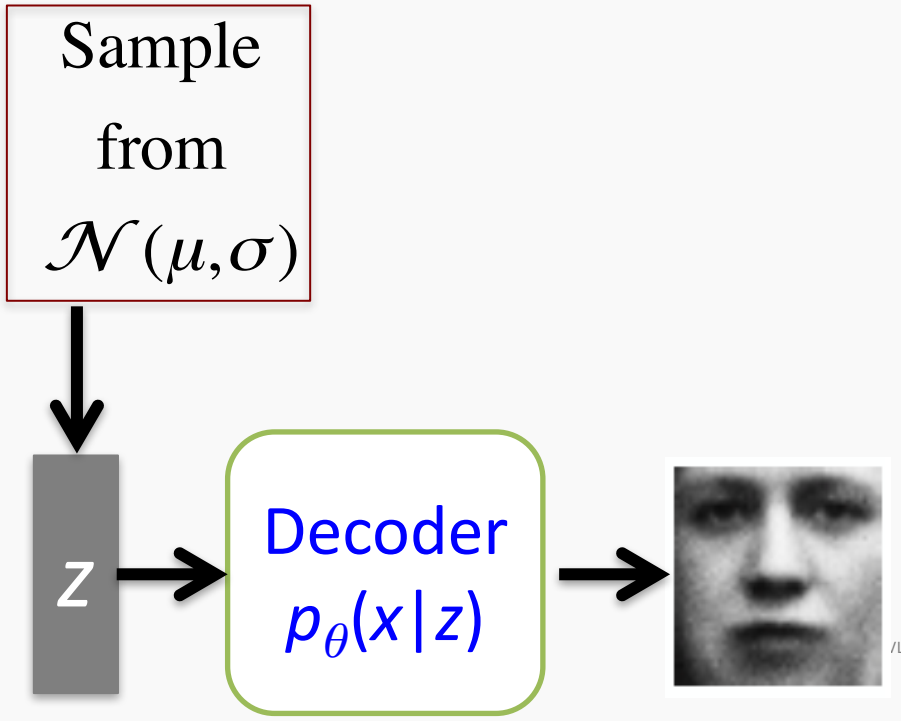
The Generator

Is it really noise? Remember our Variational Autoencoder.



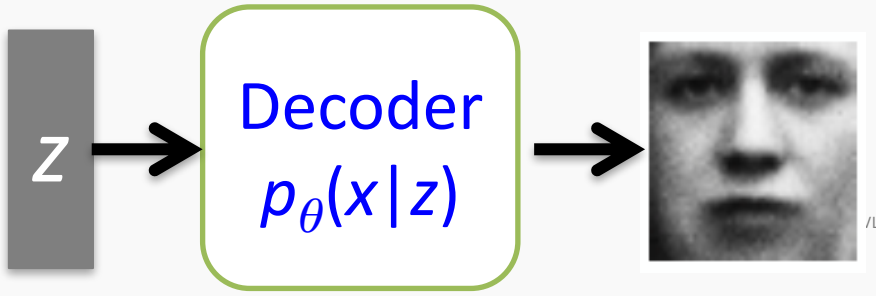
The Generator

Is it really noise? Remember our Variational Autoencoder.



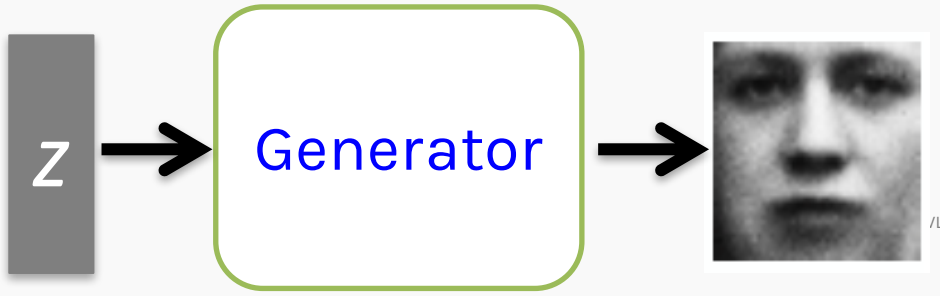
The Generator

Is it really noise? Remember our Variational Autoencoder.

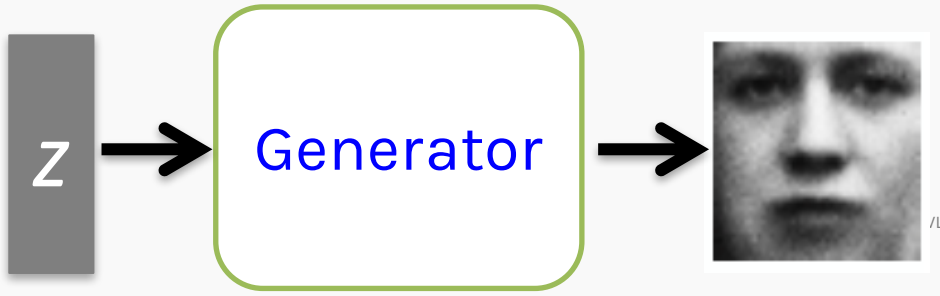


The Generator

Is it really noise? Remember our Variational Autoencoder.

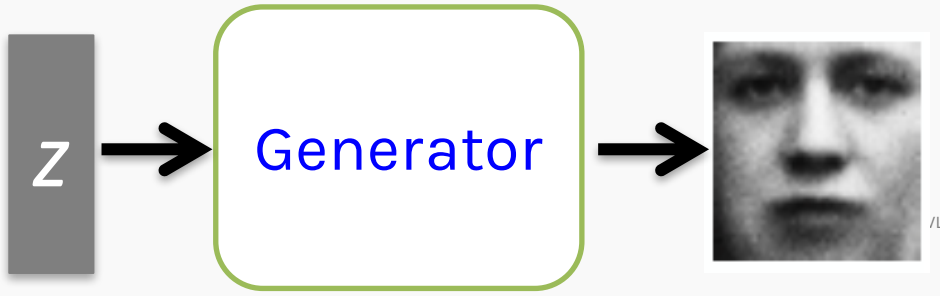


The Generator

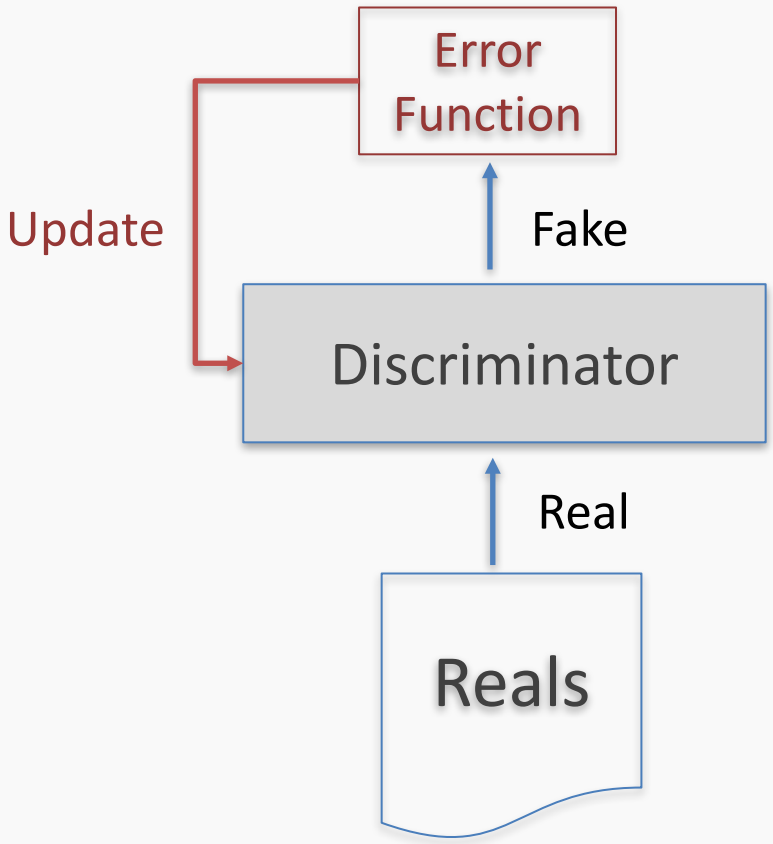


The Generator

So the random noise is not “random” but represents (an image in the example below) in the “latent” space.



Training the GAN

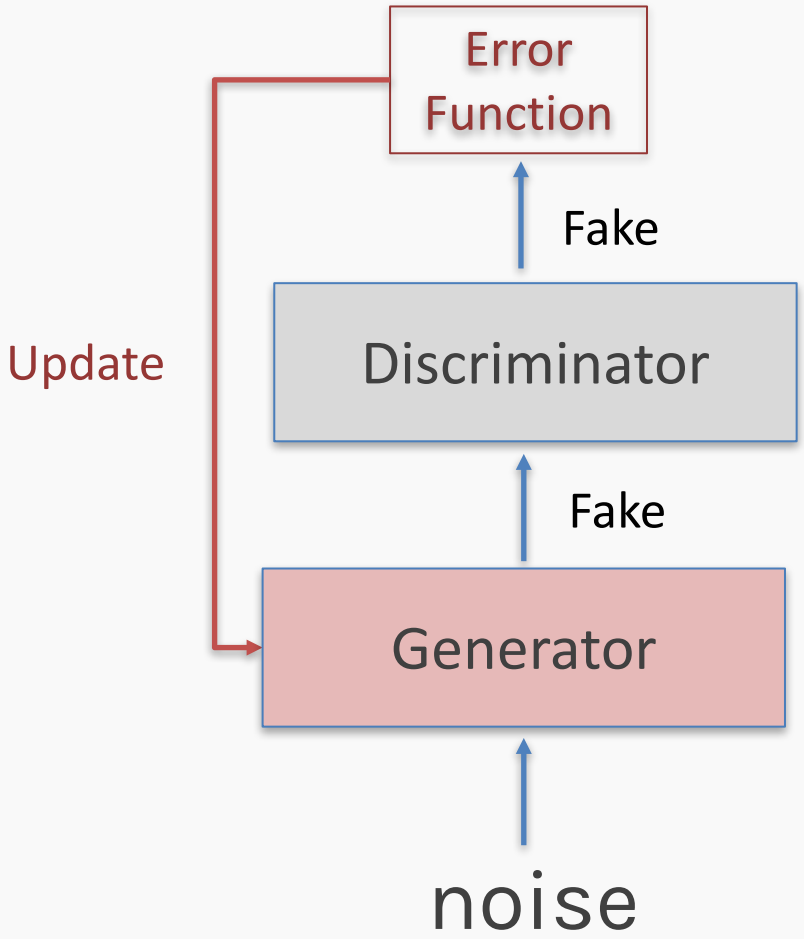


False negative (I: Real/D: Fake):

In this case we feed reals to the discriminator. The Generator is not involved in this step at all.

The error function here only involves the Discriminator and if it makes a mistake the error drives a backpropagation step through the discriminator, updating its weights, so that it will get better at recognizing reals.

Training the GAN



True negative (I: Fake/D: Fake):

We start with random numbers going into the generator.

The generator's output is a fake.

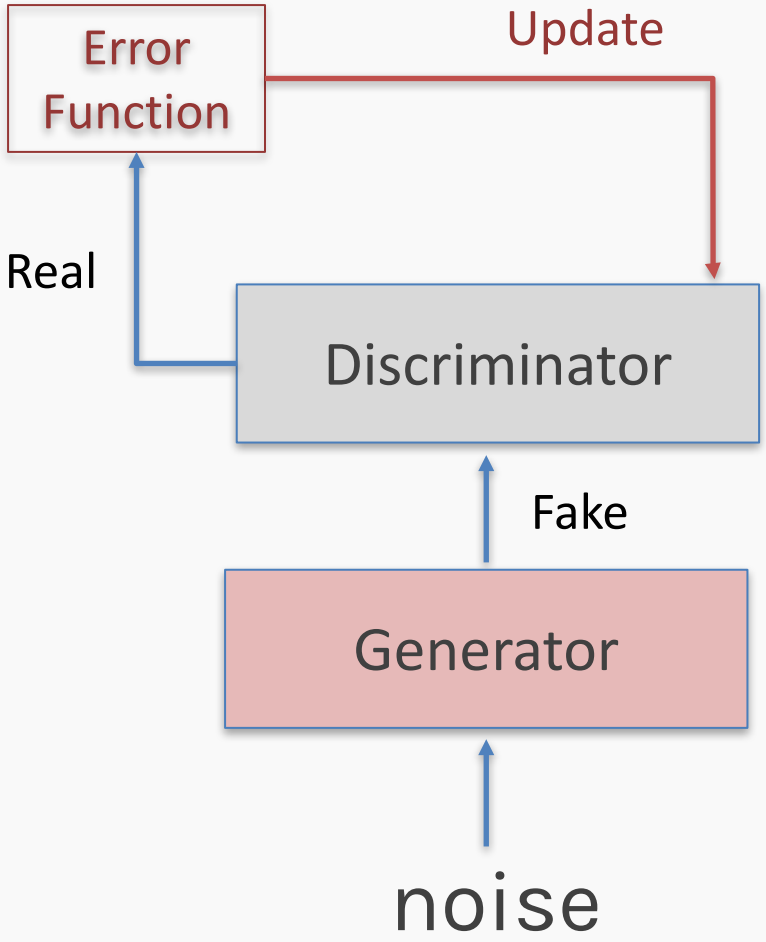
The error function gets a large value if this fake is correctly identified as fake.

