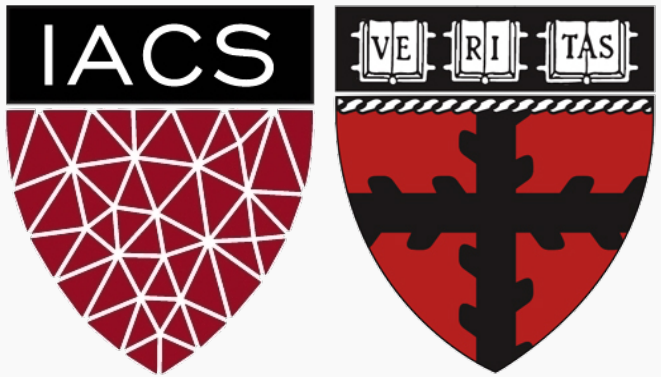
Lecture 20: GANS

CS109B Data Science 2

Pavlos Protopapas and Mark Glickman



Outline

Review of AE and VAE

GANS

- Motivation
- Formalism
- Training

Game Theory, minmax

Challenges:

- Big Samples
- Modal collapse

Outline

Review of AE and VAE

GANS

- Motivation
- Formalism
- Training

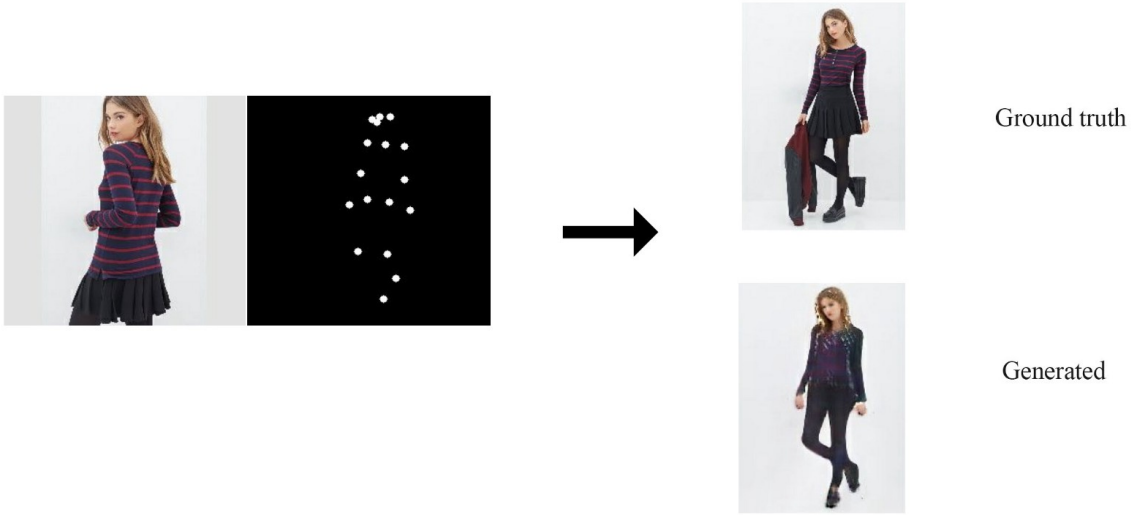
Game Theory, minmax

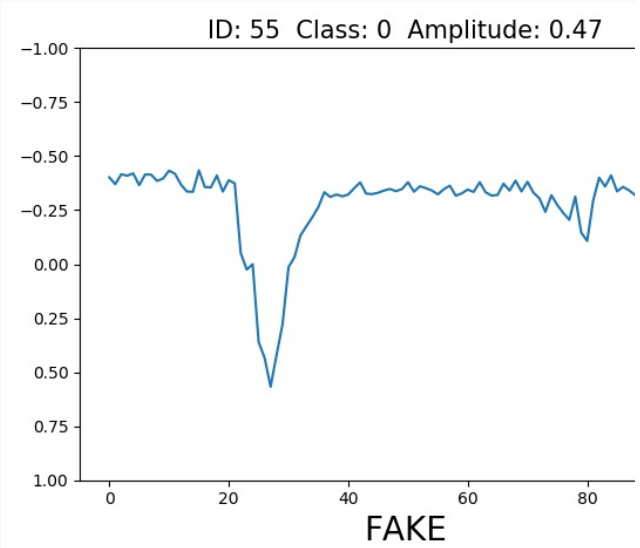
Challenges:

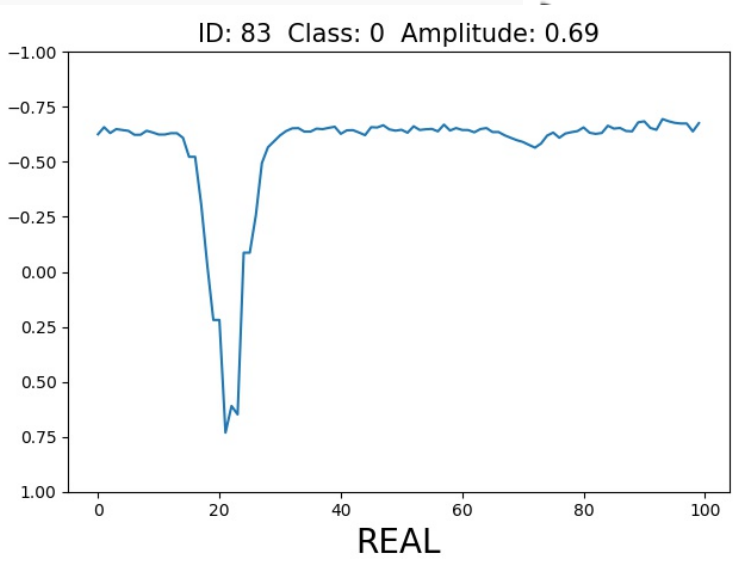
- Big Samples
- Modal collapse

Generating Data (is exciting)

Figure 7: Generated samples

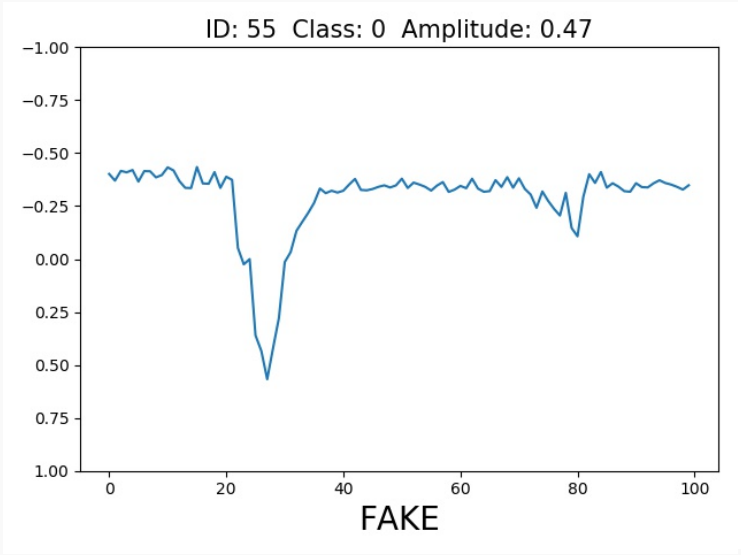


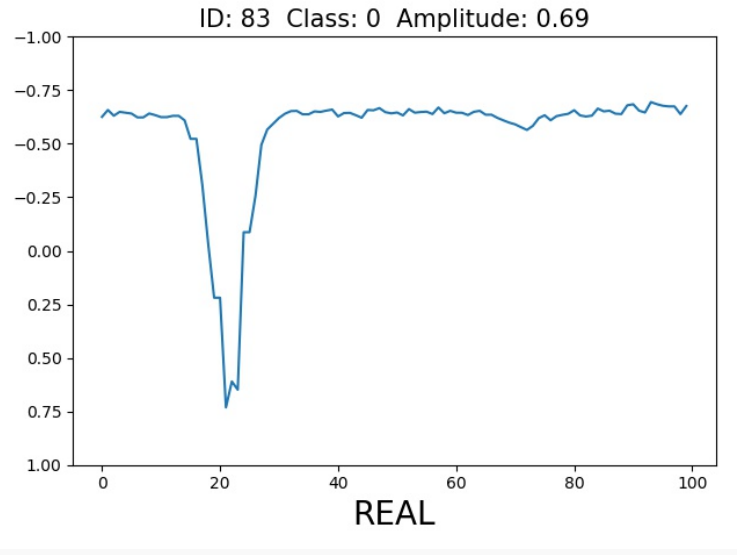




Generating Data (is exciting)

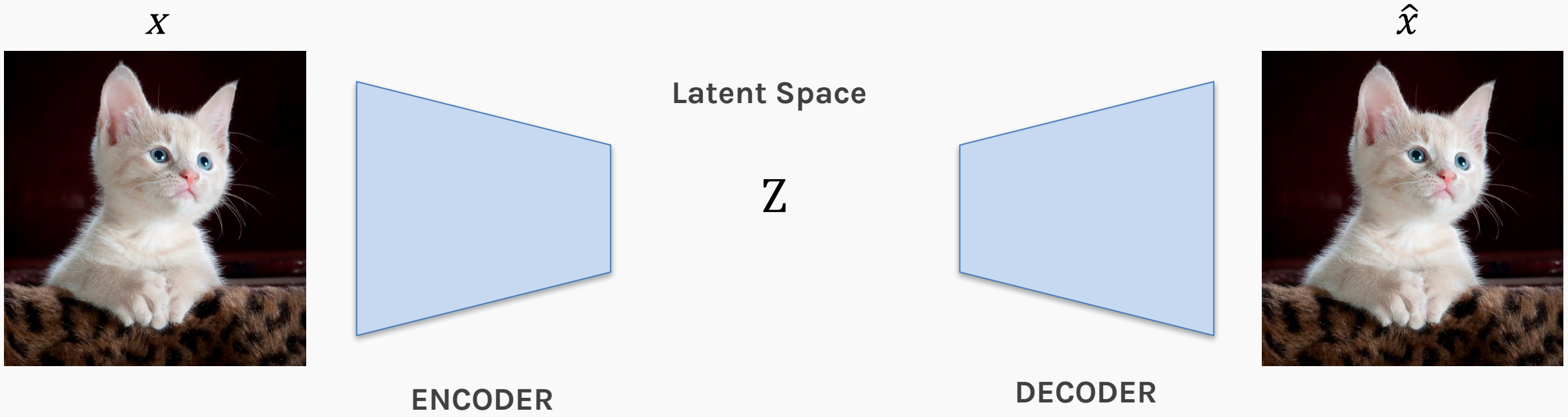
Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.





Autoencoder

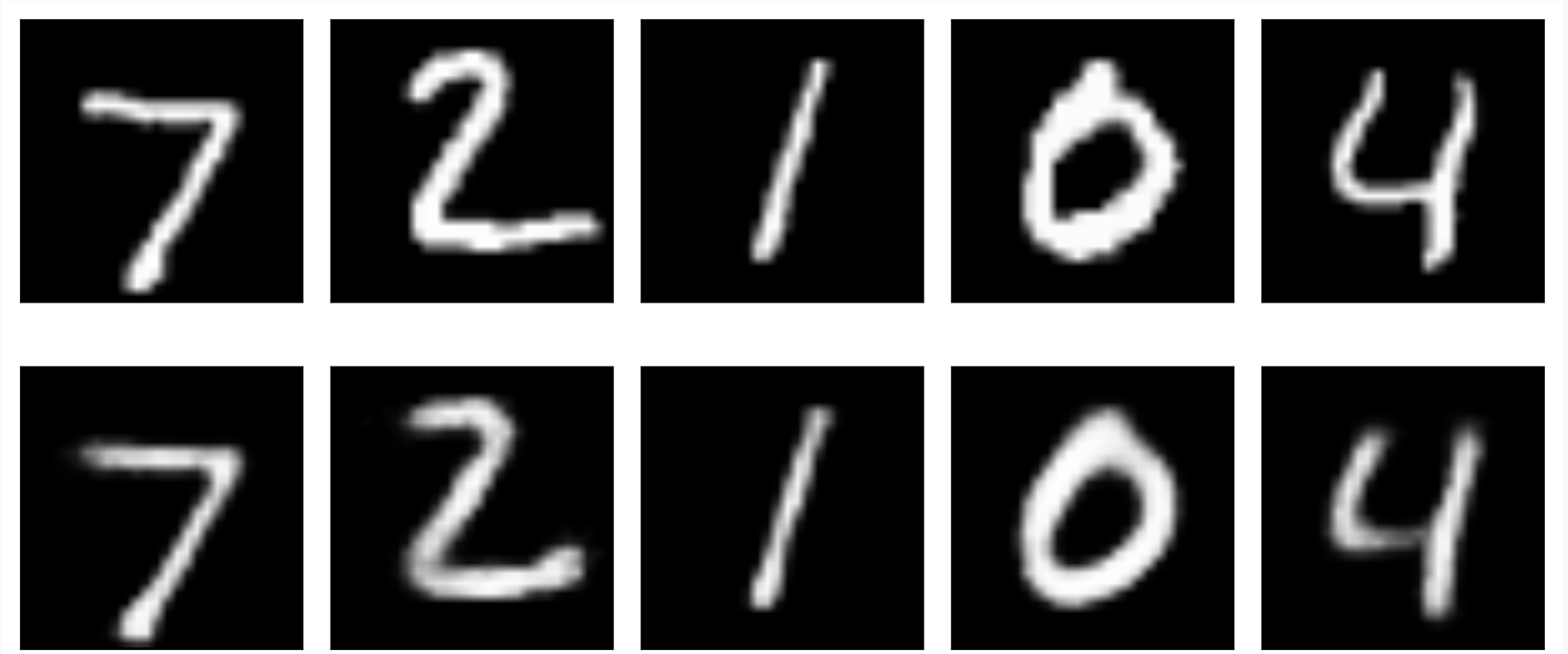
We train the two networks by minimizing the reconstruction loss function: $\mathcal{L} = \sum(x_i - \hat{x}_i)^2$



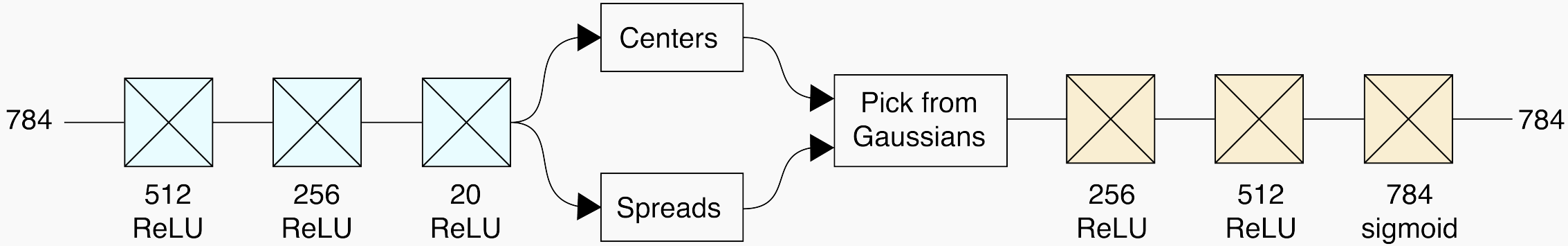
This is an autoencoder. It gets that name because it automatically finds the best way to encode the input so that the decoded version is as close as possible to the input.

In [30]:
```
# Build and train our first autoencoder
num_latent_vars = 20

modelAE1_s = Sequential()
modelAE1_s.add(Dense(num_latent_vars, input_dim=N_pixels, activation='relu'))
modelAE1_s.add(Dense(N_pixels, activation='sigmoid'))
modelAE1_s.compile(optimizer='adadelta', loss='binary_crossentropy')


weights_filename = "modelAE1-weights-sigmoid"
np.random.seed(52)
if not file_helper.load_model_weights(modelAE1_s, weights_filename):
    modelAE1_s.fit(X_train, X_train,
                   epochs=20, batch_size=128, shuffle=True,
                   verbose=2,
                   validation_data=(X_test, X_test))
    file_helper.save_model_weights(modelAE1_s, weights_filename)
```

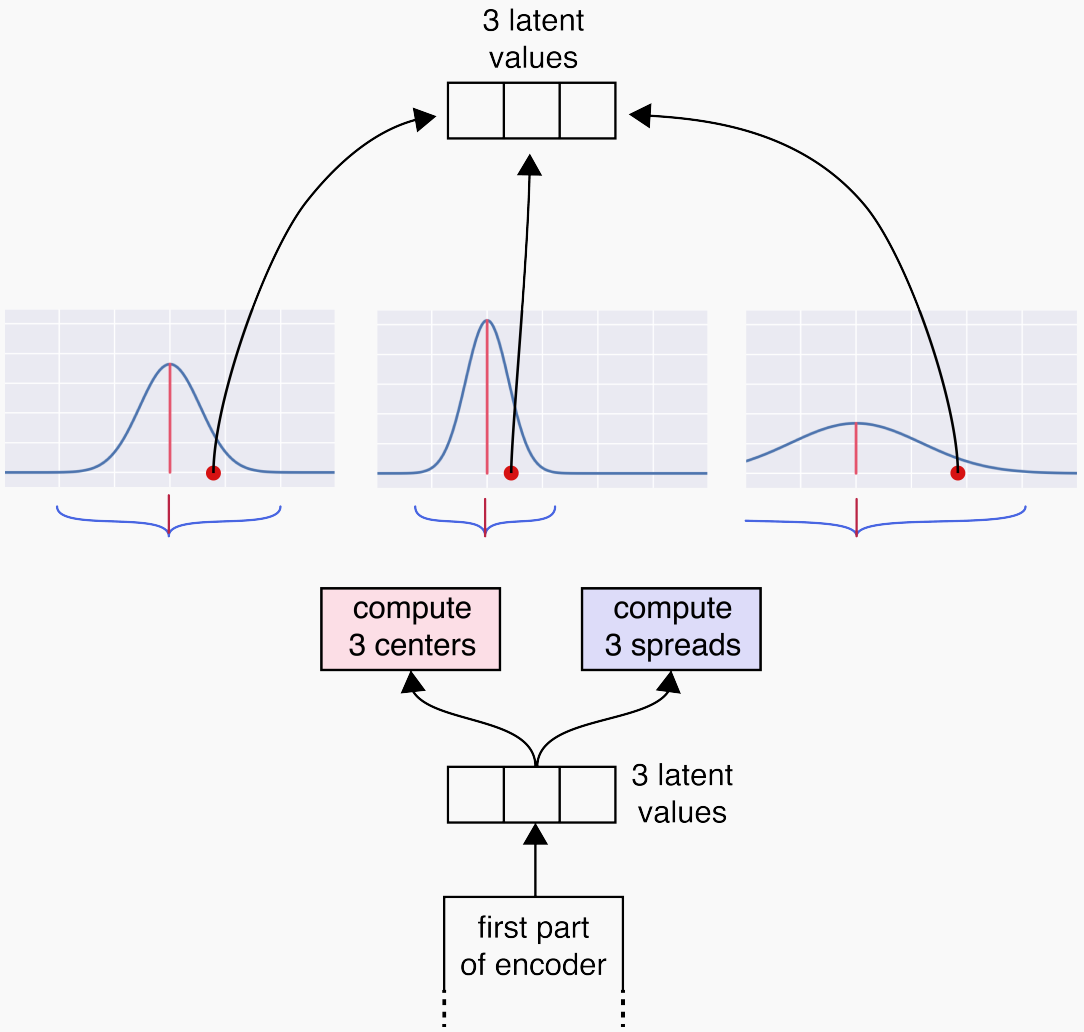
In [9]:
```
predictions1 = modelAE1_s.predict(X_test)
draw_predictions_set(predictions1, 'FCC1-s')
```



Variational Autoencoders



Variational Autoencoders

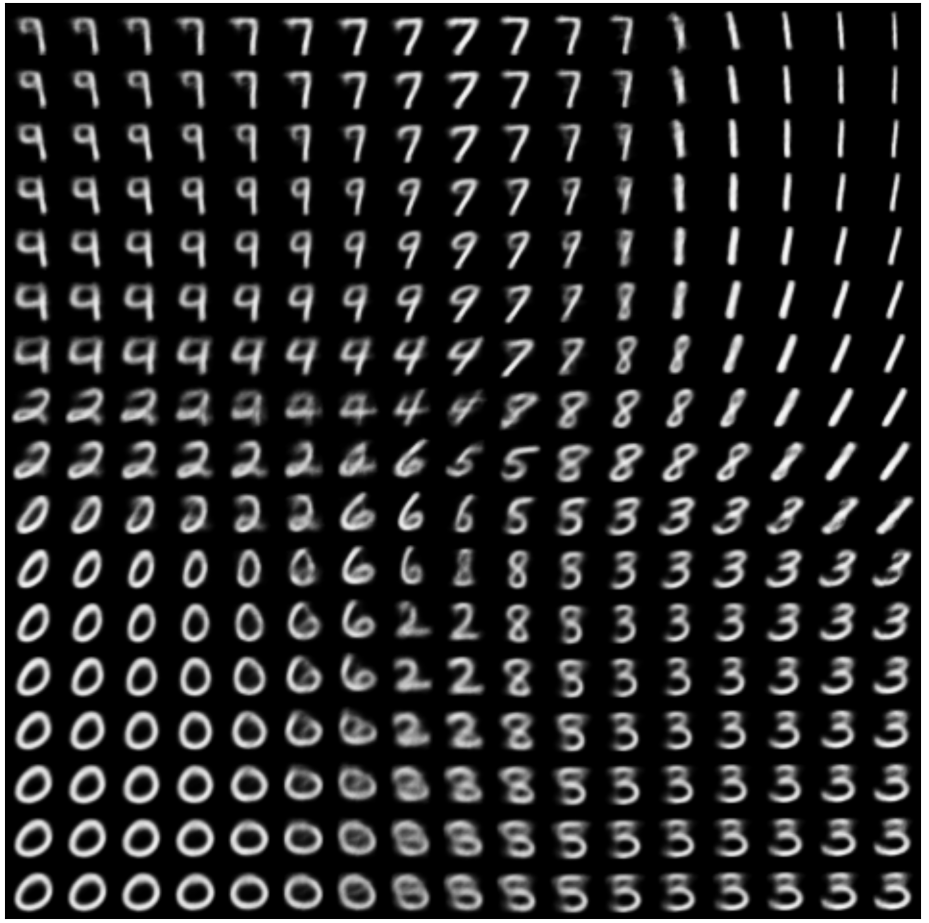


Example code (could help a lot for HW7)

https://github.com/Harvard-IACS/2019-computefest/tree/master/Wednesday/auto_encoder

Generating Data

We saw how to generate new data with a VAE in Lecture 19.



Outline

Review of AE and VAE

GANS

- Motivation
- Formalism
- Training

Game Theory, minmax

Challenges:

- Big Samples
- Modal collapse

Generating Data

In this lecture 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.

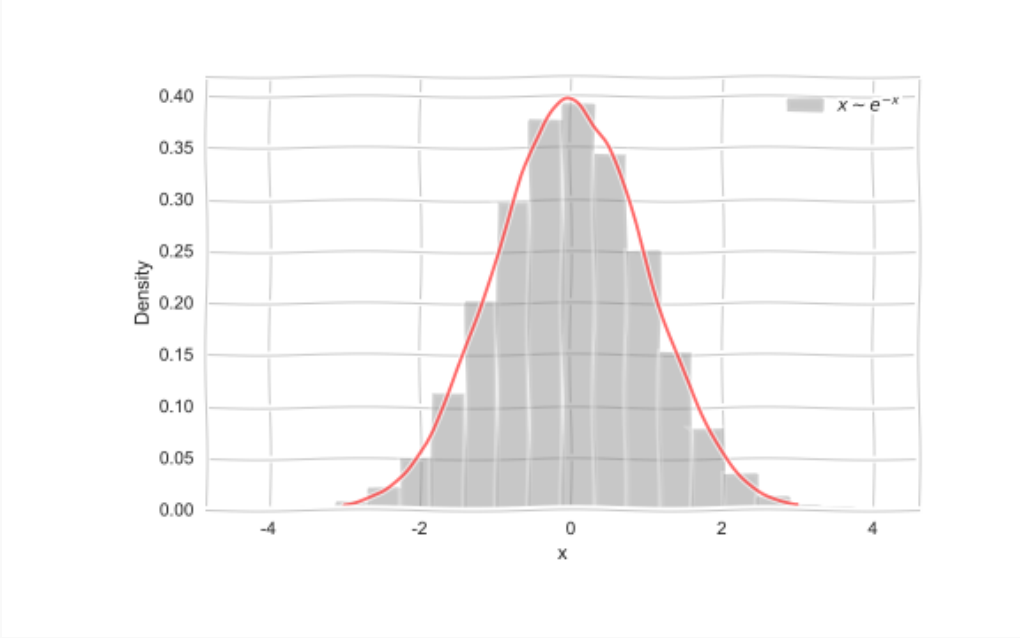
Generative models

Imagine we want to generate data from a distribution,

$$x \sim p(x)$$

e.g.

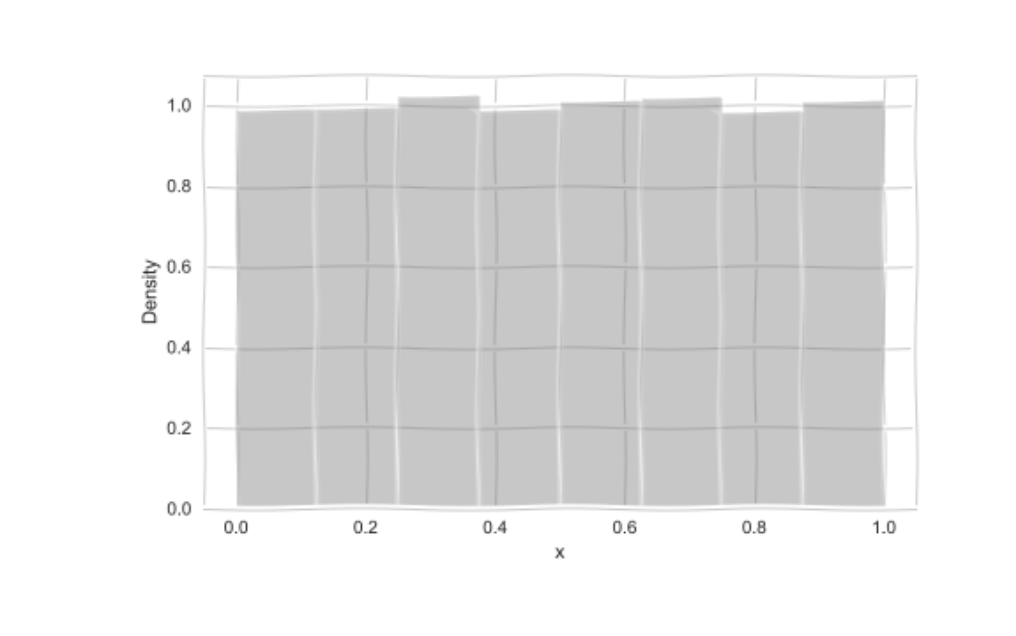
$$x \sim \mathcal{N}(\mu, \sigma)$$



Generative models

But how do we generate such samples?