Meaning that the generator got caught.

Backprop, goes through the discriminator (which frozen) to the generator.

Update the generator, so it can better learn how to fool the discriminator.

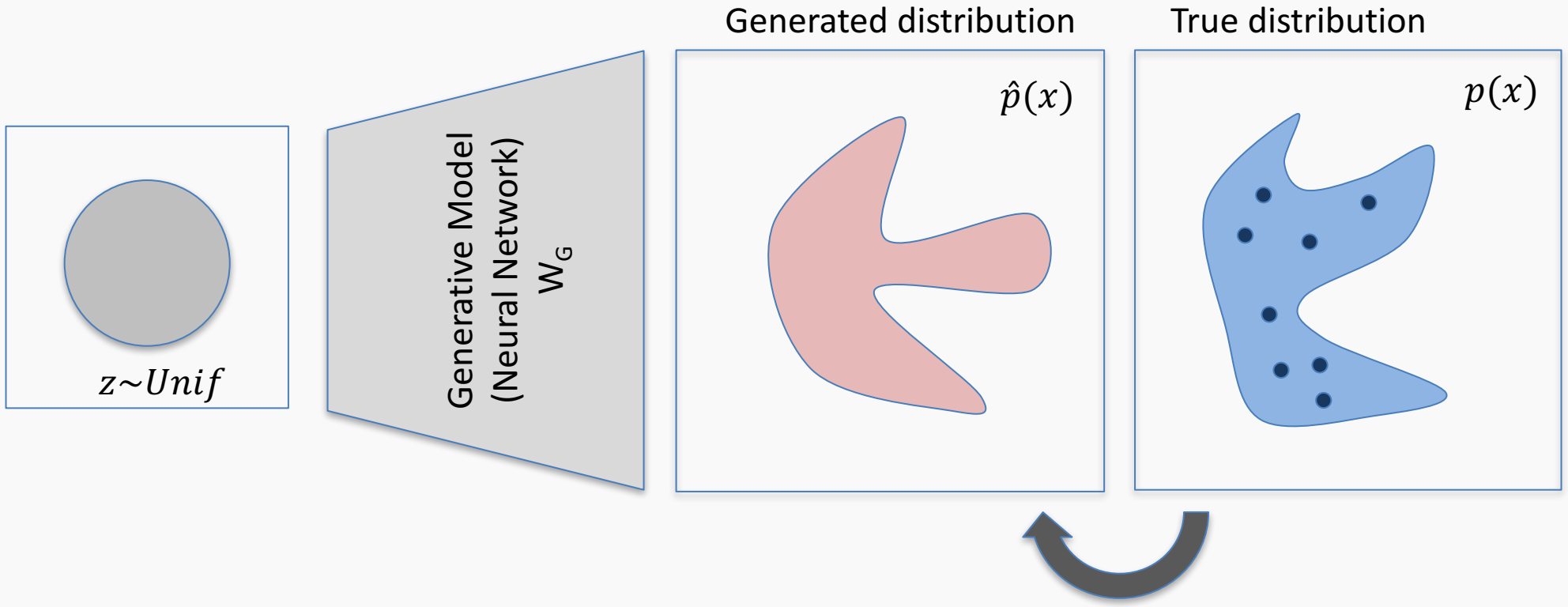
Training the GAN



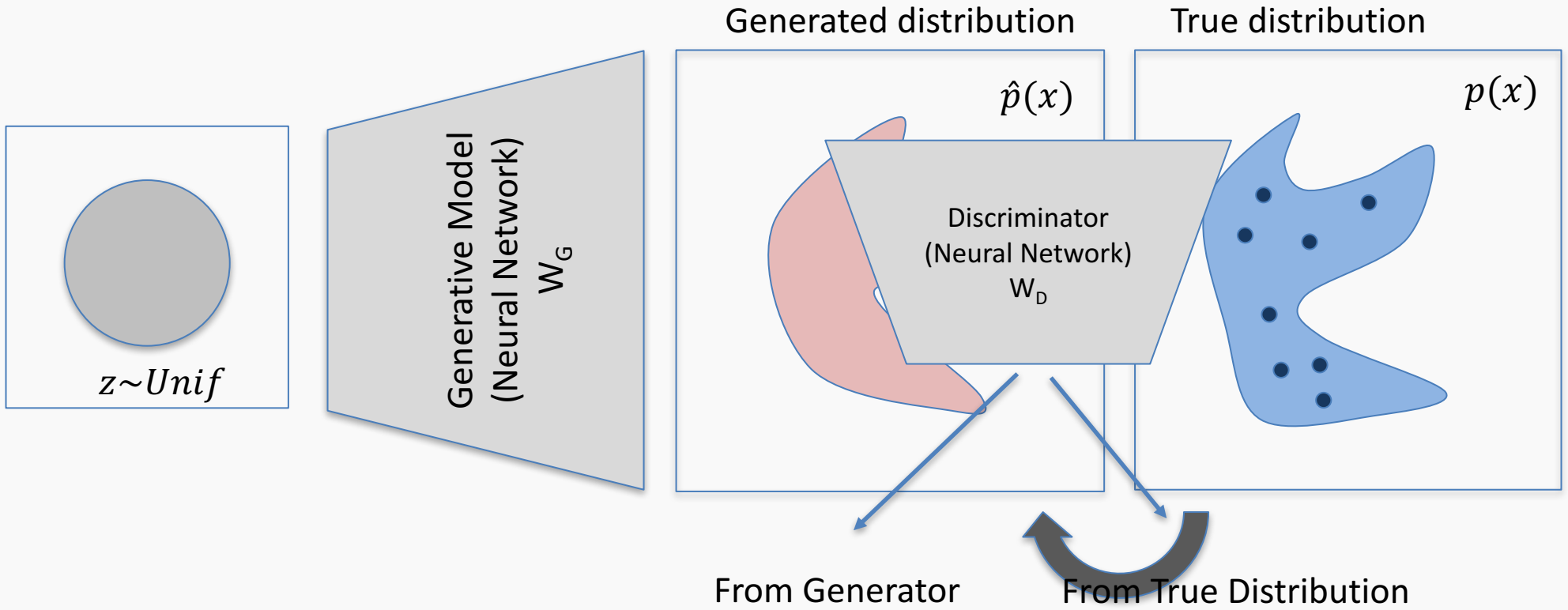
False positives (Fake/D:Real):

Here we generate a fake and punish the discriminator if it classifies it as real.