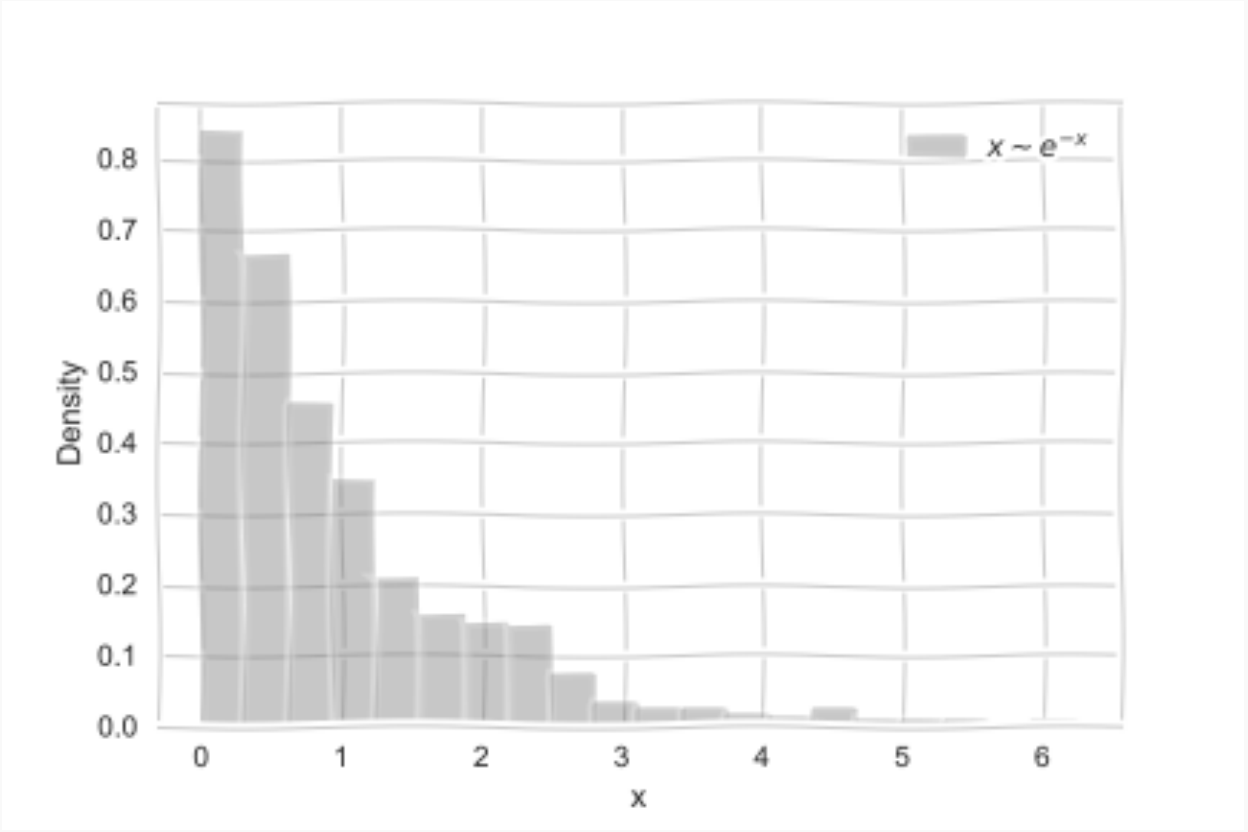
$$z \sim \text{Unif}(0, 1)$$



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:

$$x \sim p(x)$$

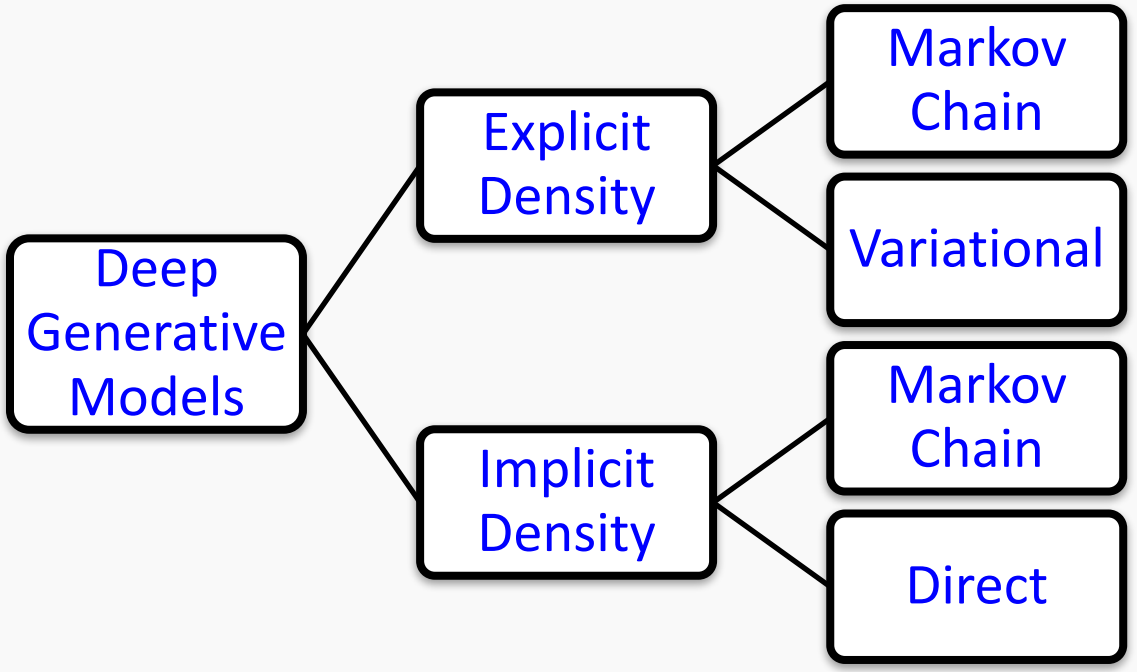
where in general f is some complicated function.

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

Generative models

We would like to construct our generative model, which we would like to train to generate lightcurves like these from **scratch**.

A generative model in this case could be one large neural network that outputs lightcurves: ***samples from the model***.



Boltzmann Machines
Deep Belief Networks

Variational Autoencoders (VAE)

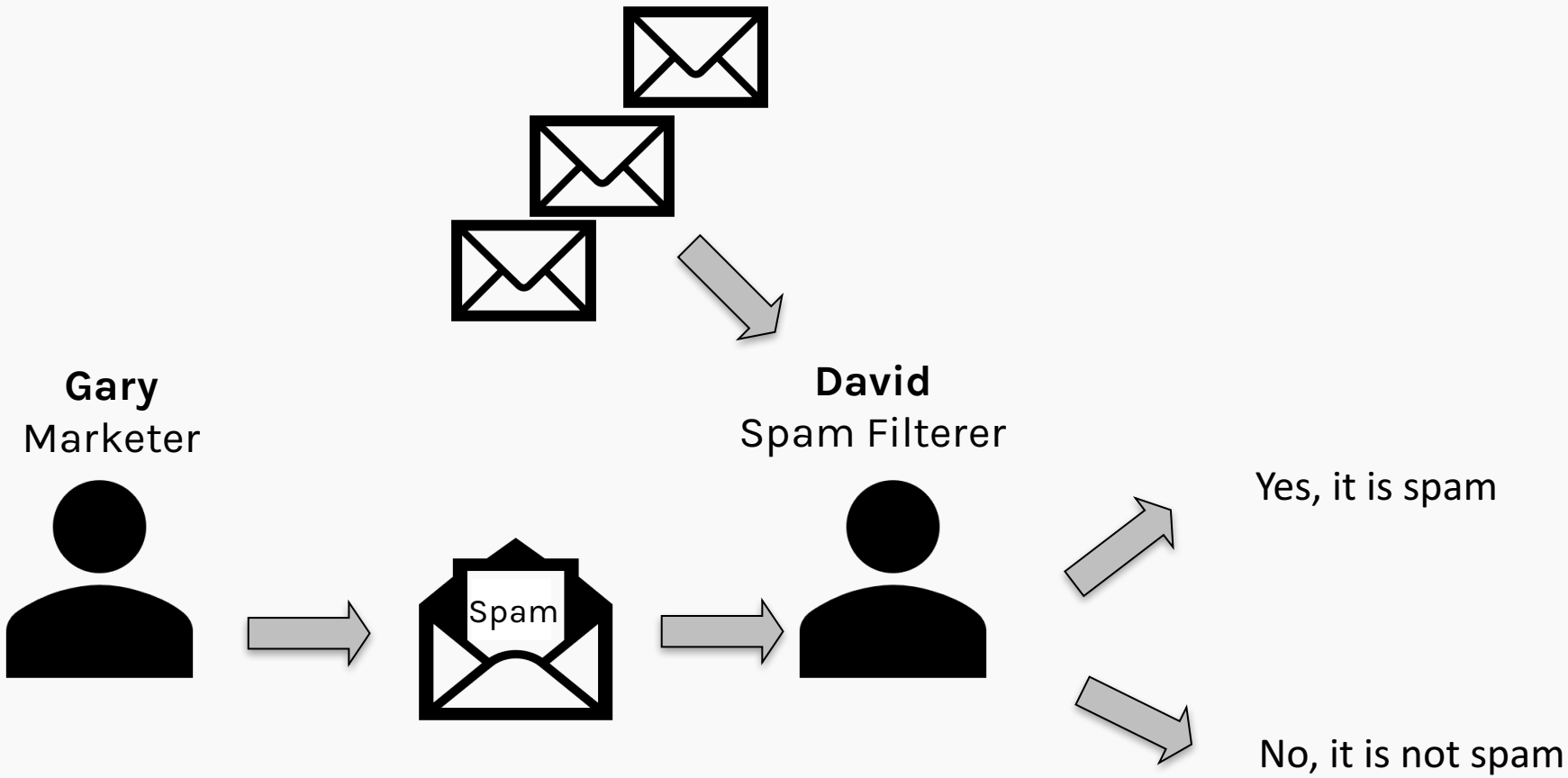
Generative Adversarial Networks (GAN)

Generative Moment Matching Networks (GMM)

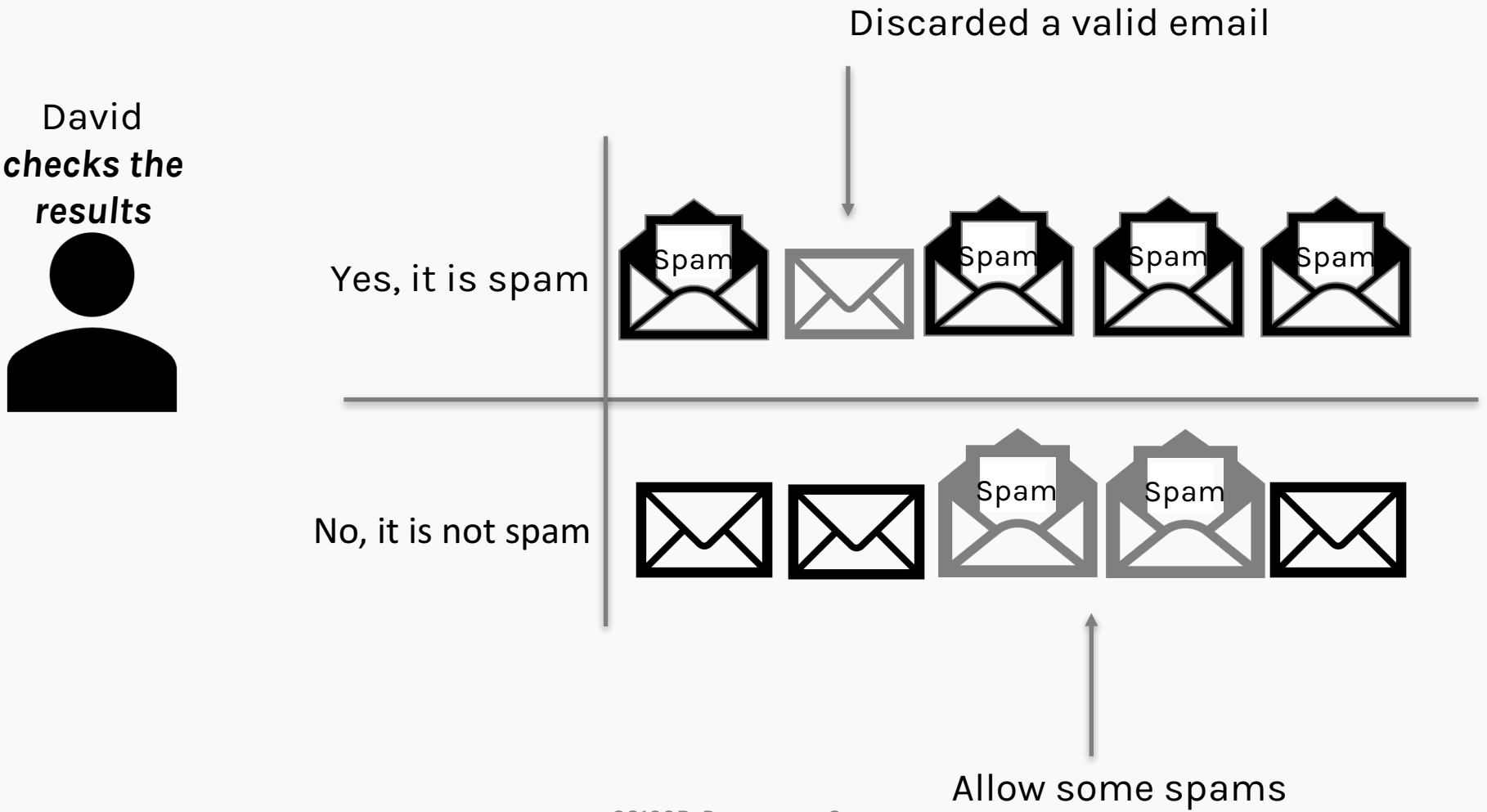
Adversarial

2 Athletes: Messi and Ronaldo. Adversaries!

Generative Adversarial Networks (GANs)

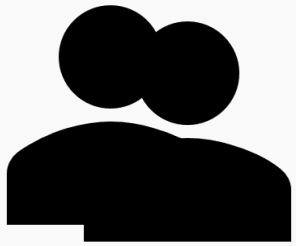


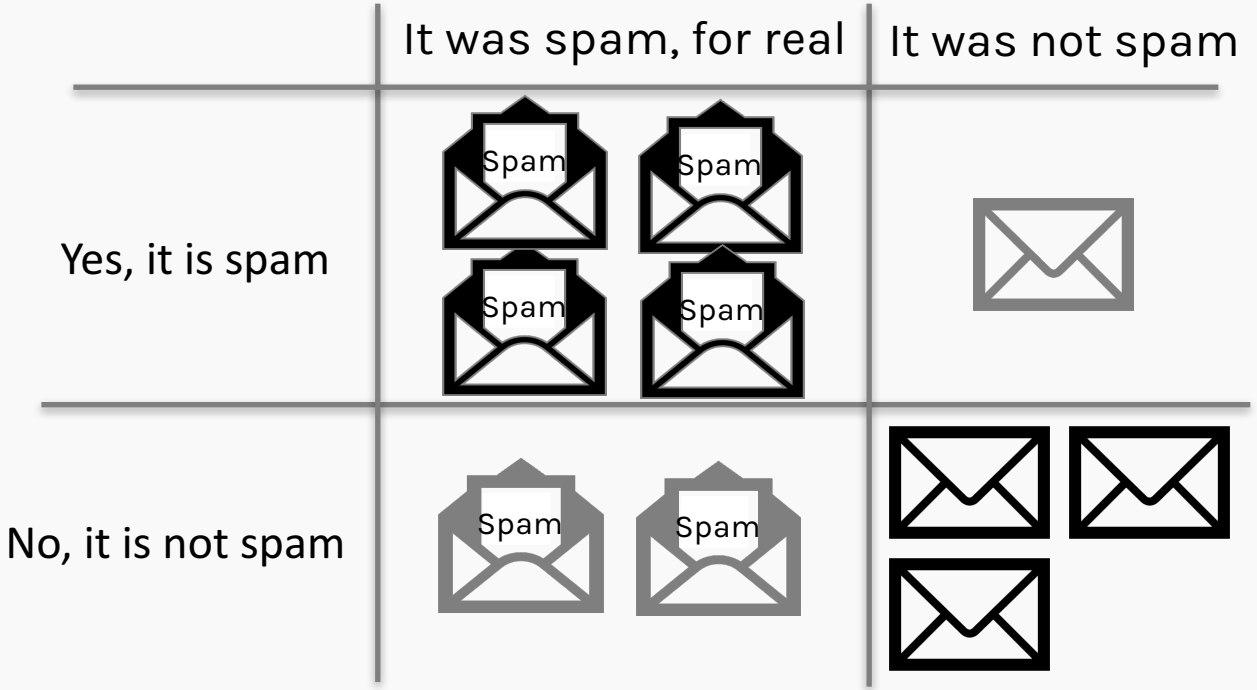
Generative Adversarial Networks (GANs)



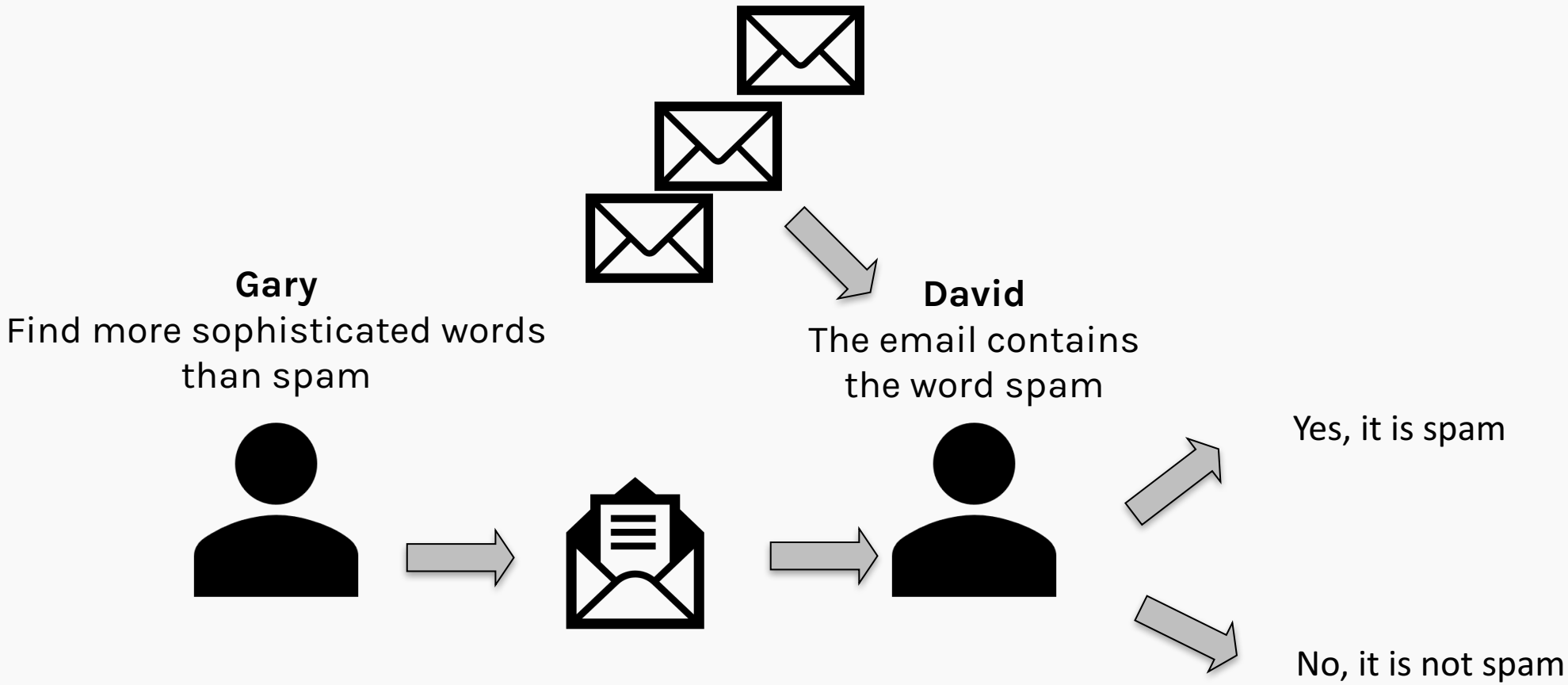
Generative Adversarial Networks (GANs)

David and Gary
learned what went wrong

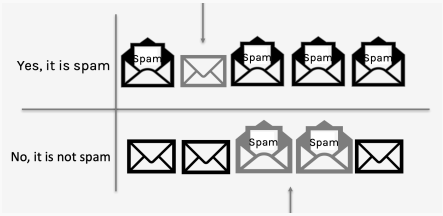




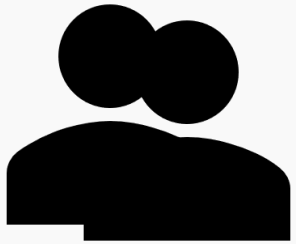
Generative Adversarial Networks (GANs)



Generative Adversarial Networks (GANs)



David and Gary
learned what went wrong



Discarded a valid email

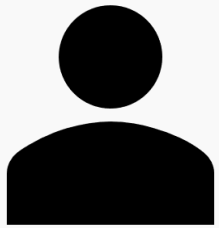
Yes, it is spam

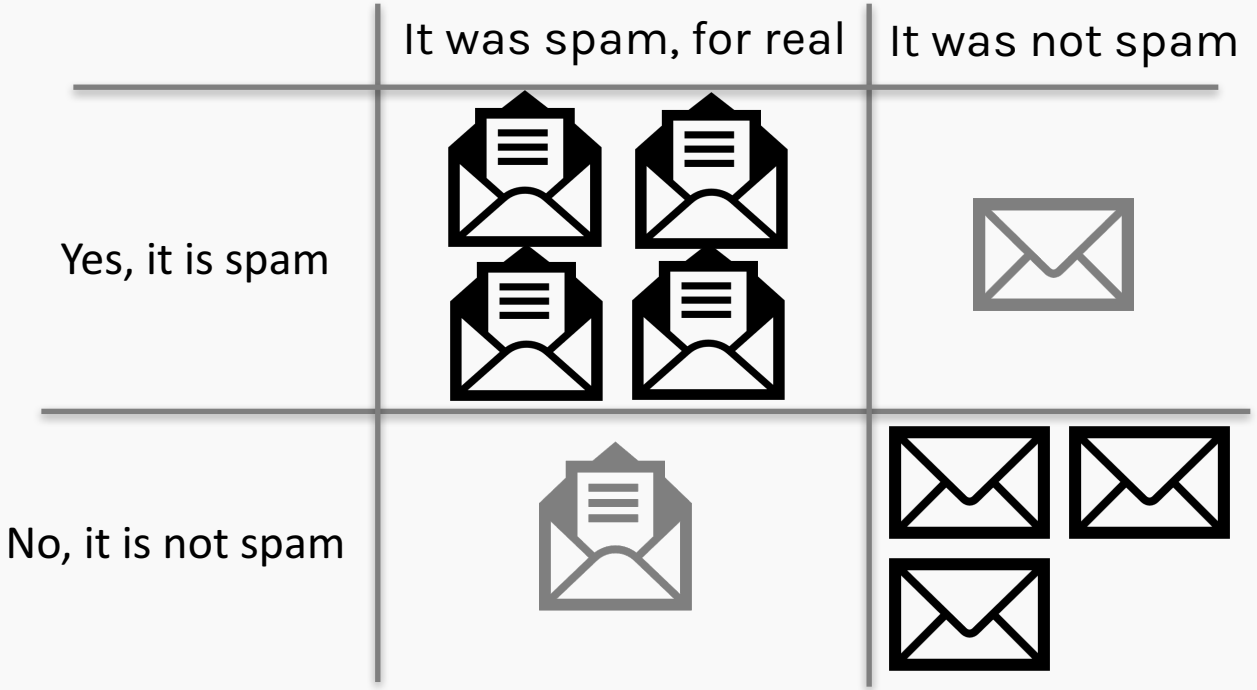
No, it is not spam

Allow fewer spams

Generative Adversarial Networks (GANs)

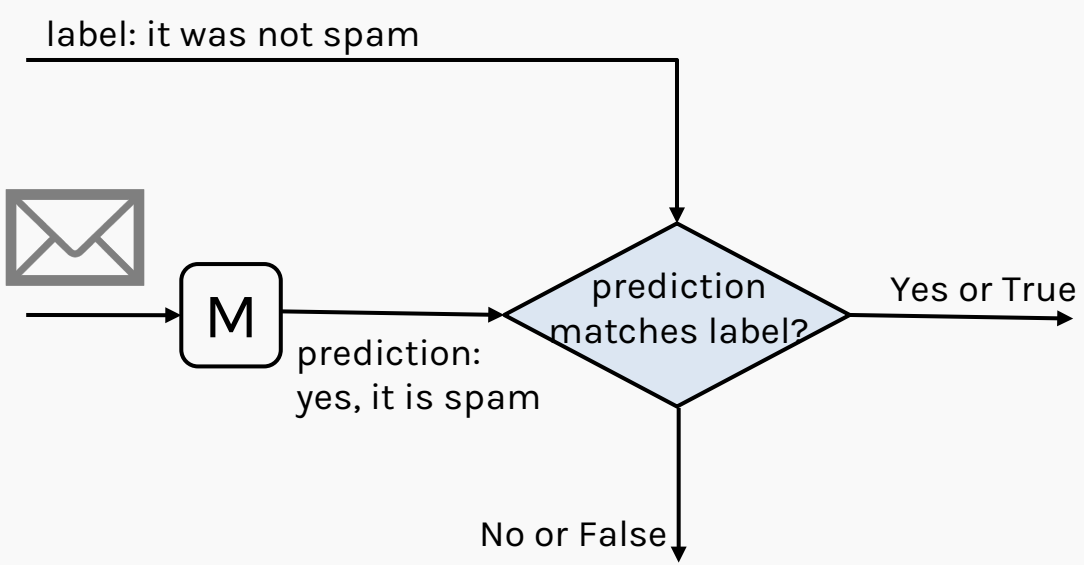
David
learned what went wrong

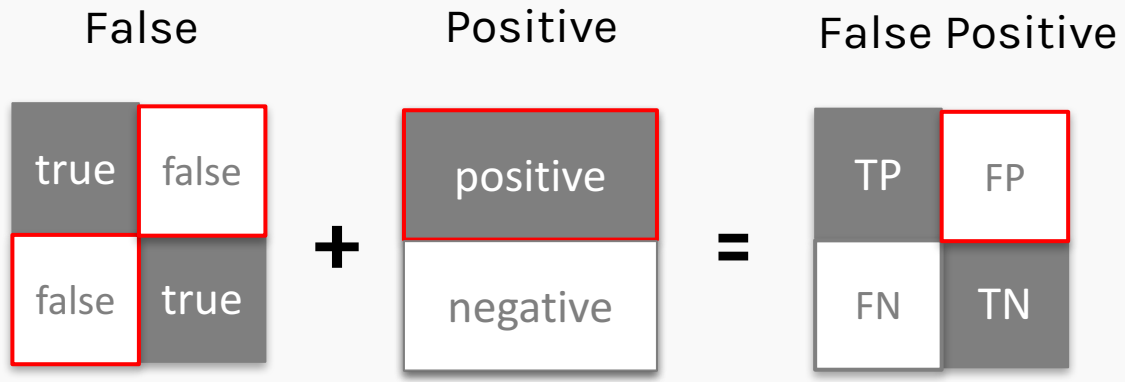




Generative Adversarial Networks (GANs)

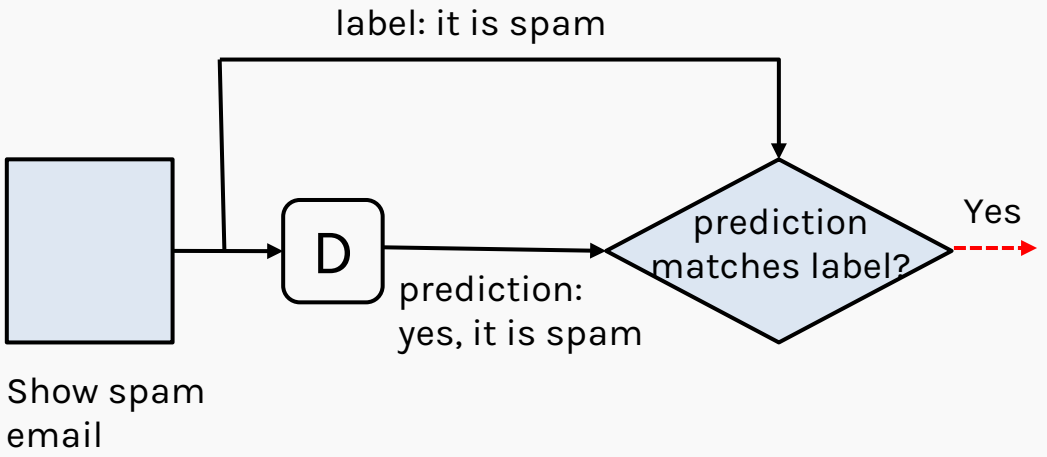
Understanding confusion matrix through an example