Generative Models



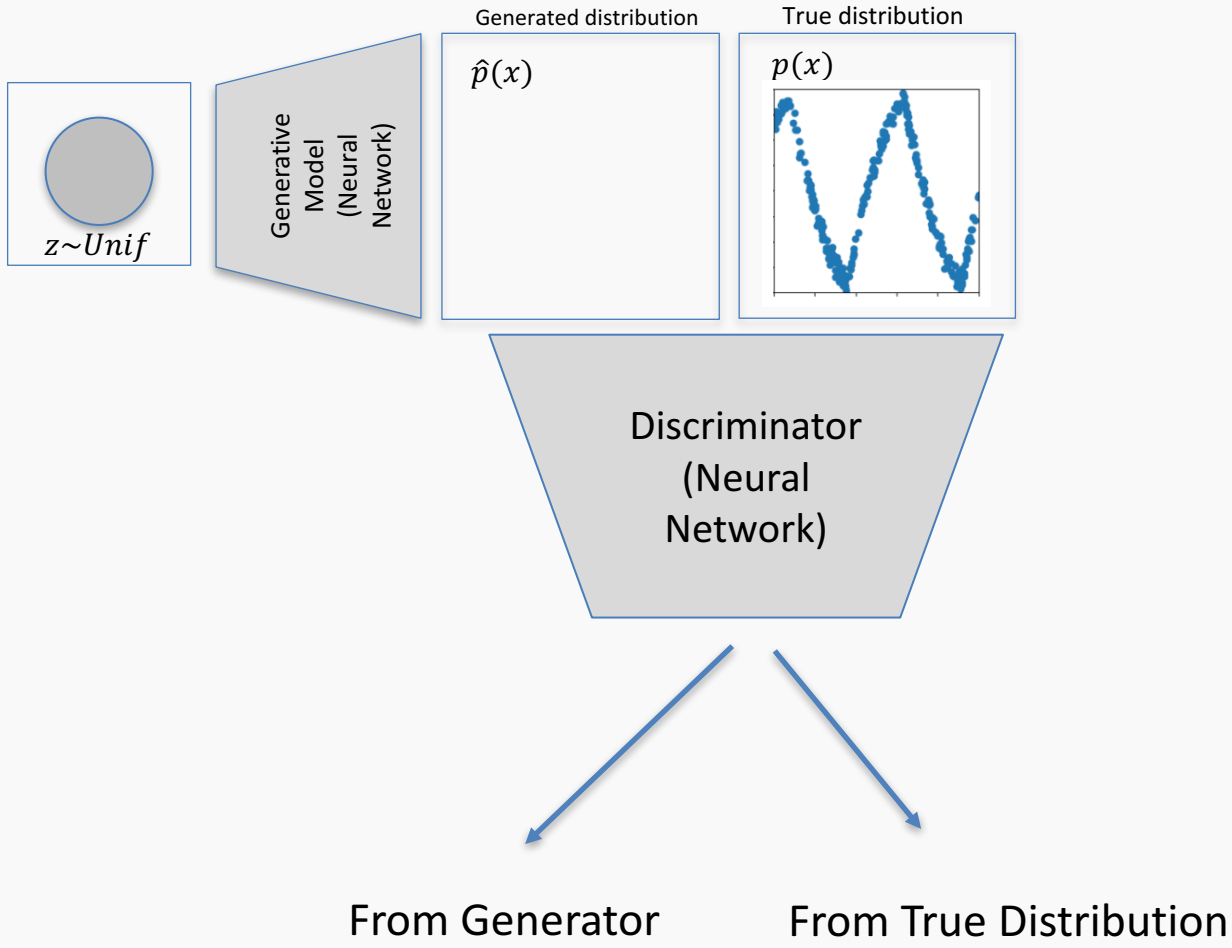
Generative Adversarial Networks



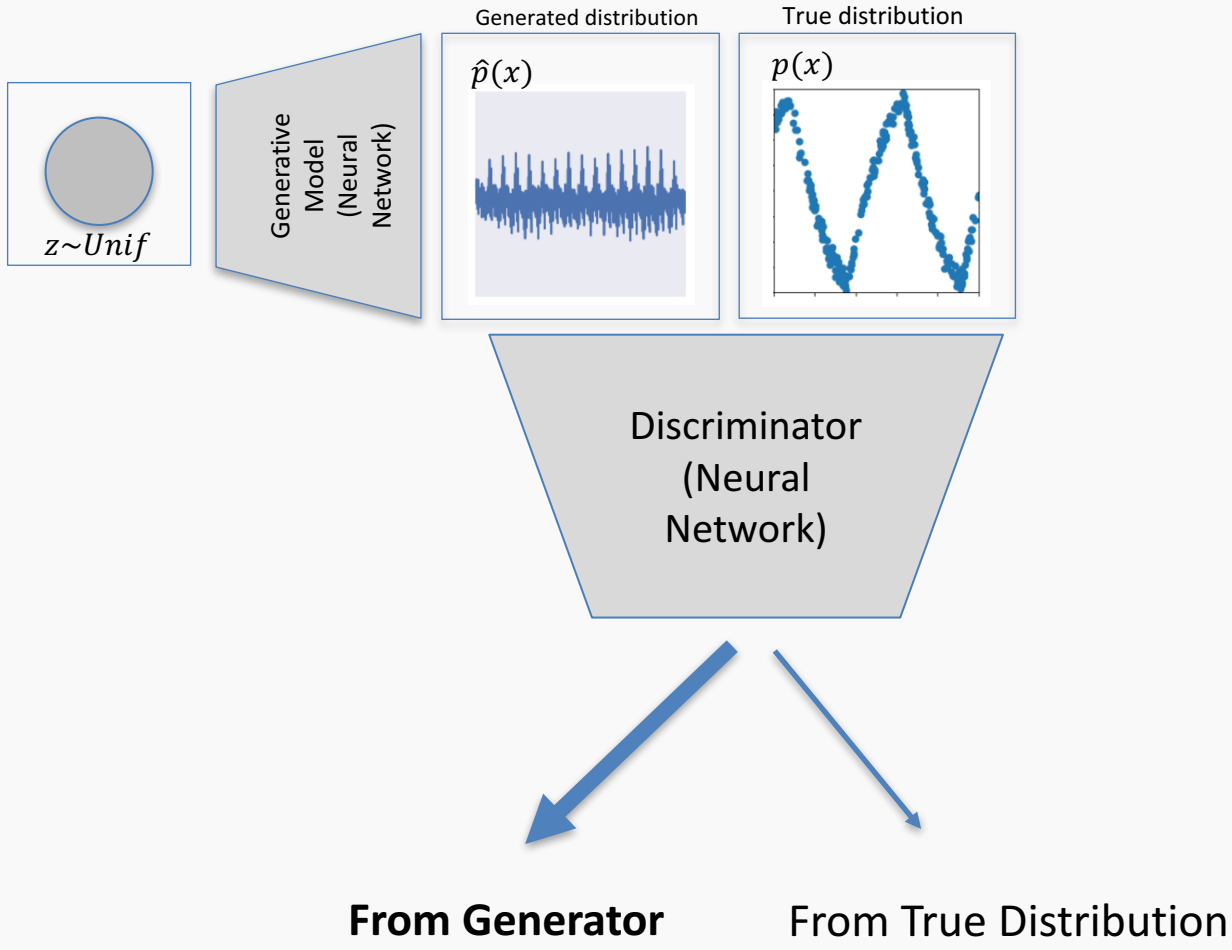
Loss function: Binary Cross Entropy, L_D