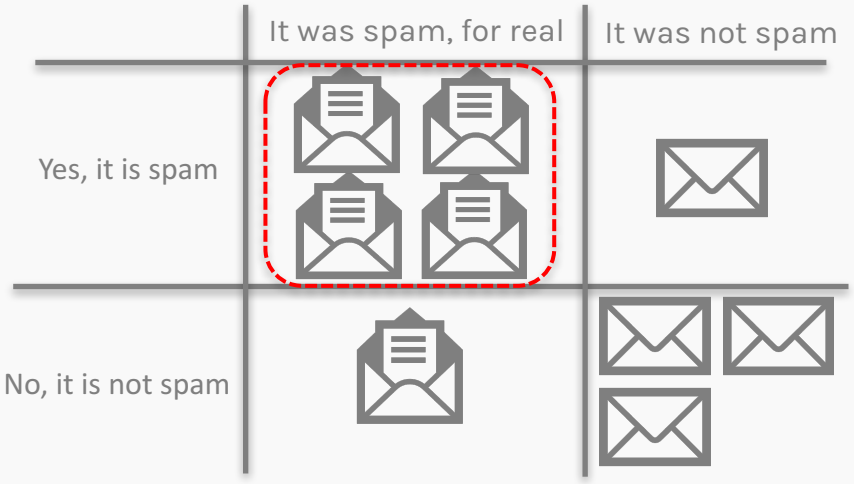


SPAM: POSITIVE
TRUE/FALSE: If prediction and true match do not match
POSITIVE/NEGATIVE: Prediction class

Generative Adversarial Networks (GAN)

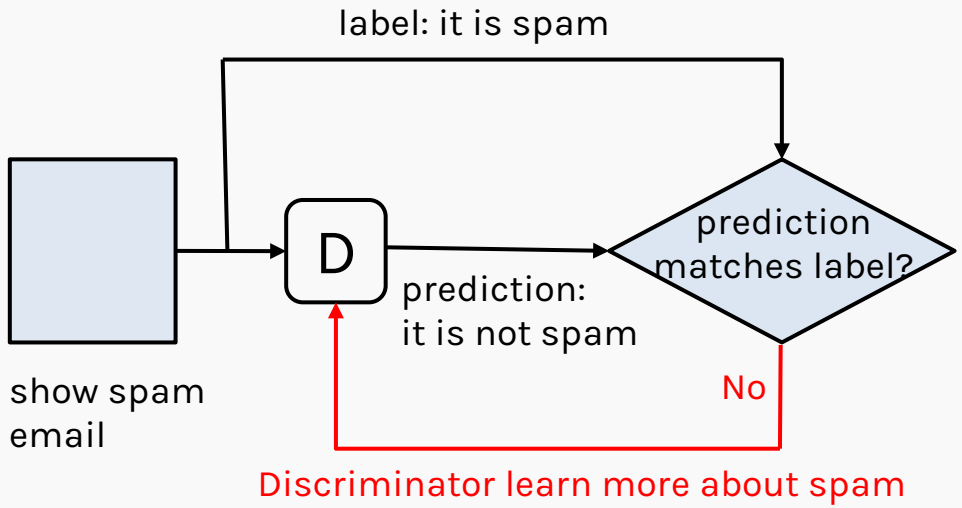


No action for discriminator. **Generator must do better.**

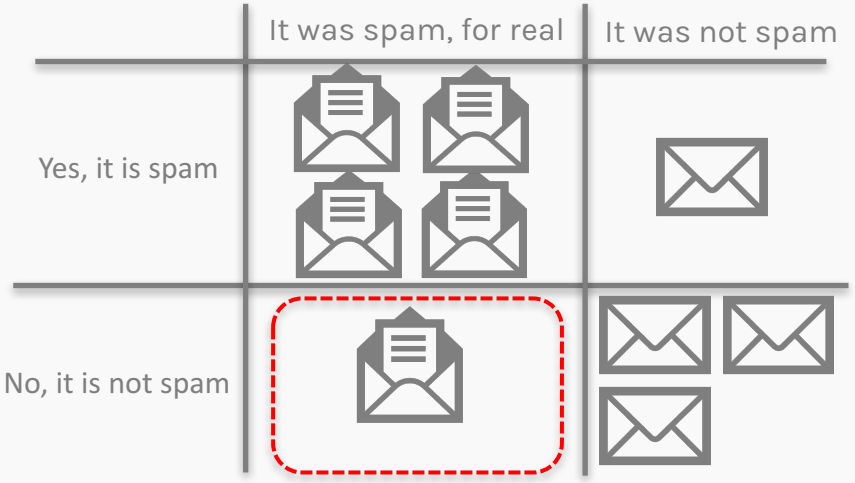


True positive (TP): the discriminator see a spam and predicts correctly. No need for further actions for discriminator. Generator must do a better job.

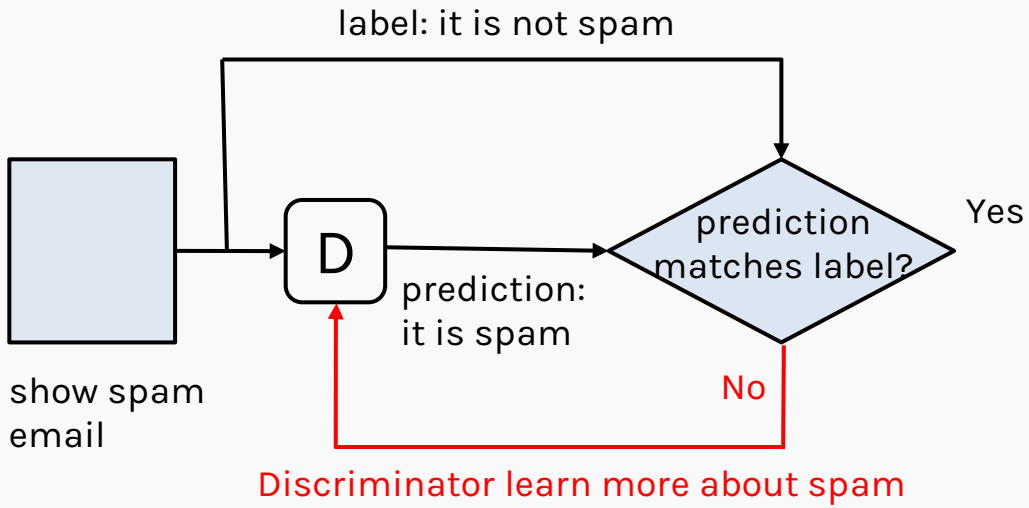
Generative Adversarial Networks (GANs)



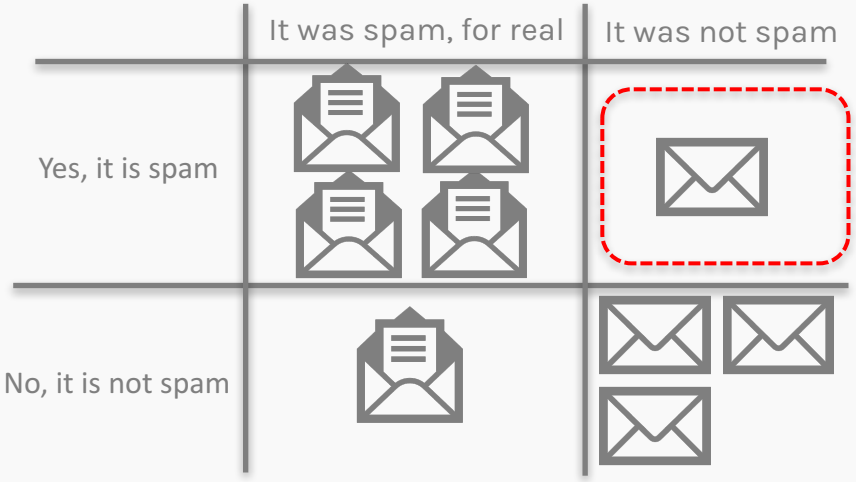
False Negative (FN): the discriminator see an email and predict it as spam even though it is not. The discriminator learn more



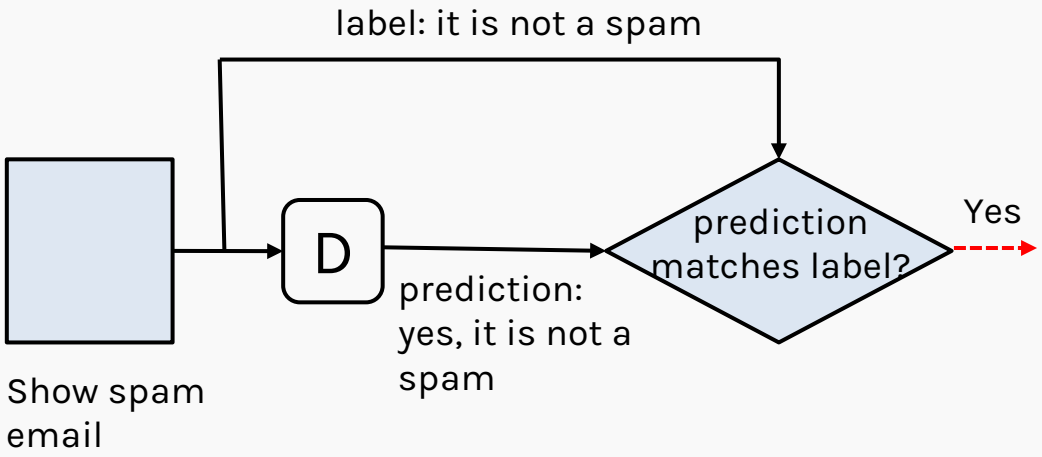
Generative Adversarial Networks (GANs)



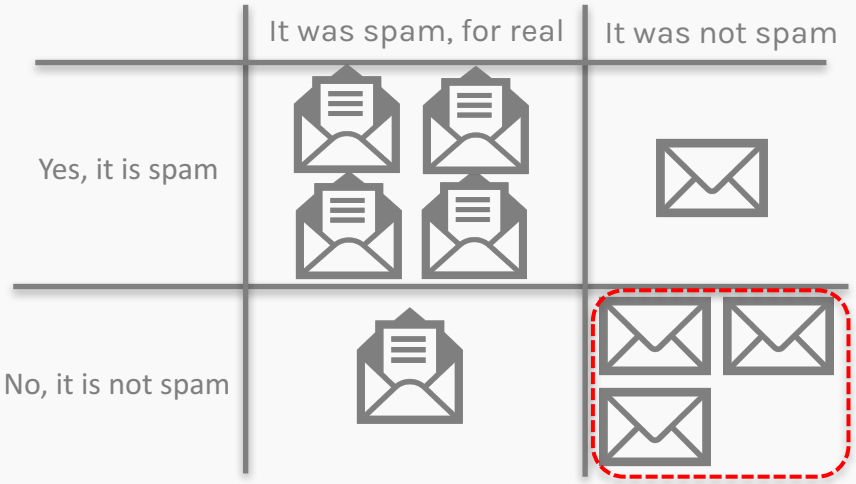
False positive (FP): generator try to fool discriminator and the discriminator fails. The generator succeeded and the generator is forced to improve.



Generative Adversarial Networks (GANs)



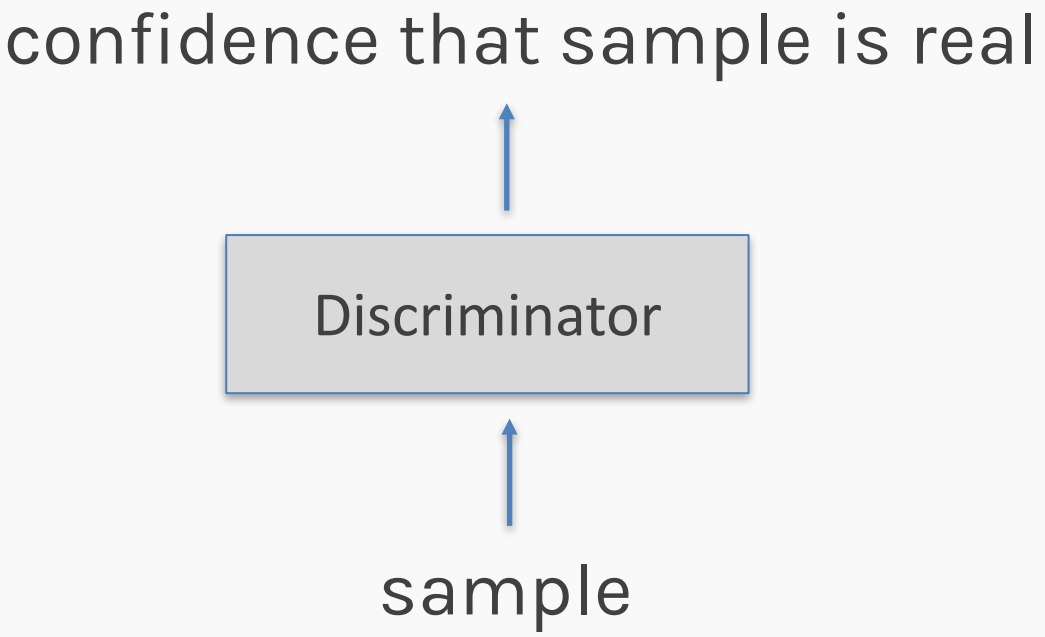
True negative (TN): generator try to fool discriminator, however the discriminator predict correctly. The generator learn what was wrong and try something else.



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.

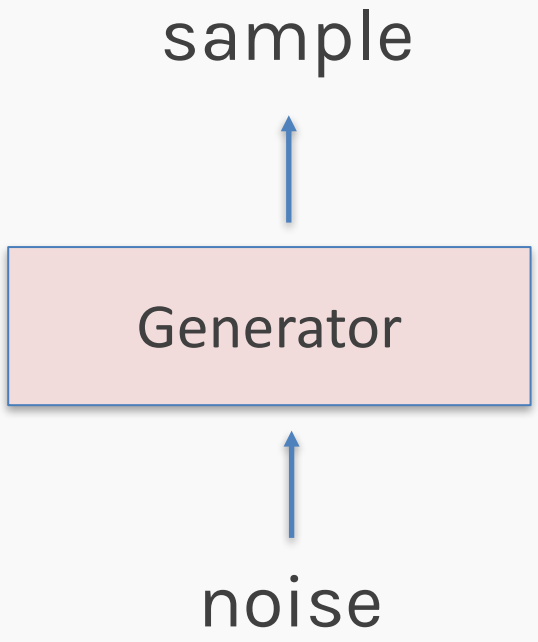
There are not many restrictions on what the discriminator is.



The Generator

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

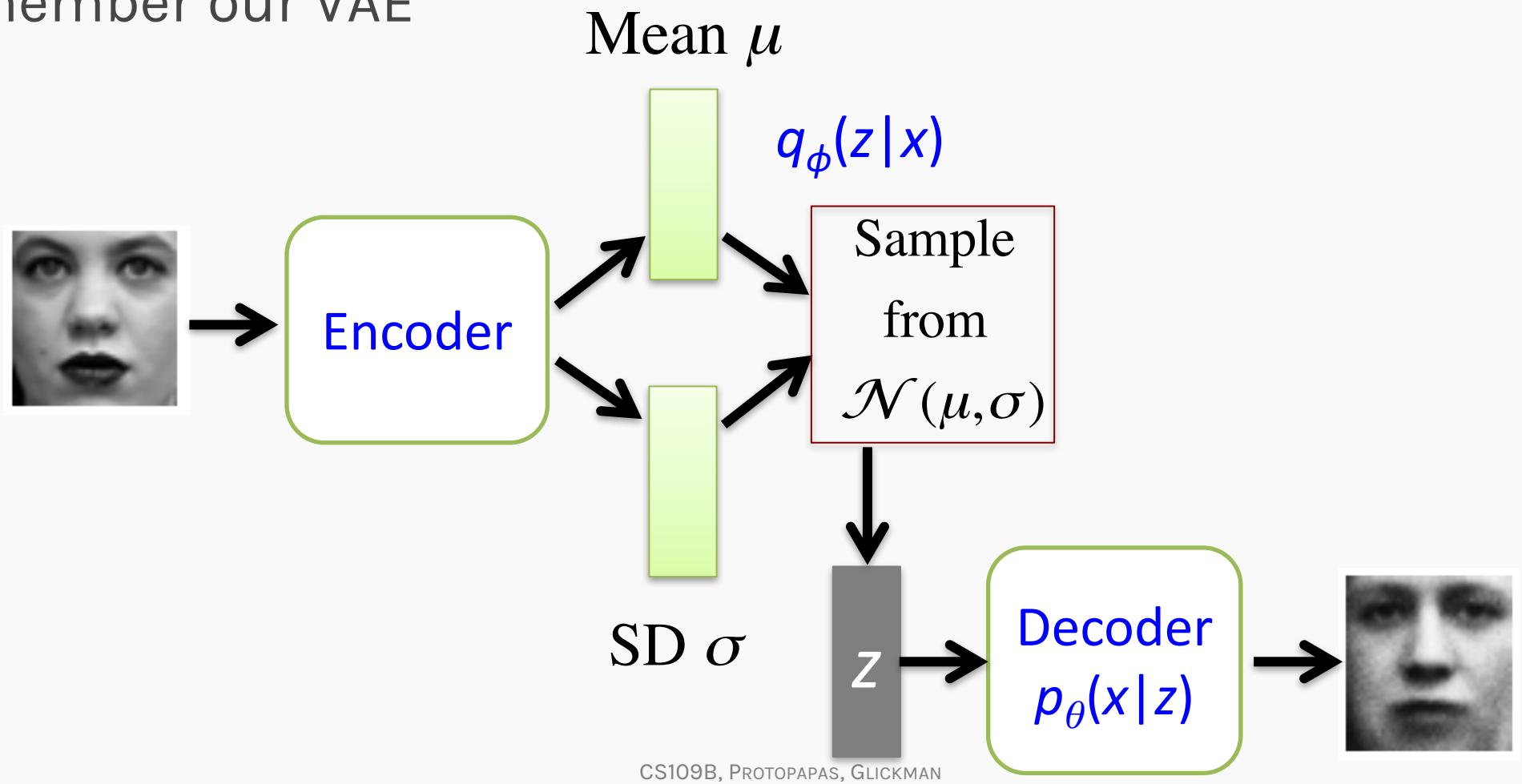
If we build our generator to be deterministic, then the same input will always produce the same output. In that sense we can think of the input values as latent variables. But here the latent variables weren't discovered by analyzing the input, as they were for the VAE.



The Generator

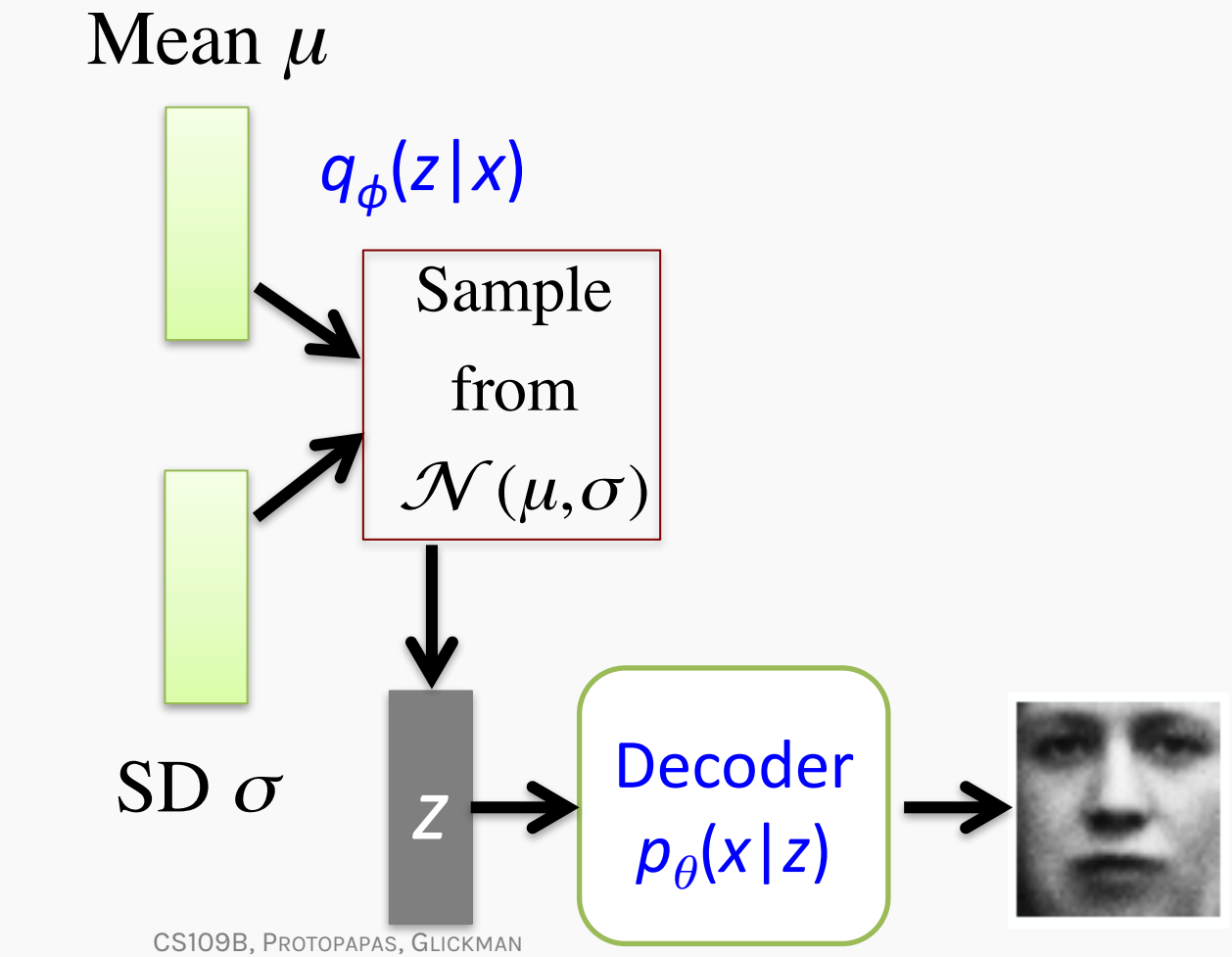
Is it really noise?

Remember our VAE



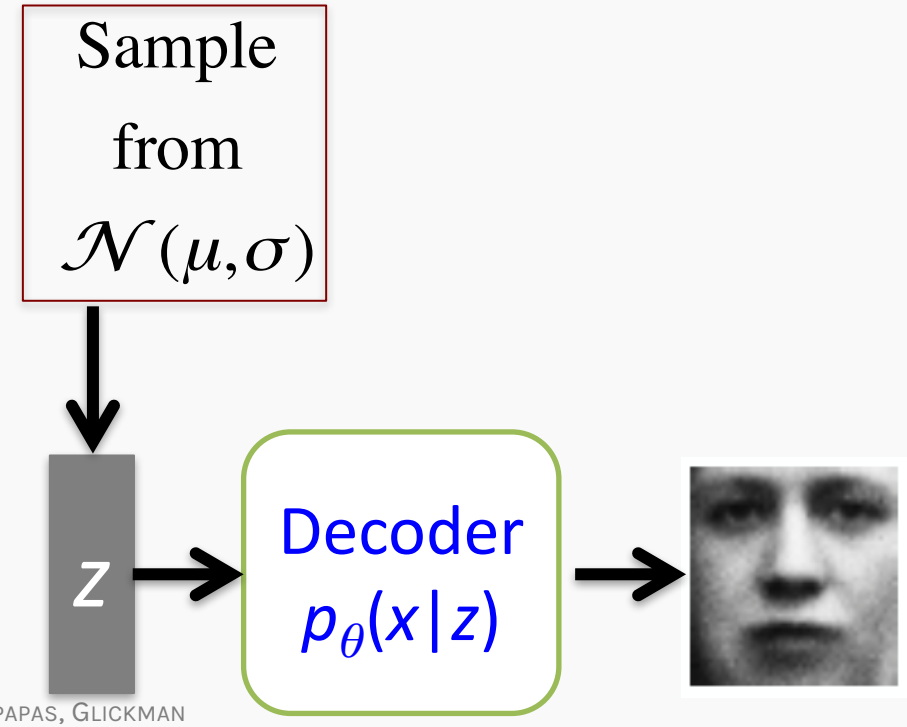
The Generator

Is it really noise? Remember our Variational Autoencoder.



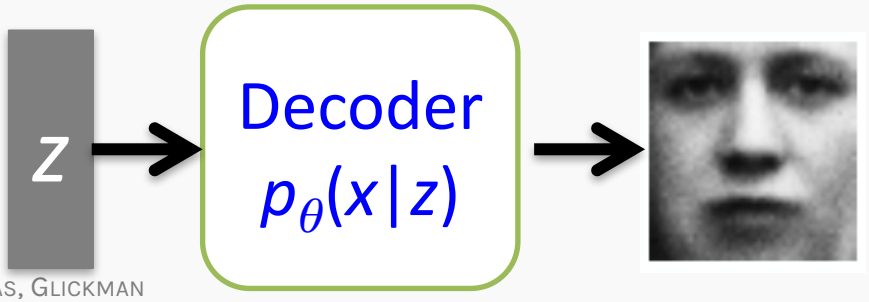
The Generator

Is it really noise? Remember our Variational Autoencoder.



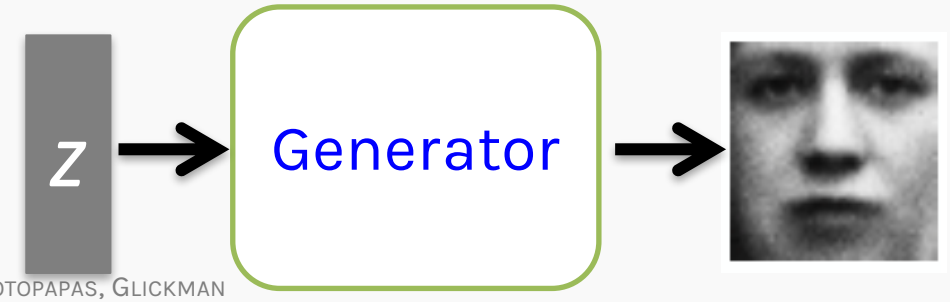
The Generator

Is it really noise? Remember our Variational Autoencoder.



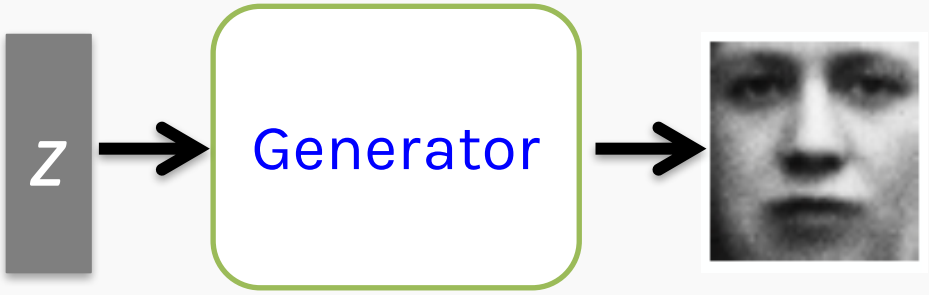
The Generator

Is it really noise? Remember our Variational Autoencoder.