KL Divergence

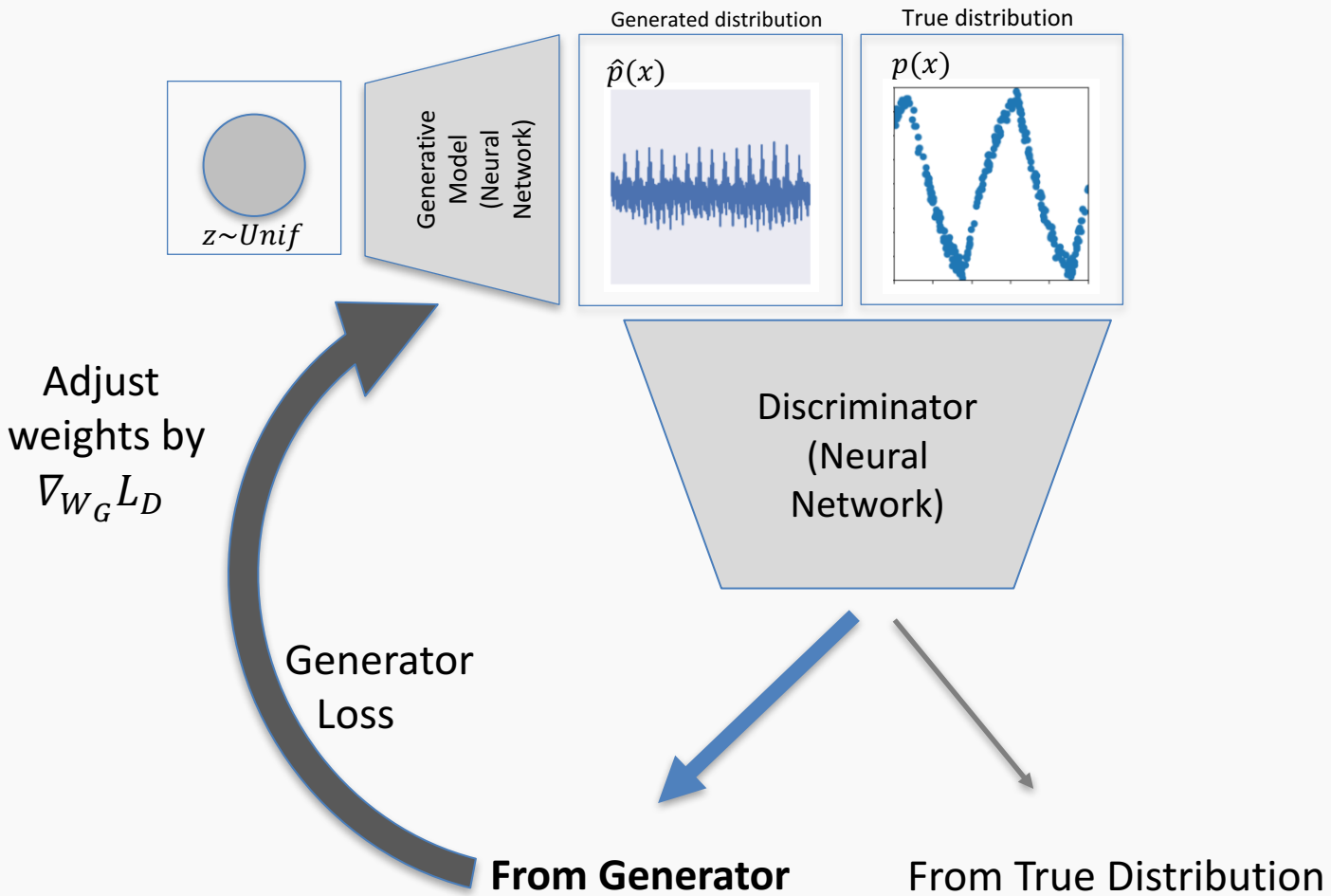
Generative Adversarial Networks



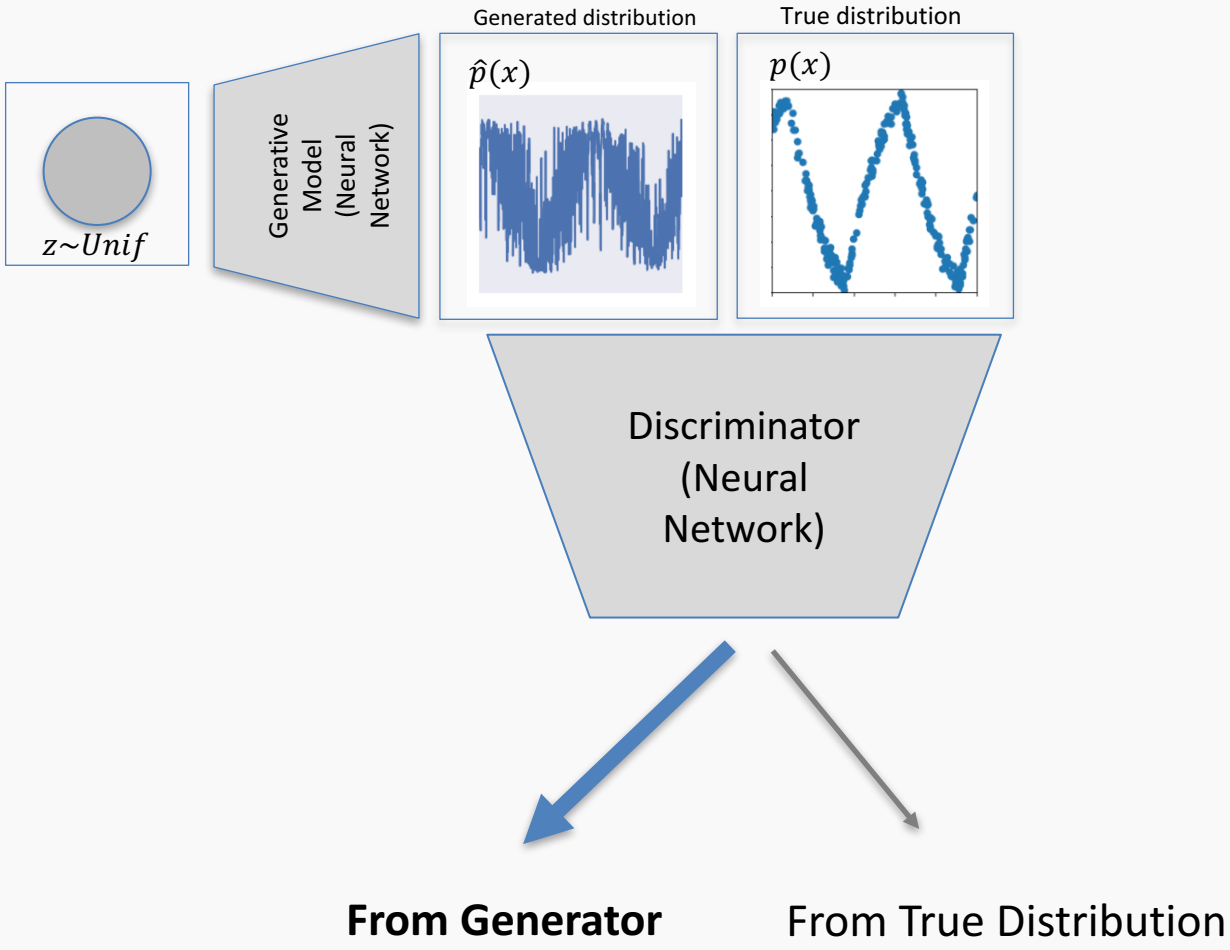
Generative Adversarial Networks



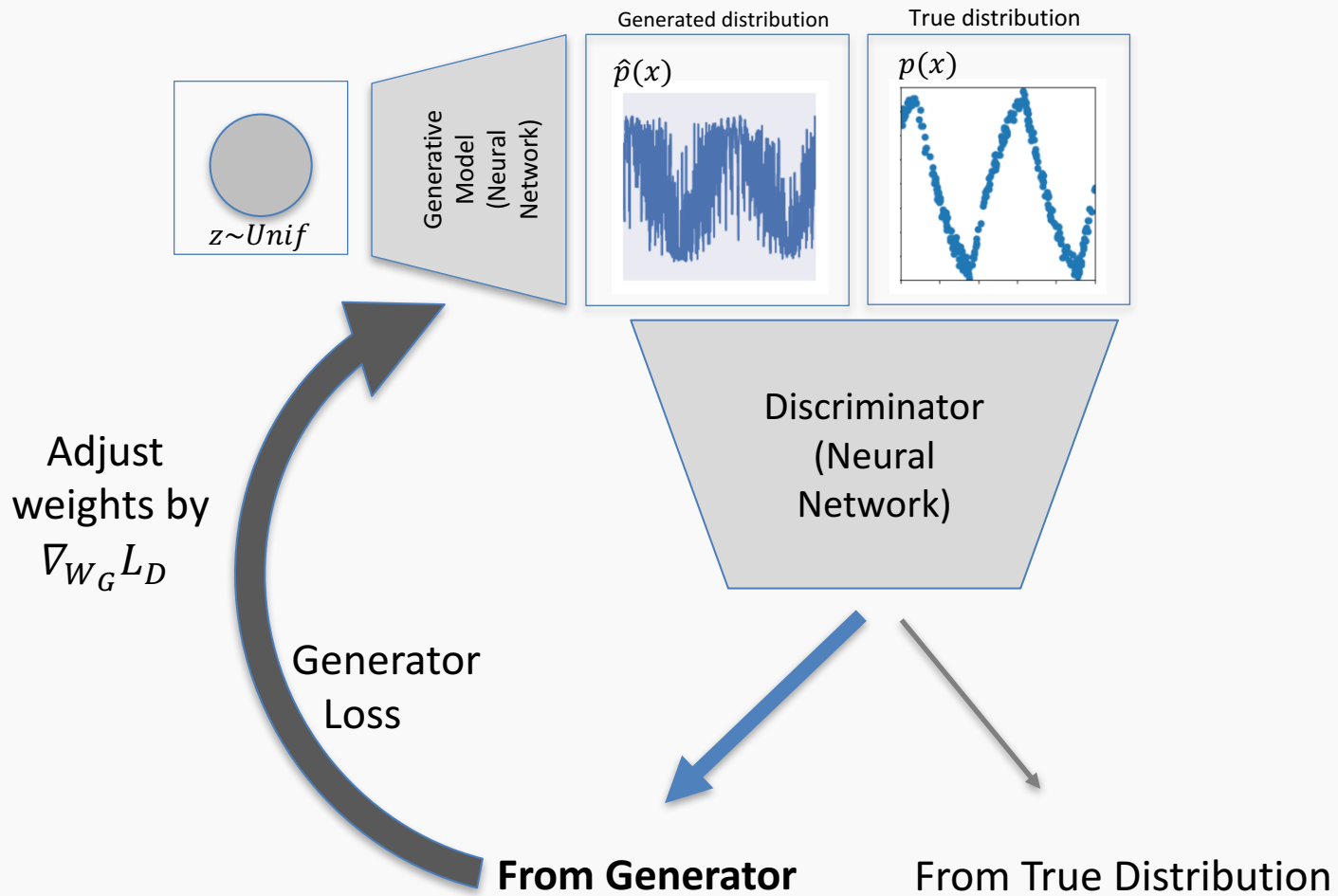
Generative Adversarial Networks



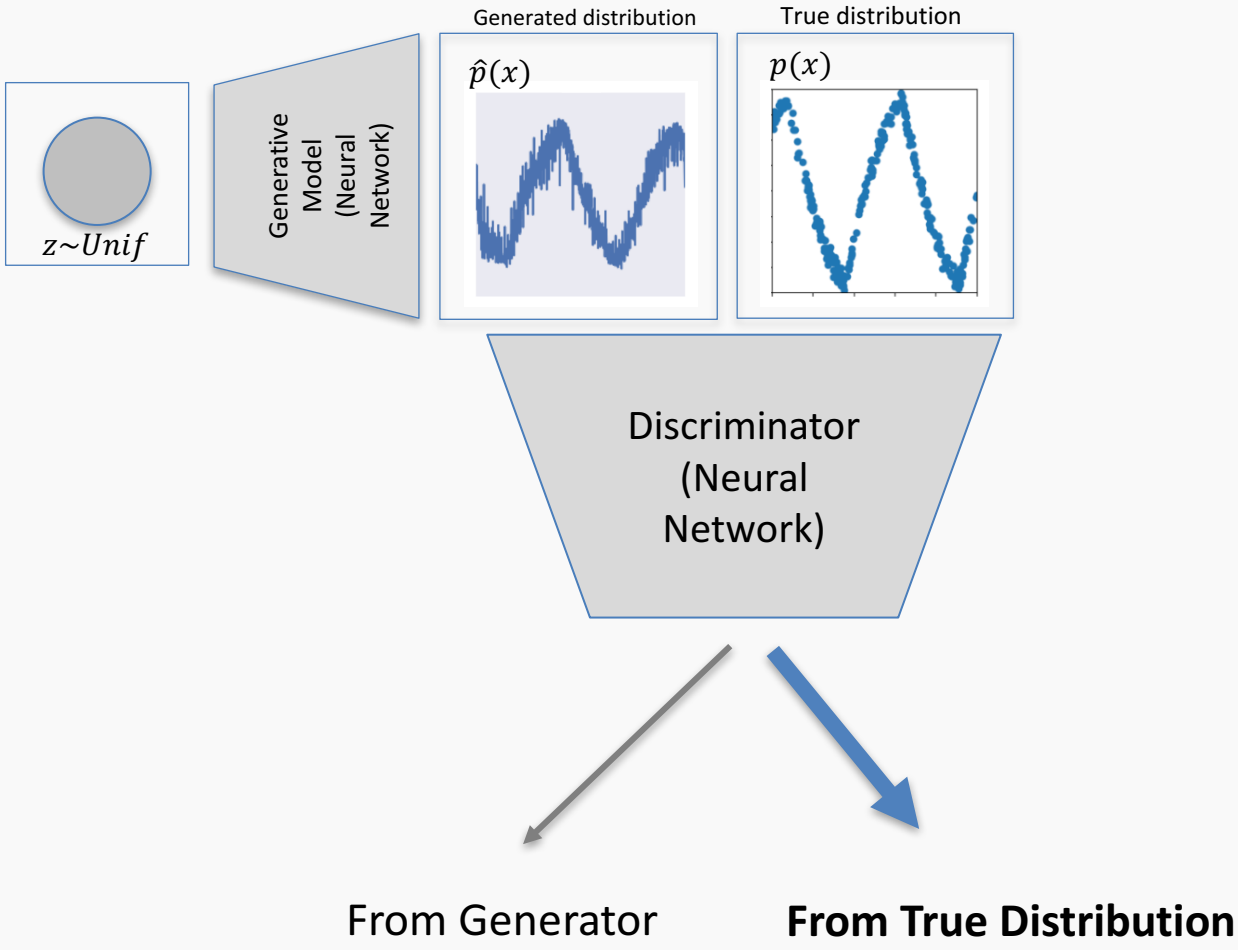
Generative Adversarial Networks



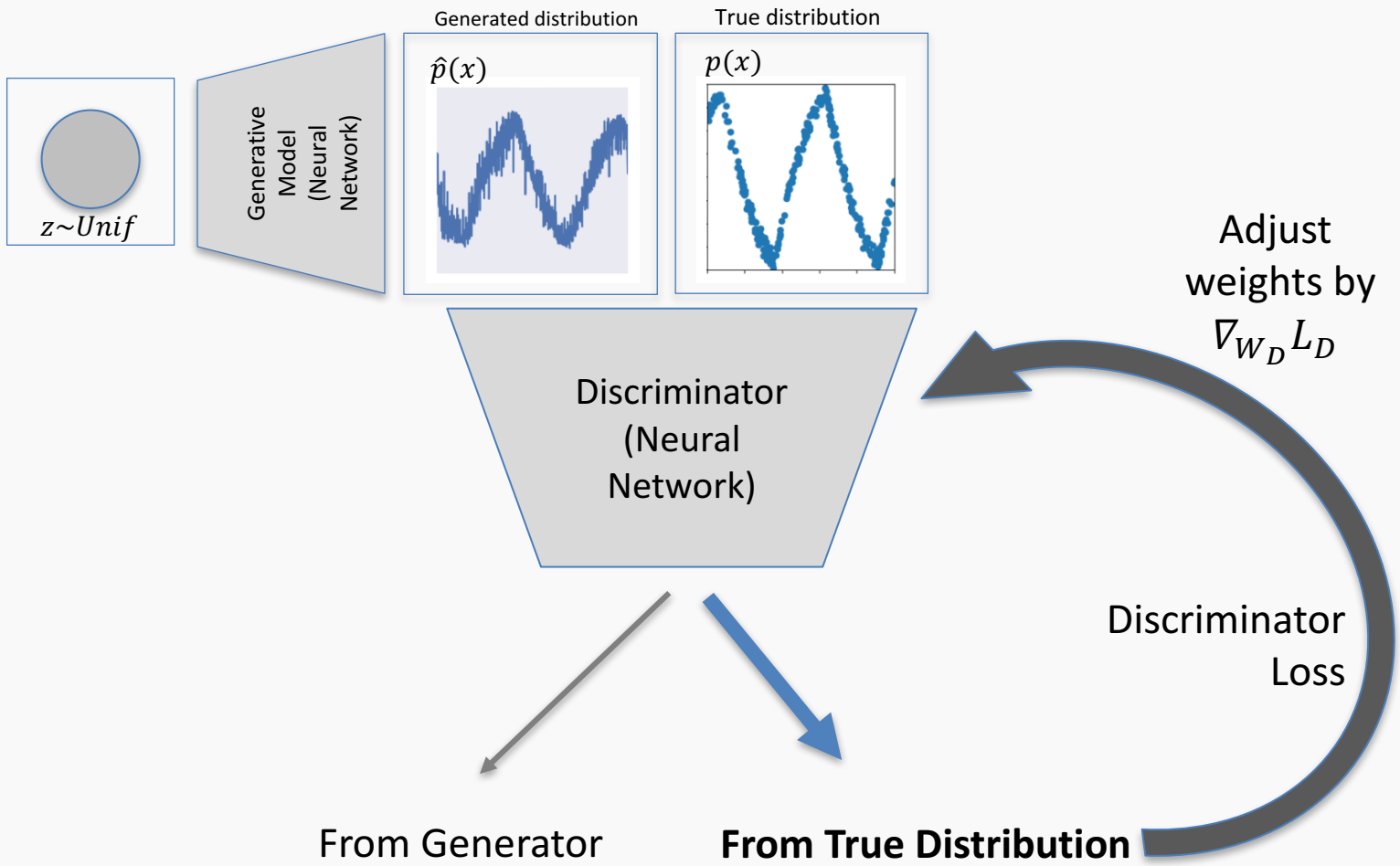
Generative Adversarial Networks



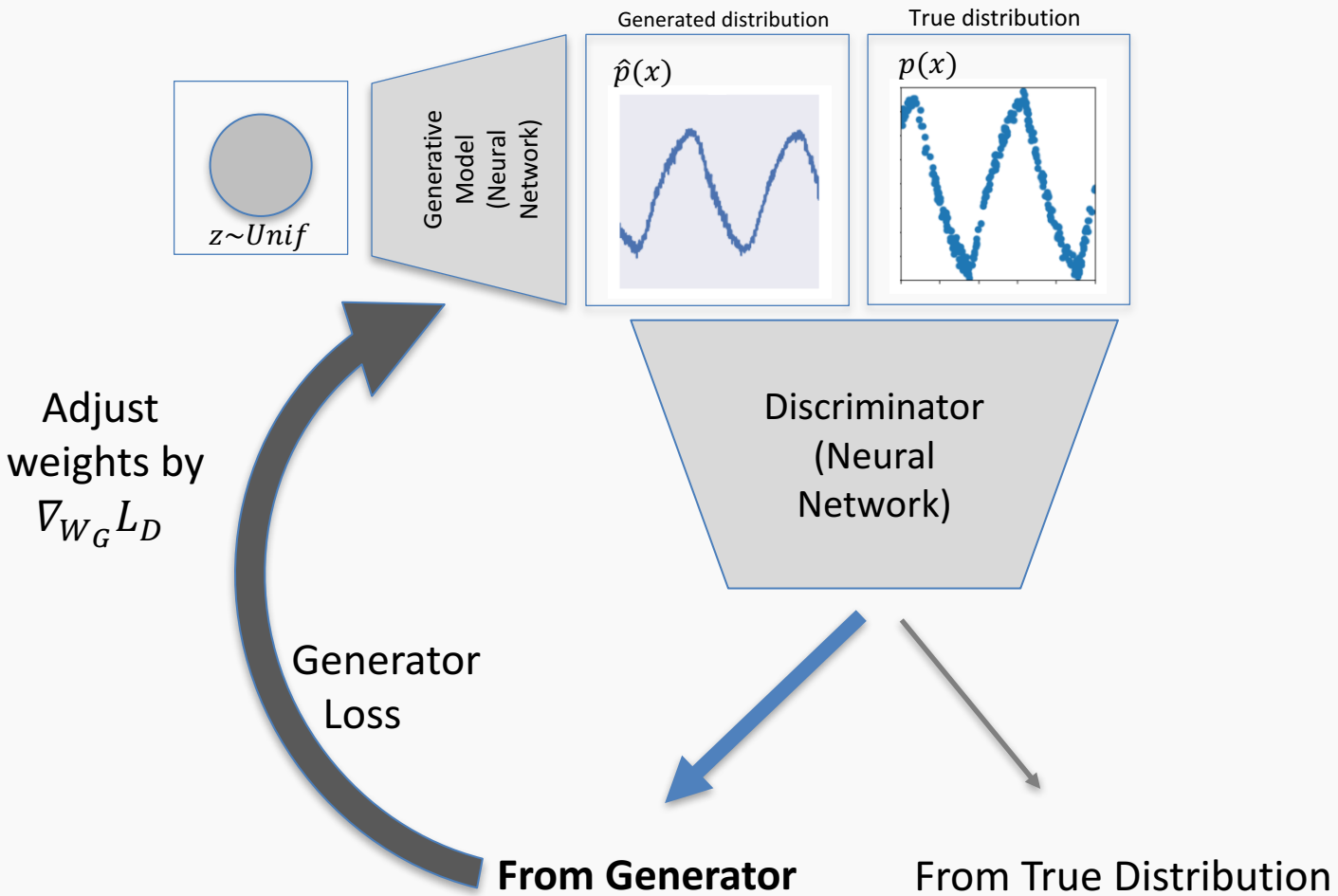
Generative Adversarial Networks



Generative Adversarial Networks



Generative Adversarial Networks

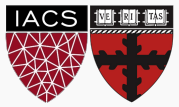


Game Theory

In some games there are unbounded resources. For example, in a game of poker, the pot can theoretically get larger and larger without limit.