The Generator

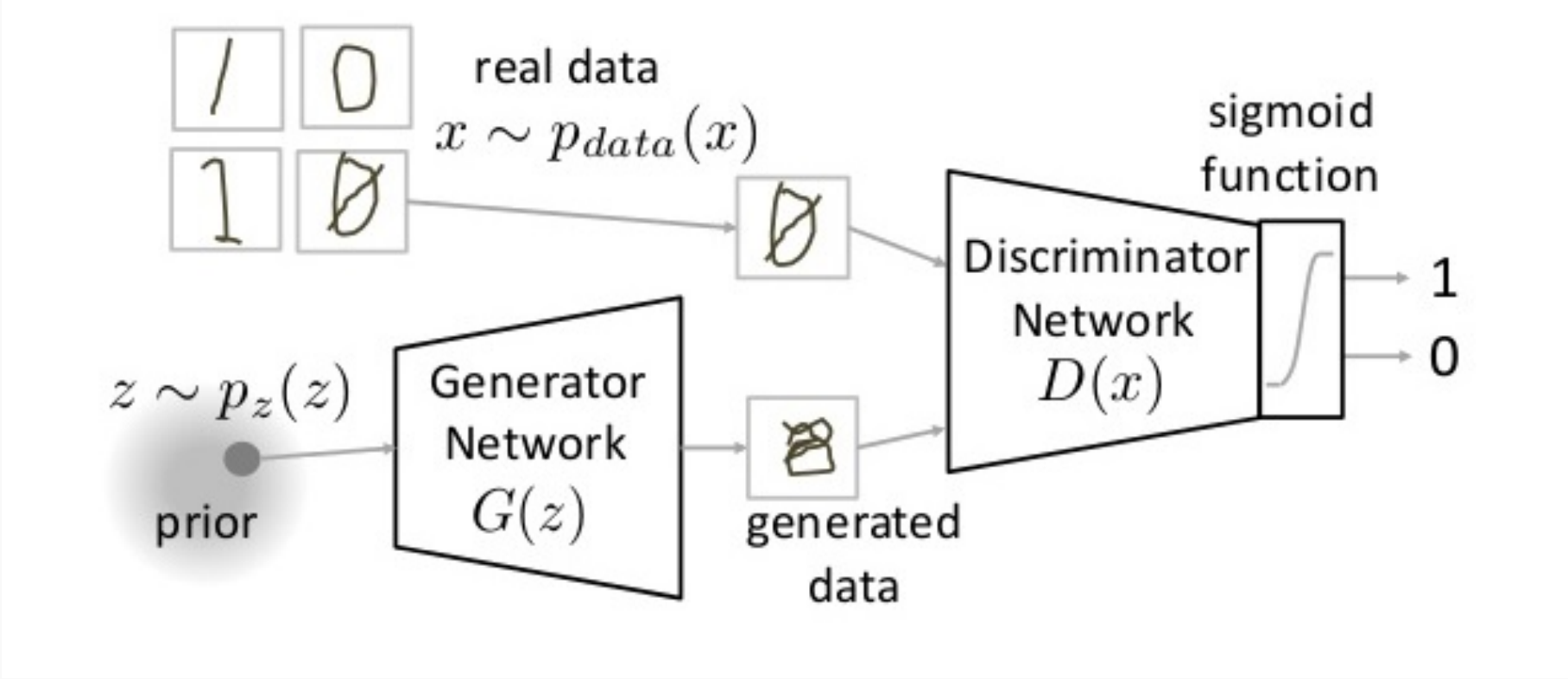
So the random noise is not “random” but represents (an image in the example below) in the “latent” space.



Learning

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


real data
$x \sim p_{data}(x)$
sigmoid
function
Discriminator
Network
$D(x)$
1
0
$z \sim p_z(z)$
prior
Generator
Network
$G(z)$
generated
data

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!

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

Training GANs

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

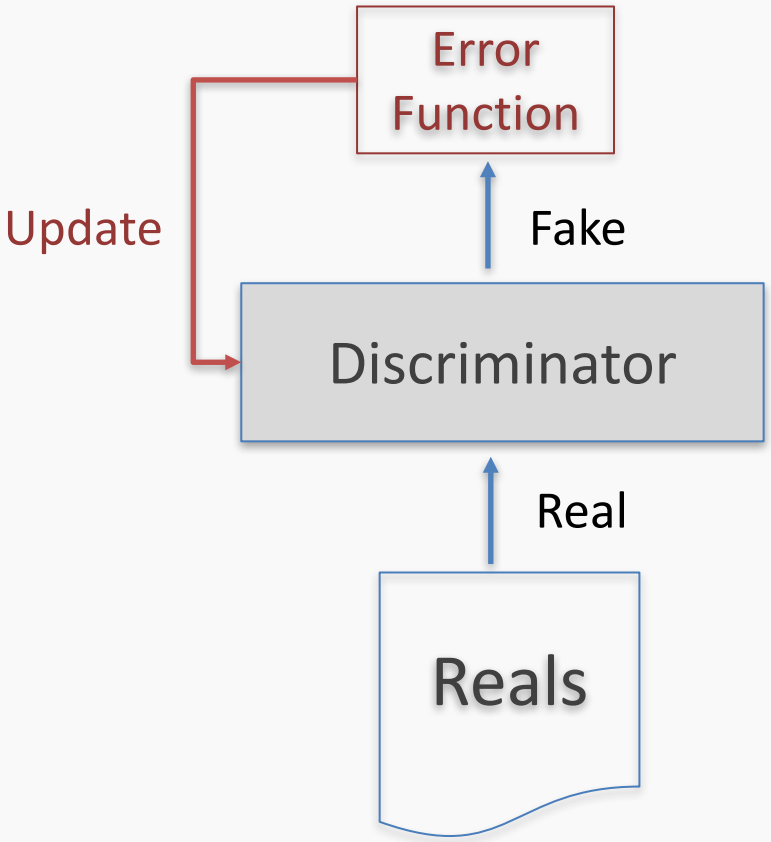
Update *G* using backprop

Update *D* using backprop

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

D:4 G:1 (this was the norm a year ago. So much has changed)

Training the GAN

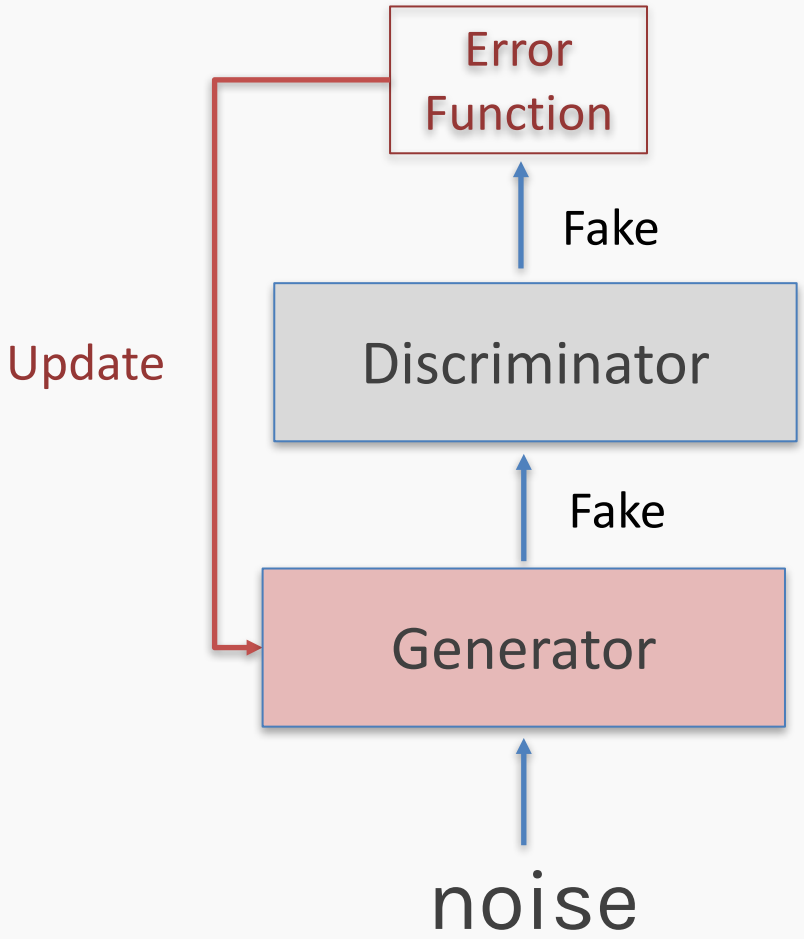


False negative (I: Real/D: Fake):

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.

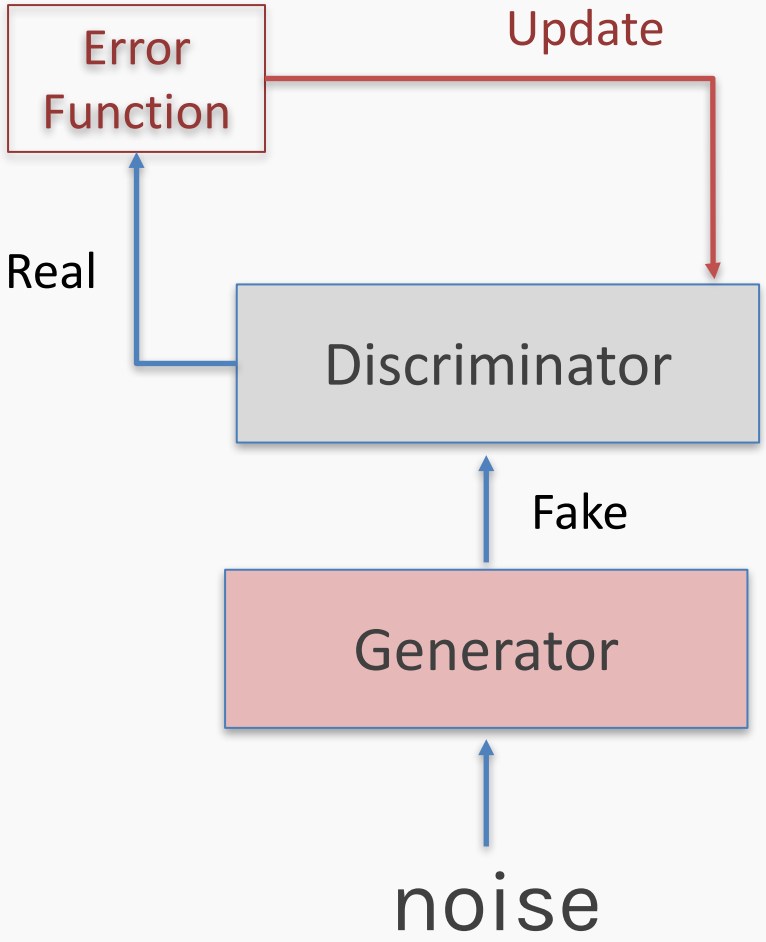
Training the GAN



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 (I:Fake/D:Real):

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

Min-max Cost

Discriminator seeks to predict 1 on training samples

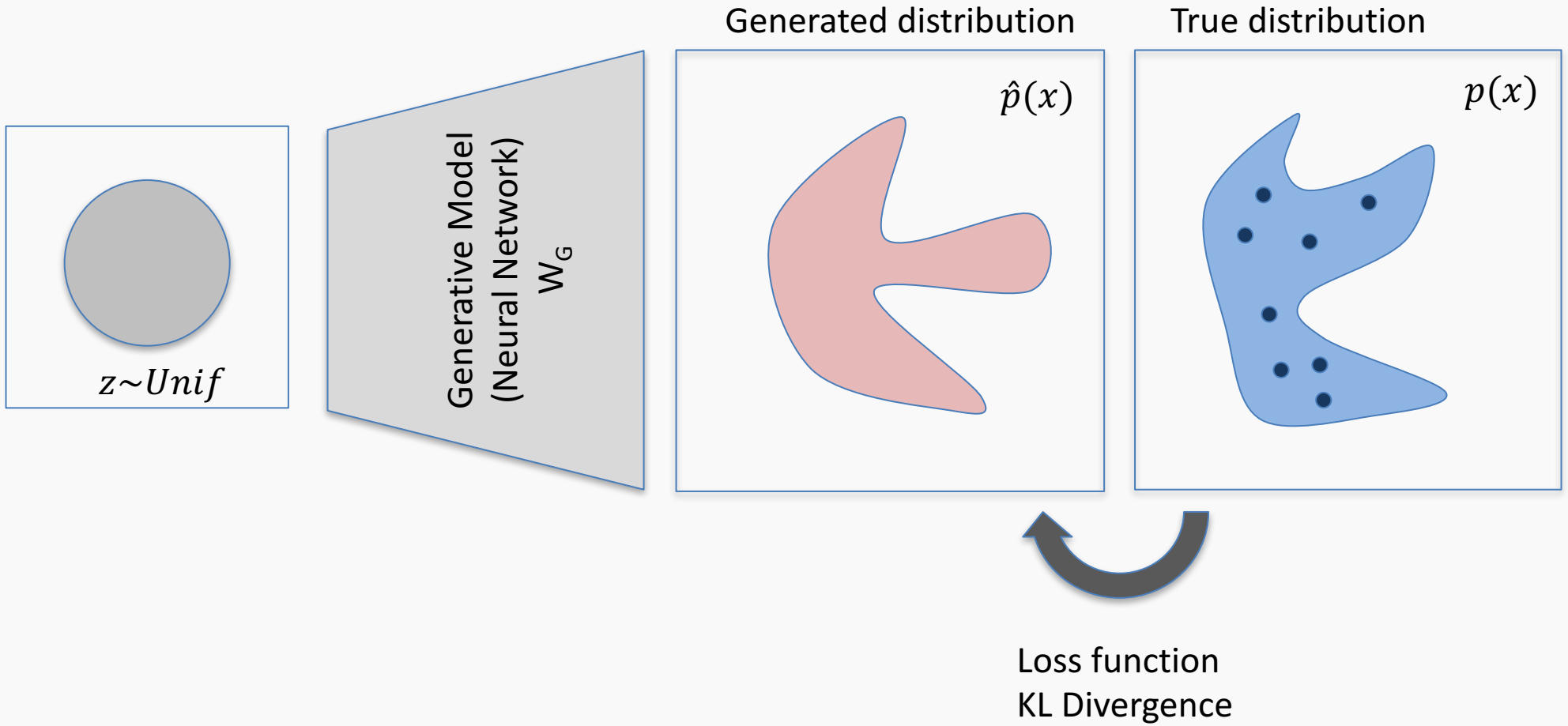
Discriminator wants to predict 0 on samples generated by *G*