Zero-sum game: Players compete for a fixed and limited pool of resources. Players compete for resources, claiming them and each player's total number of resources can change, but the total number of resources remain constant.

In zero-sum games each player can try to set things up so that the other player's best move is of as little advantage as possible. This is called a **minimax**, or **minmax**, technique.



Game Theory (cont)

Our goal in training the GAN is to produce two networks that are each as good as they can be. In other words, we don't end up with a "win- ner." Instead, both networks have reached their peak ability given the other network's abilities to thwart it. Game theorists call this state a **Nash equilibrium**, where each network is at its best configuration with respect to the other.

More on this in the lab tomorrow.

Challenges

Biggest challenge to using GANs is practice is their sensitivity to both structure and parameters.

If either the discriminator or generator gets better than the other too quickly, the other will never be able to catch up.

Finding that combination can be challenging. Following the rules of thumb we discussed above is generally recommended when we're building a new DCGAN.

Challenges (cont)

Also there is no proof that they will **converge**.

GANs do seem to perform very well most of the time when we find the right parameters, but there's no guarantee beyond that.

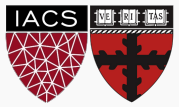
Challenges: Using Big Samples

Trying to train a GAN generator too produce large images, such as 1000 by 1000 pixels can be problematic.

The problem is that with large images, it's easy for the discriminator to tell the generated fakes from the real images.

Many pixels can lead to error gradients that cause the generator's output to move in almost random directions, rather than getting closer to matching the inputs.

Compute power, memory, and time to process large numbers of these big samples.



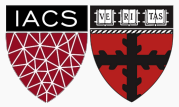
Challenges: Using Big Samples (cont)

Trying to train a GAN generator too produce large images, such as 1000 by 1000 pixels can be problematic.

The problem is that with large images, it's easy for the discriminator to tell the generated fakes from the real images.

Many pixels can lead to error gradients that cause the generator's output to move in almost random directions, rather than getting closer to matching the inputs.

Compute power, memory, and time to process large numbers of these big samples.

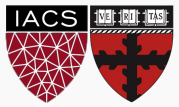


Challenges: Using Big Samples (cont)

- Start by resizing the images: 512x512, 128x128, 64x64, … ,4x4.
- Then build a small generator and discriminator, each with just a few layers of convolution.
- Train with the 4 by 4 images until it does well.
- Add a few more convolution layers to the end network, and now train them with 8 by 8 images. Again, when the results are good, add some more convolution layers to the end of each network and train them on 16 by 16 images.

This process takes much less time to complete than if we'd trained with only the full-sized images from the start

.



Challenges: Modal collapse

I would like to use GAN to produce faces like the ones below from NVIDIA [Karras, Laine, Aila / Nvidia].

Challenges: Modal collapse (cont)

The generator somehow finds one image that fools the discriminator.

[Karras, Laine, Aila / Nvidia].

Challenges: Modal collapse (cont)

A generator could then just produce that image every time independently of the input noise.

The discriminator will always say it is real, so the generator has accomplished its goal and stops learning.

However: The problem is that every sample made by the generator is identical.

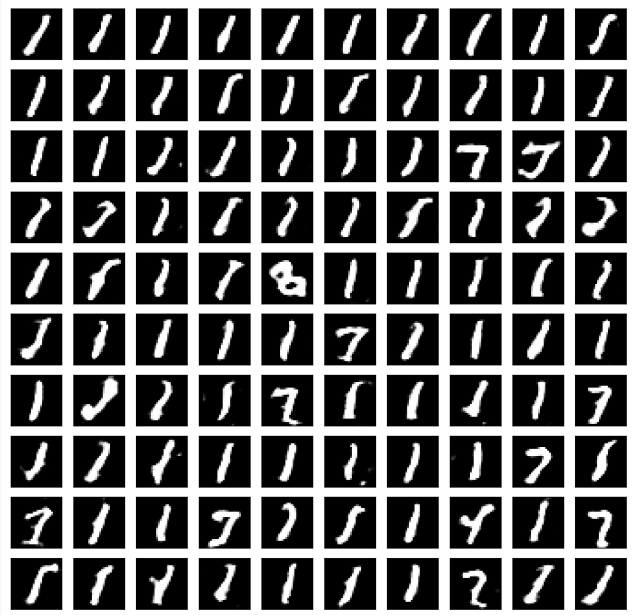
This problem of producing just one successful output over and over is called **modal collapse**.

[Karras, Laine, Aila / Nvidia].

Challenges: Modal collapse (cont)

Much more common is when the system produces the same few outputs, or minor variations of them.

This is called partial modal collapse



Solution:

- Extend the discriminator's loss function with an additional terms to measure the diversity of the outputs produced.
- If the outputs are all the same, or nearly the same, the discriminator can assign a larger error to the result.
- The generator will diversify because that action will reduce the error

Adversarial Neural Networks

How vulnerable are Neural Networks?

Uses of Neural Networks

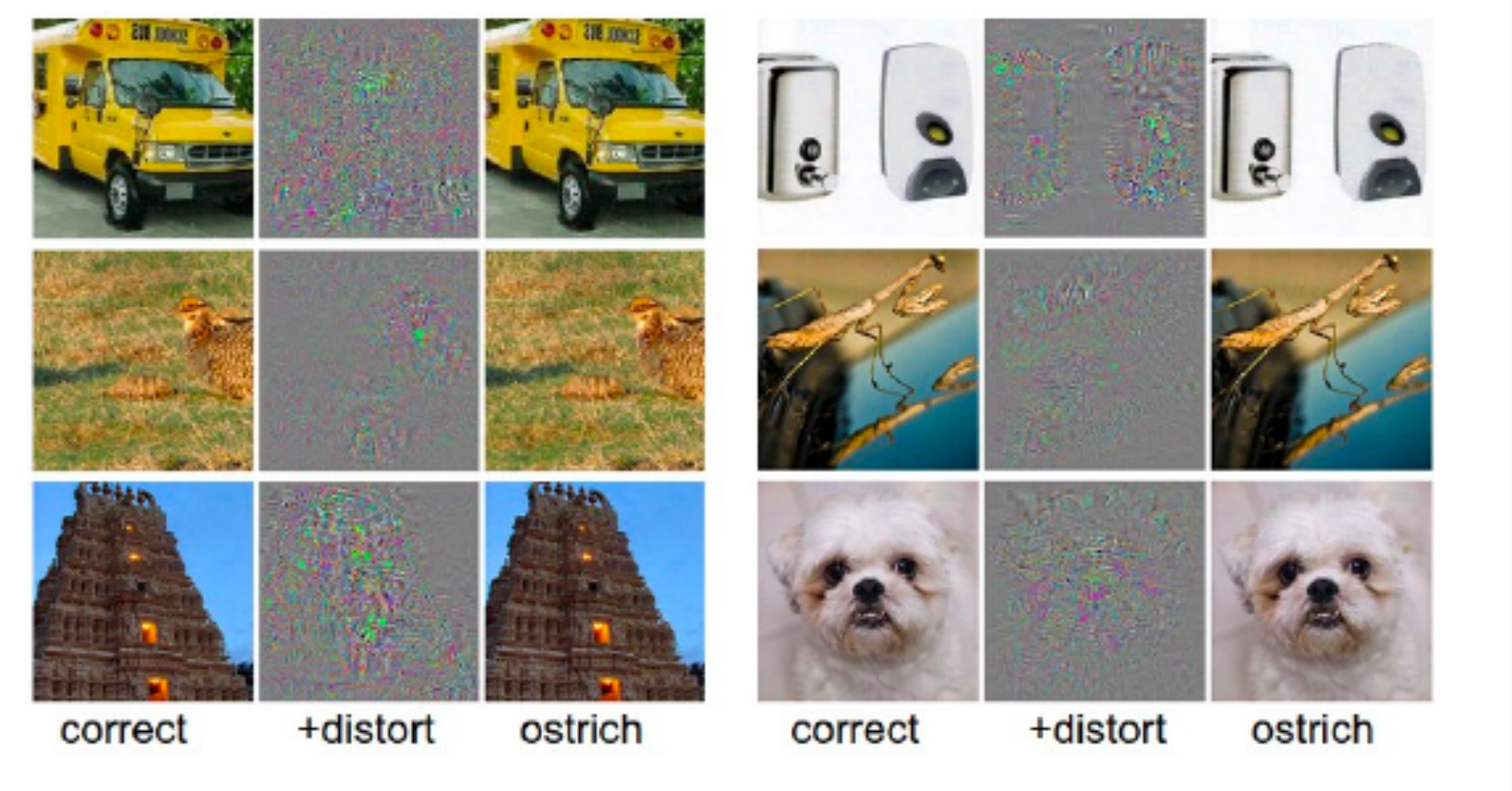
How vulnerable are Neural Networks?

Explaining Adversarial Examples

[Goodfellow et. al ‘15]

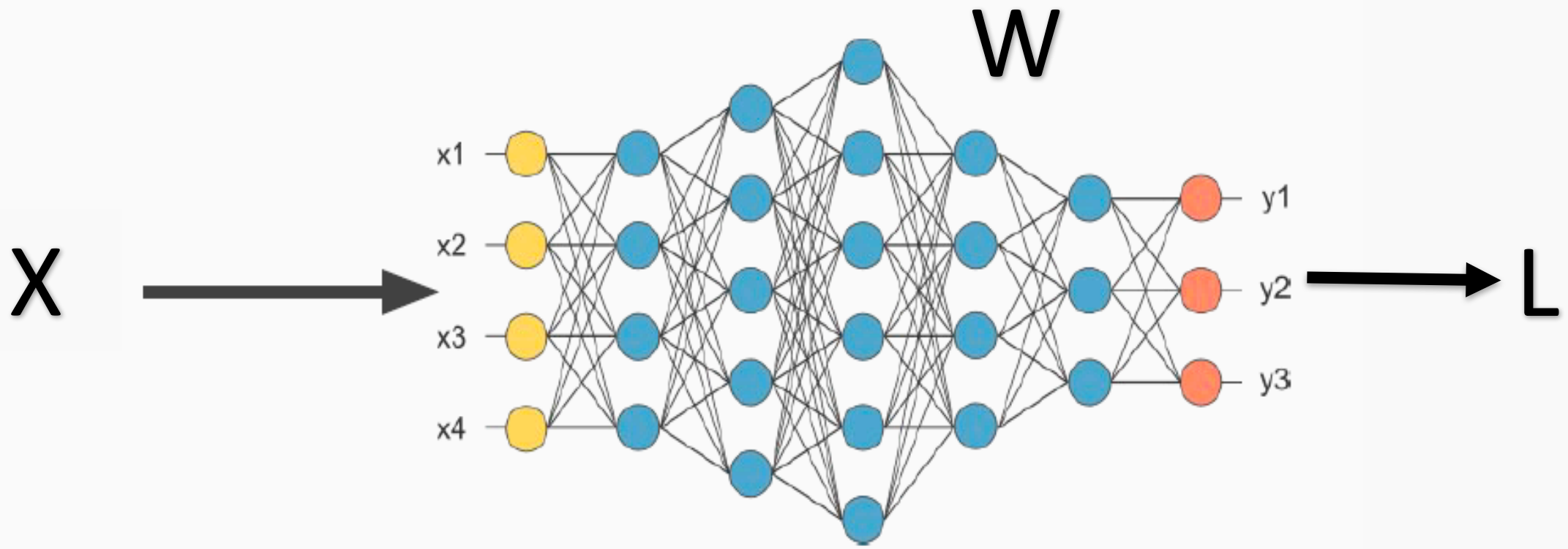
1. Robust attacks with FGSM
2. Robust defense with Adversarial Training

Explaining Adversarial Examples



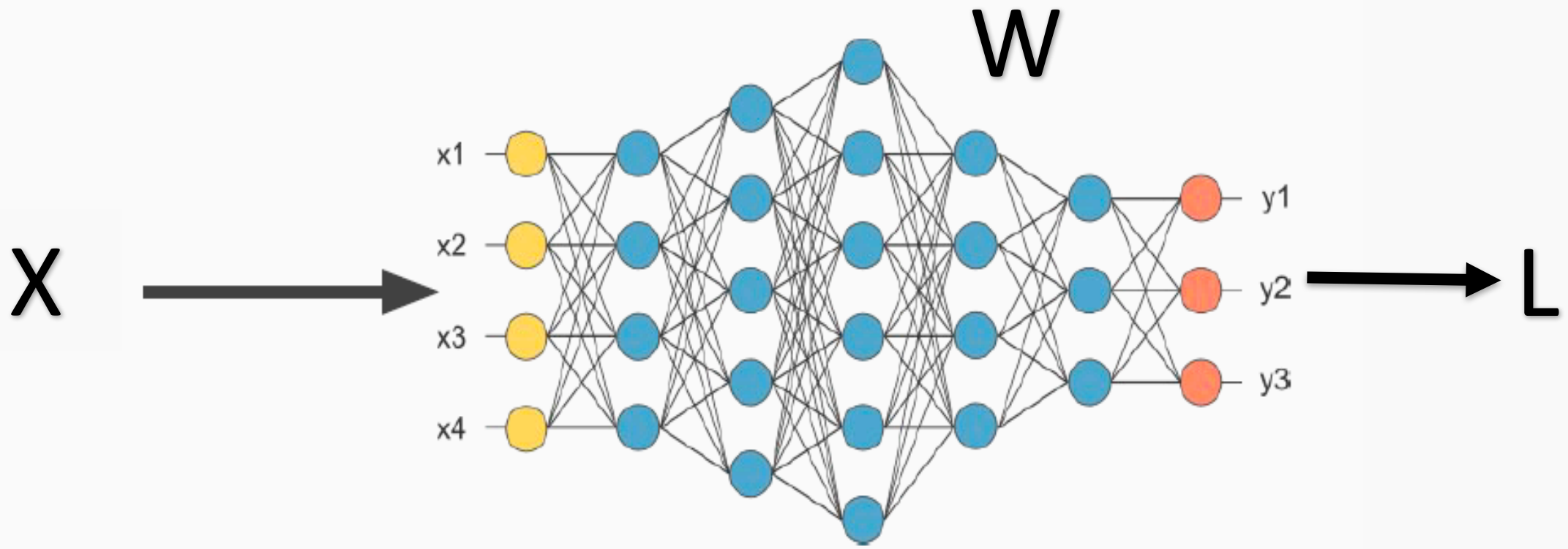
Some of these adversarial examples can even fool humans:

Attacking with **F**ast **G**radient **S**ign **M**ethod (FGSM)



$$x + \lambda \cdot \text{sign}(\nabla_{\text{x}} \text{L}) \Rightarrow \text{x}^*$$

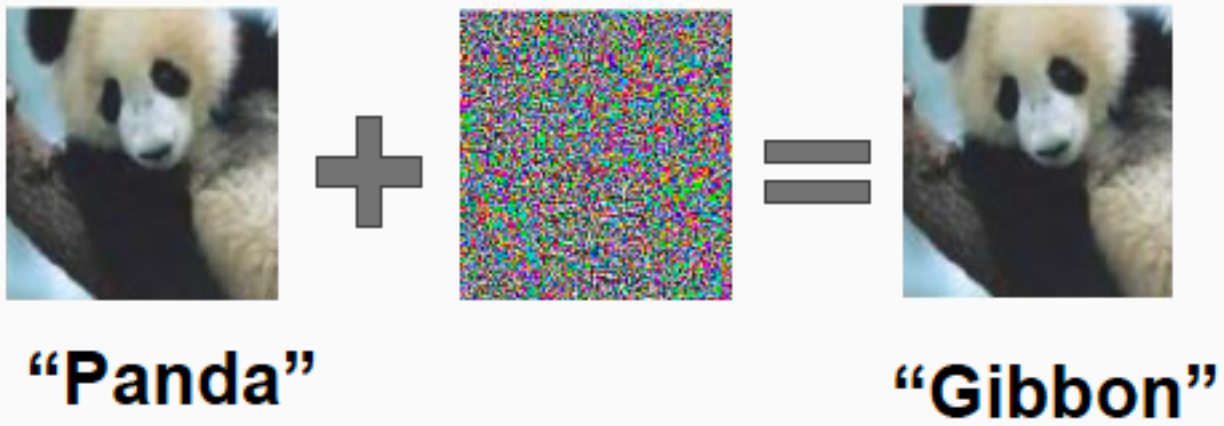
Attacking with **F**ast **G**radient **S**ign **M**ethod (FGSM)



$$x + \lambda \cdot \text{sign}(\nabla_x \text{L}) \Rightarrow \text{x}^*$$

$$x + \lambda \cdot \text{sign}(\nabla_x L) \Rightarrow x^*$$

Defending with Adversarial Training



1. Generate adversarial examples
2. Adjust labels

Defending with Adversarial Training

1. Generate adversarial examples
2. Adjust labels

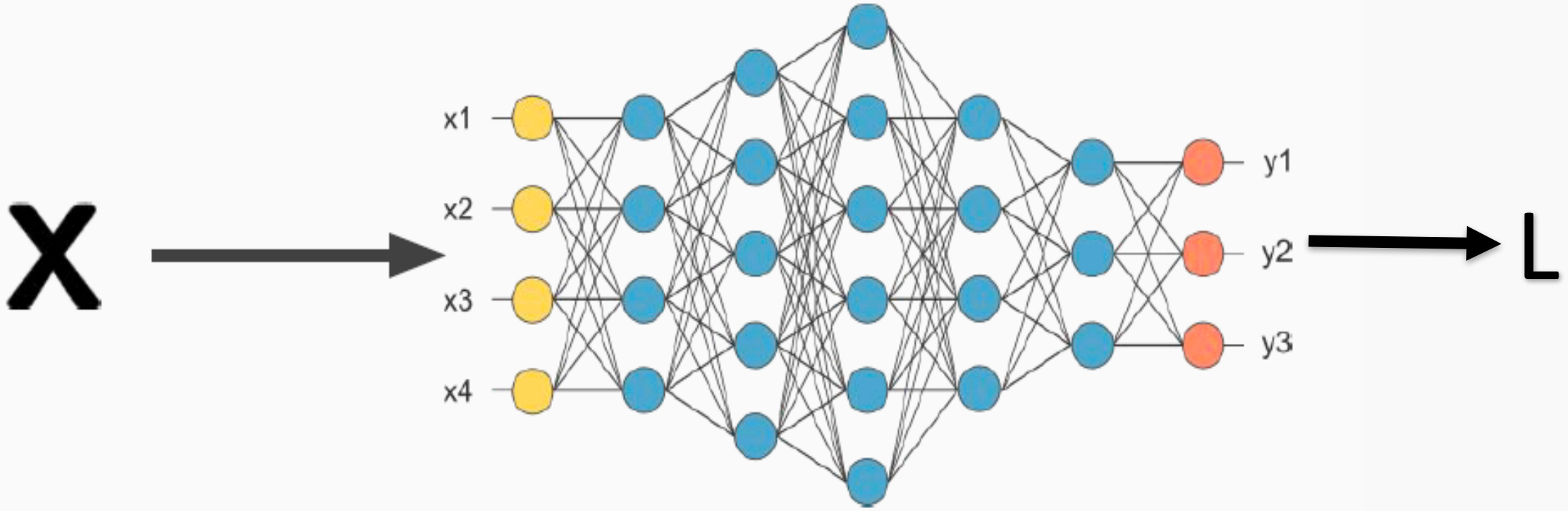
Defending with Adversarial Training

1. Generate adversarial examples
2. Adjust labels
3. Add them to the training set
4. Train new network

Attack methods post GoodFellow 2015

- FGSM [Goodfellow et. al ‘15]
- JSMA [Papernot et. al ‘16]
- C&W [Carlini + Wagner ‘16]
- Step-LL [Kurakin et. al ‘17]
- I-FGSM [Tramer et. al ‘18]

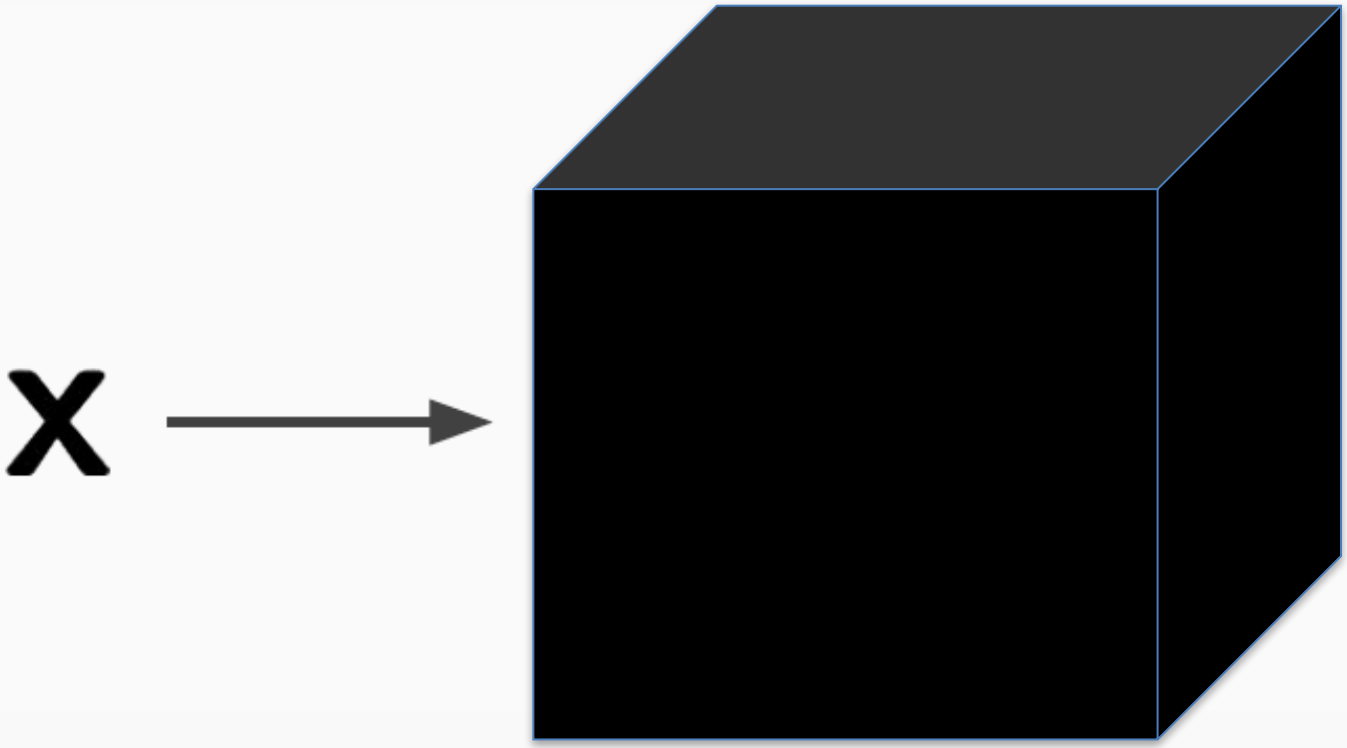
White box attacks



$$x + \lambda \cdot \text{sign}(\nabla_x \text{L}) \Rightarrow \text{x}^*$$

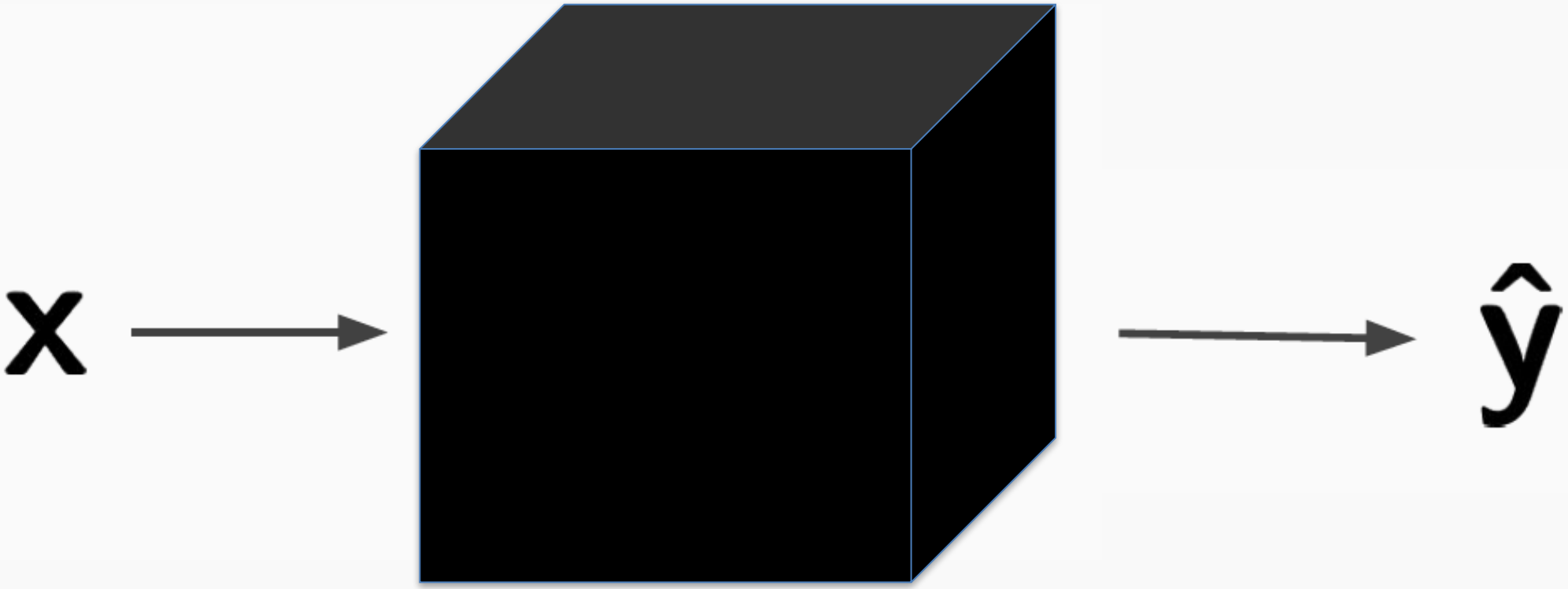
“Black Box” Attacks

“Black Box” Attacks [Papernot et. al ‘17]



“Black Box” Attacks

Examine inputs and outputs of the model

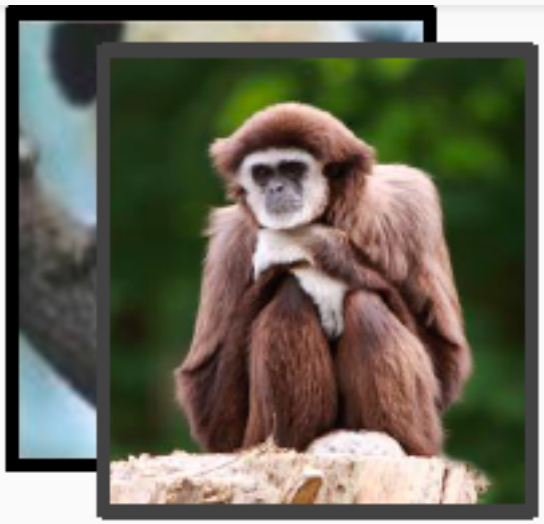


“Black Box” Attacks

Panda

x → [black box] → $\hat{y}$

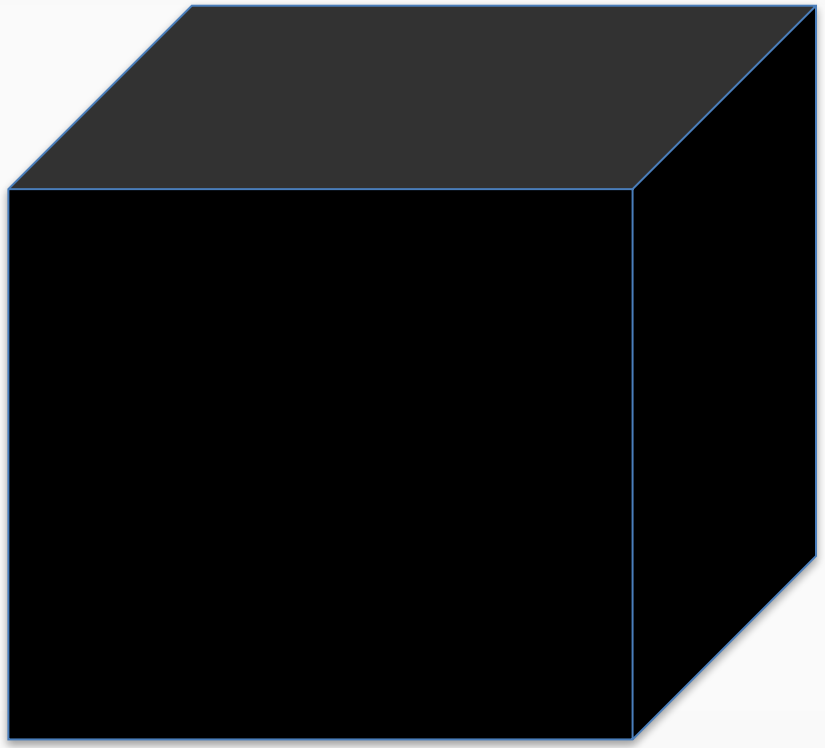
“Black Box” Attacks



Panda

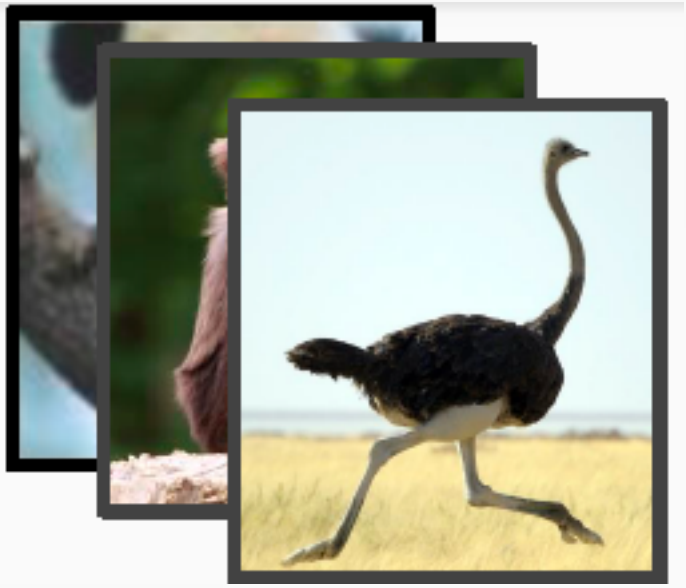
Gibbon

x

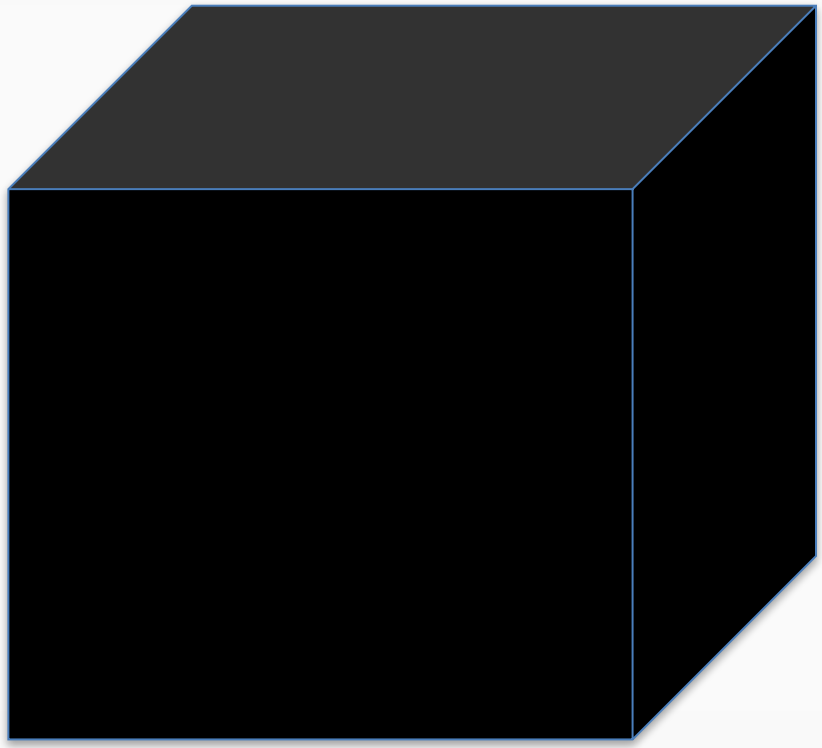


$\hat{y}$

“Black Box” Attacks



x



Panda
Gibbon
Ostrich

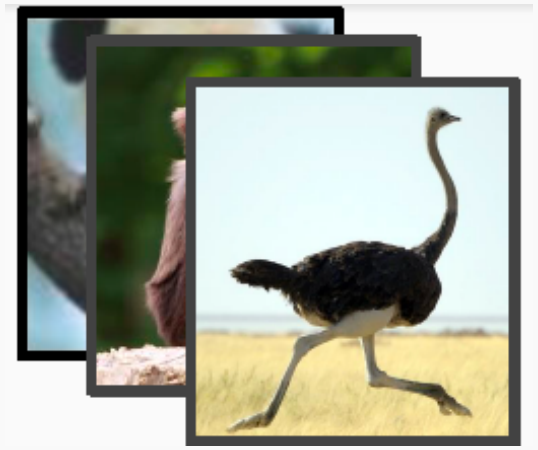
$\hat{y}$

“Black Box” Attacks

Train a model that performs the same as the black box

“Black Box” Attacks

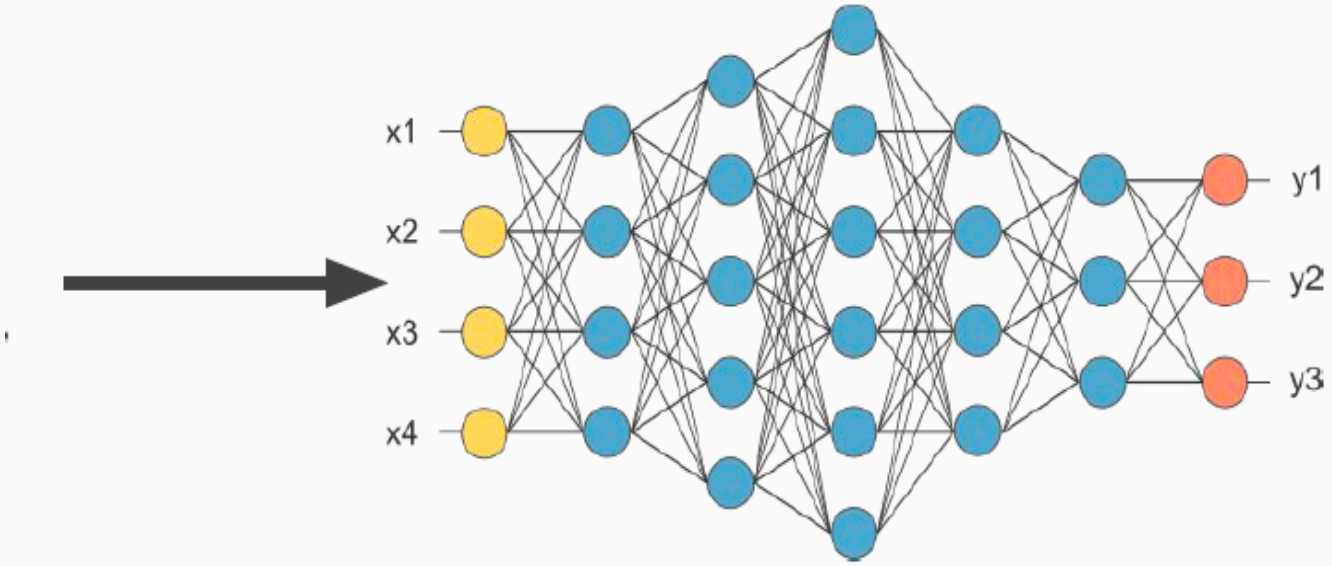
Train a model that performs the same as the black box



Panda

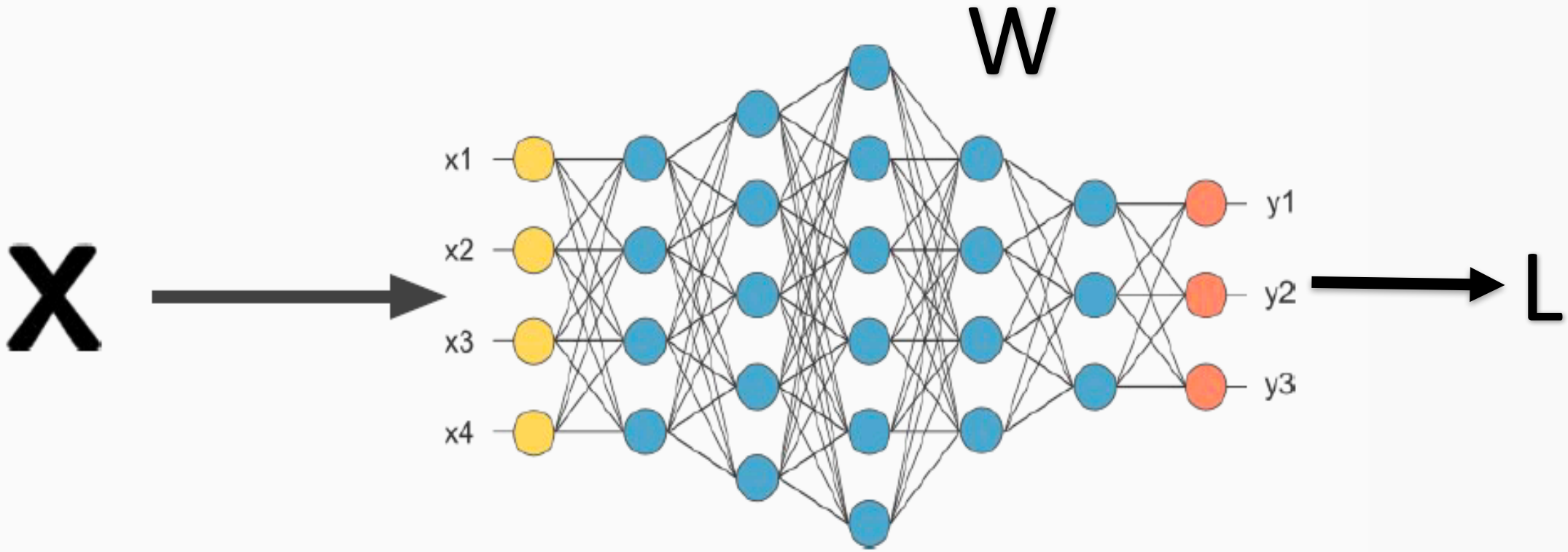
Gibbon

Ostrich



“Black Box” Attacks

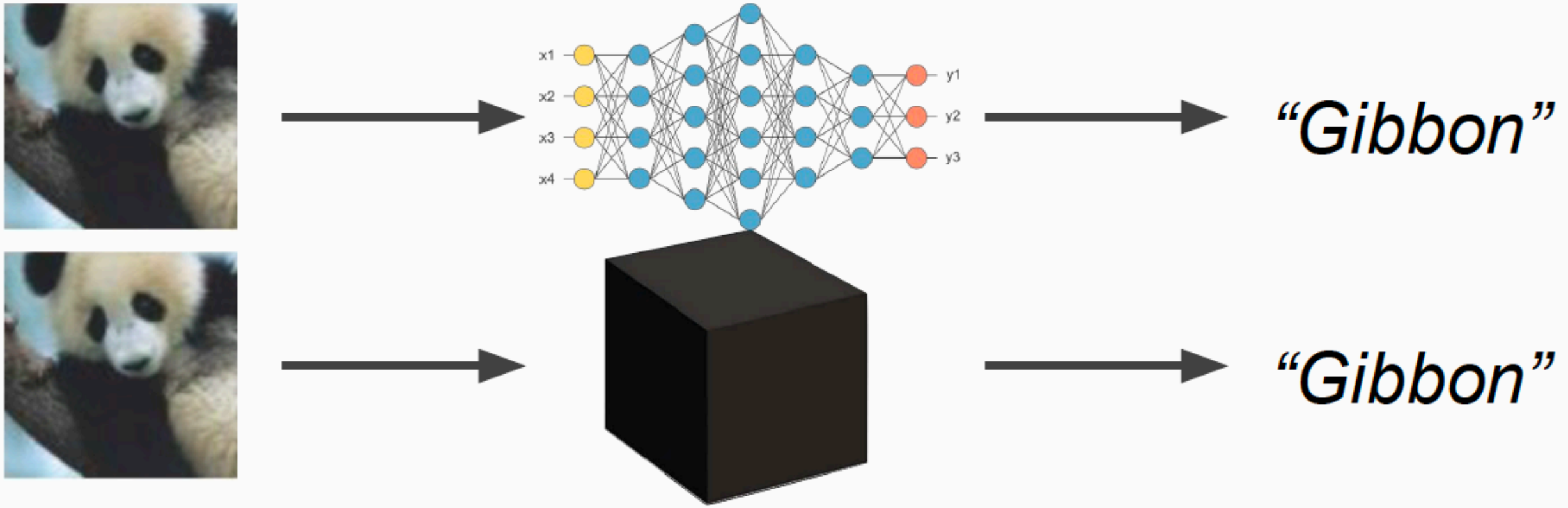
Now attack the model you just trained with “white” box attack



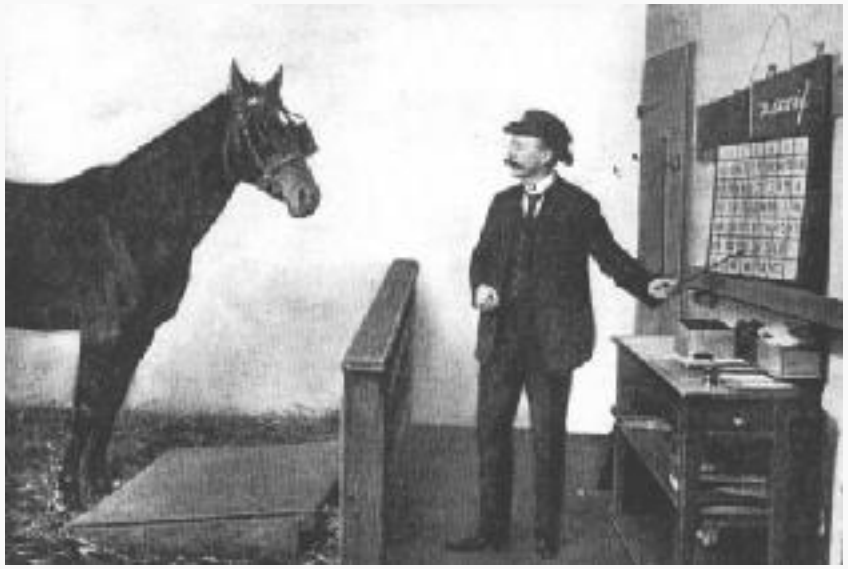
$$x + \lambda \cdot \text{sign}(\nabla_x L) \Rightarrow x^*$$

“Black Box” Attacks

Use those adversarial examples to the “black” box



CleverHans



cleverhans

A Python library to benchmark machine learning systems' vulnerability to adversarial examples.

https://github.com/tensorflow/cleverhans

http://www.cleverhans.io/

More Defenses

Mixup:

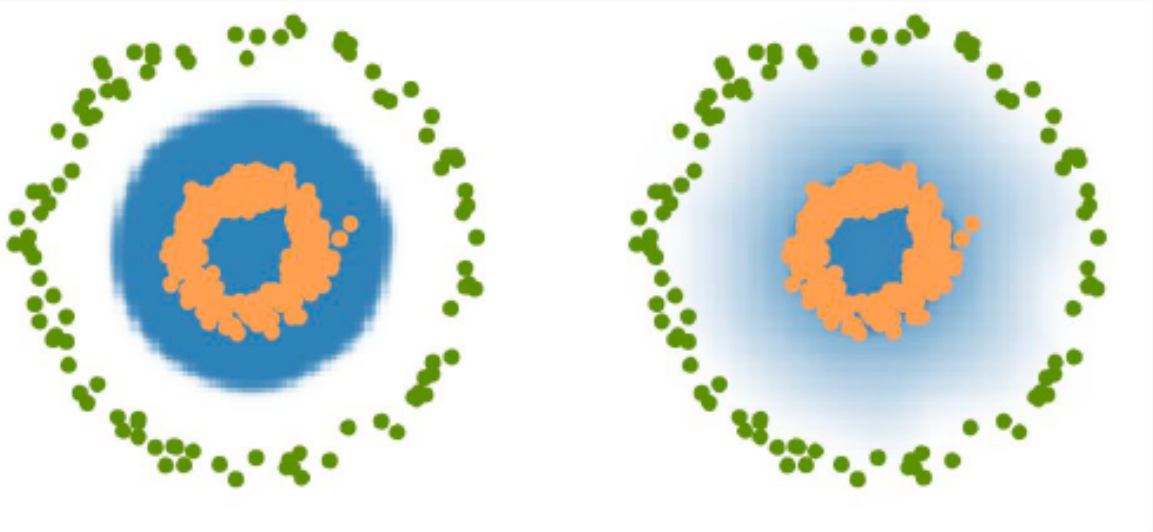
- Mix two training examples
- Augment training set

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

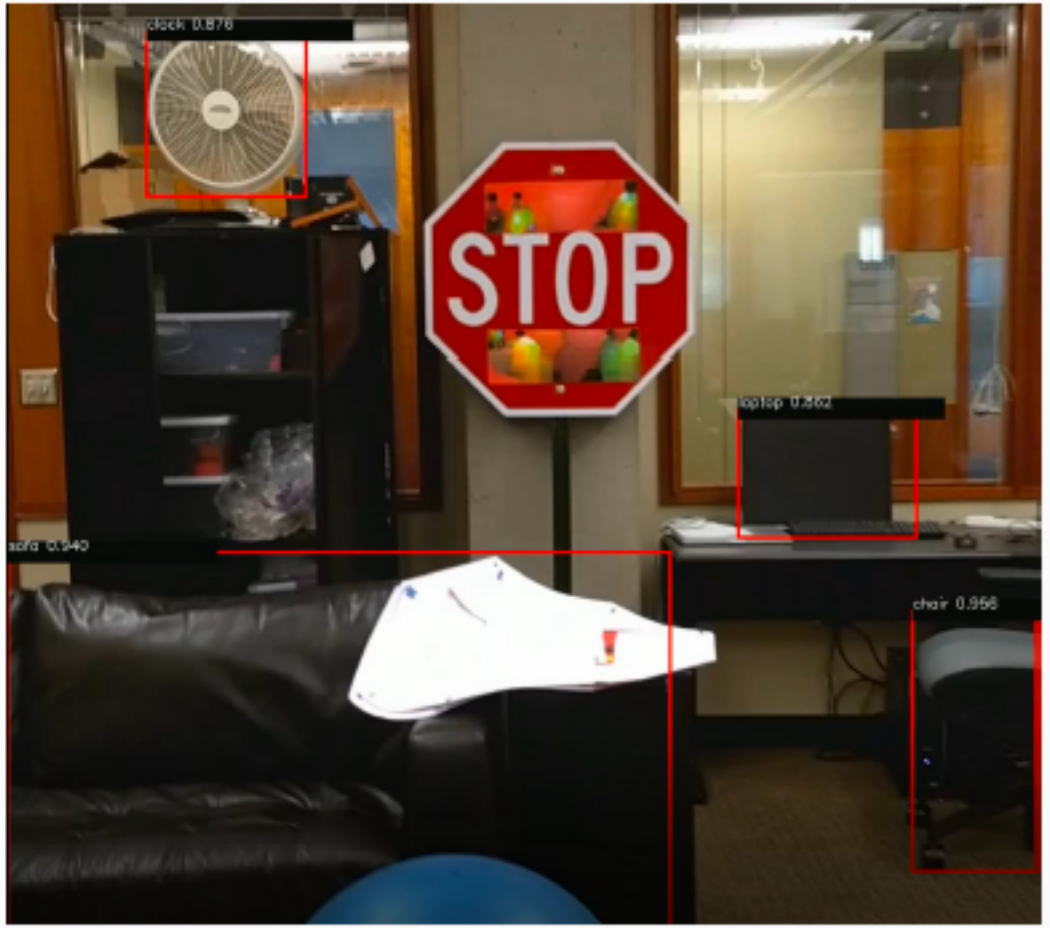
Smooth decision boundaries:

- Regularize the derivatives wrt to x



Physical attacks

- Object Detection
- Adversarial Stickers



Thank you.