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

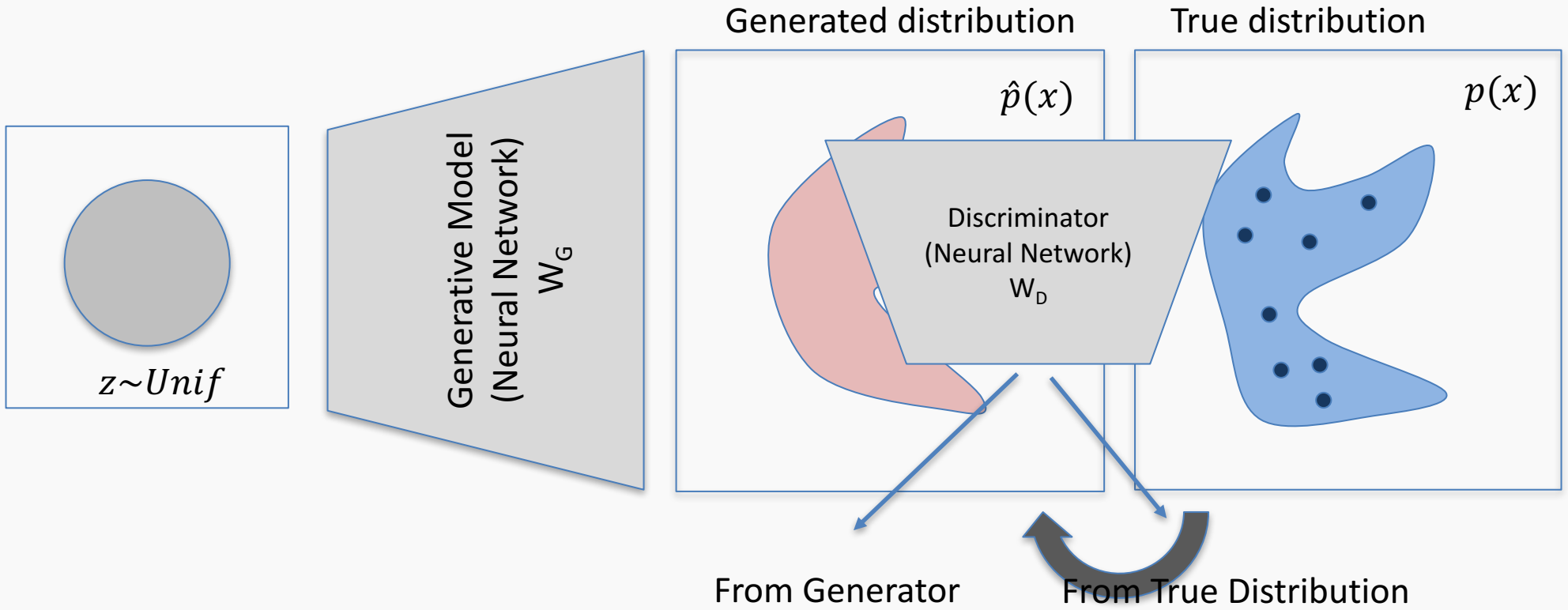
Generator wants *D* to not distinguish between original and generated samples!

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

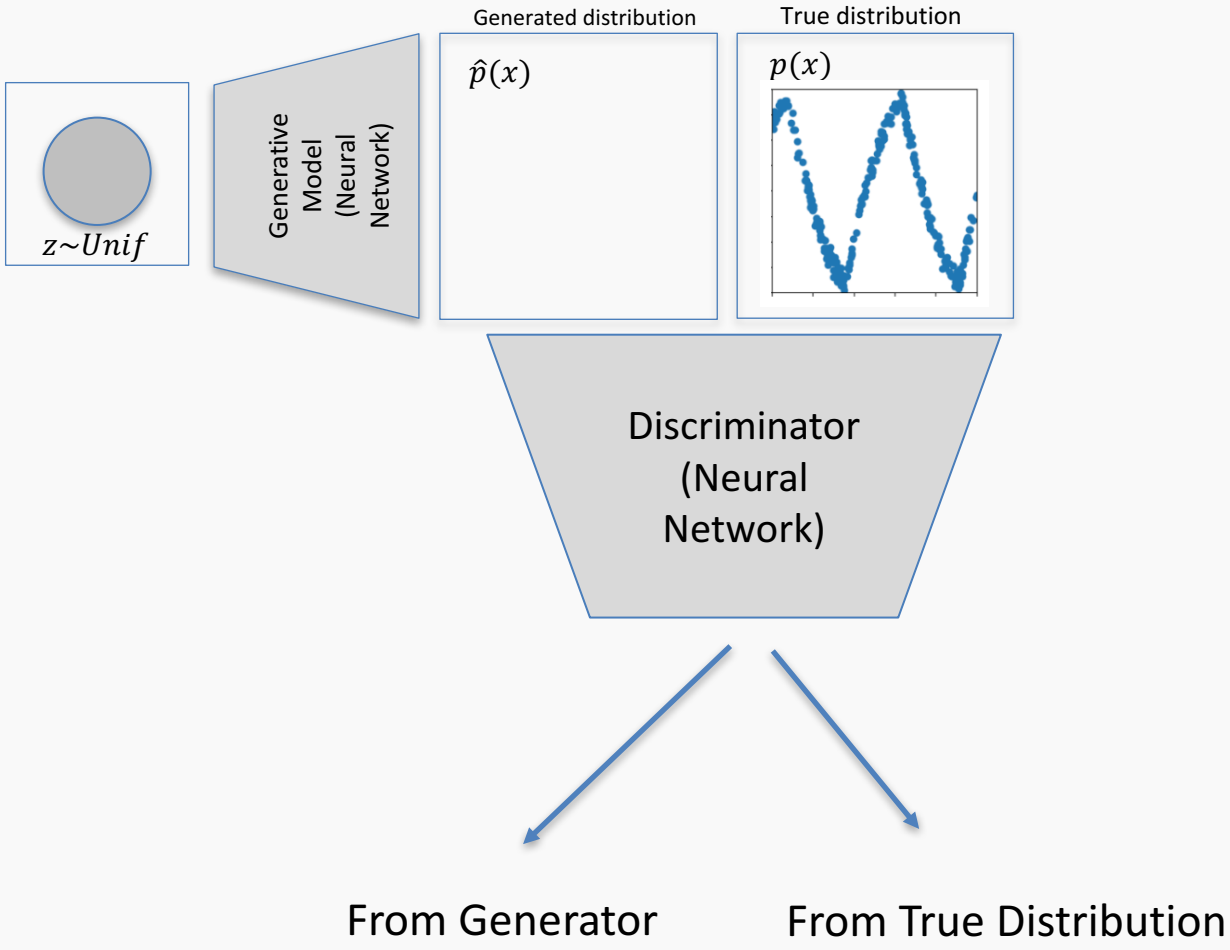
Generative Models



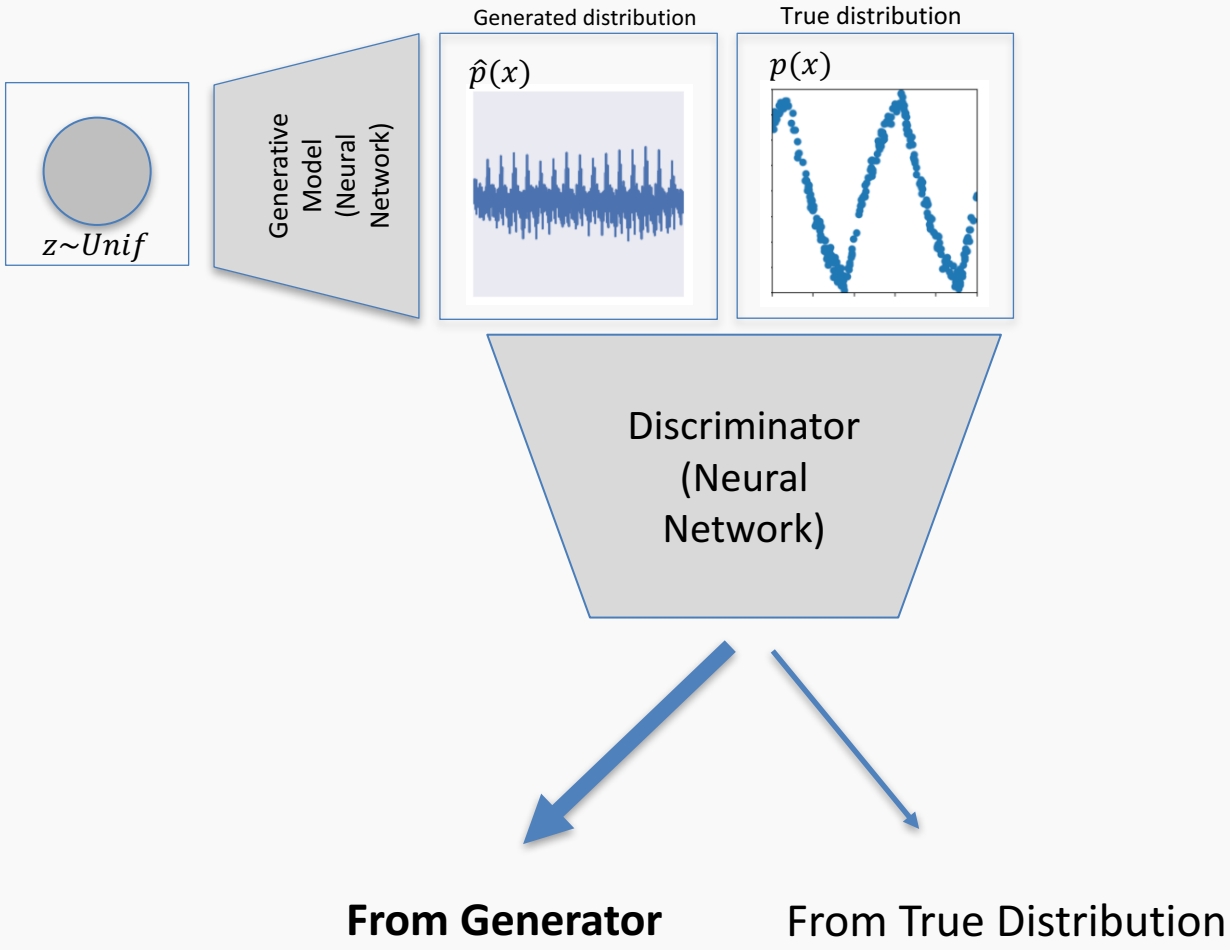
Generative Adversarial Networks



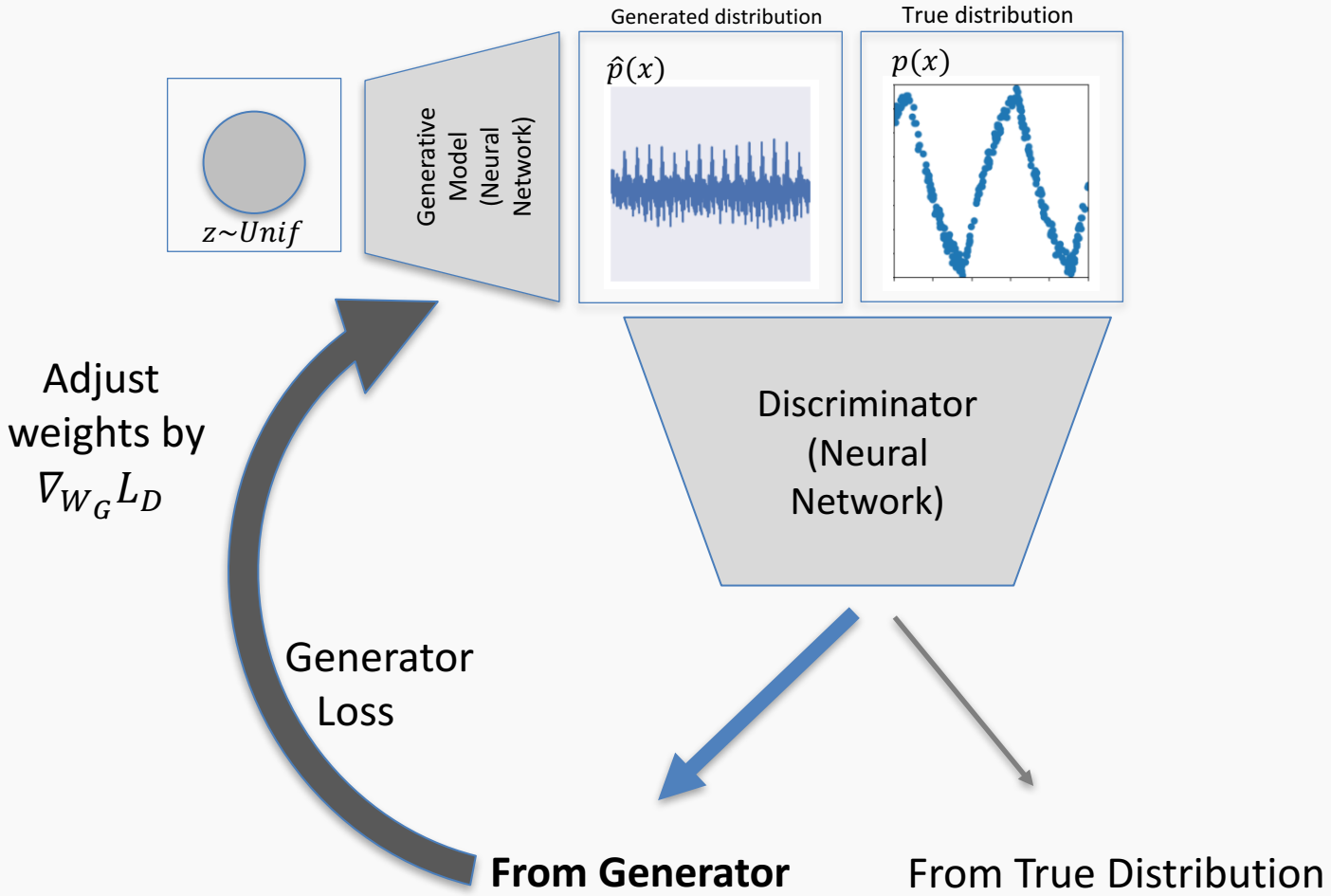
Generative Adversarial Networks



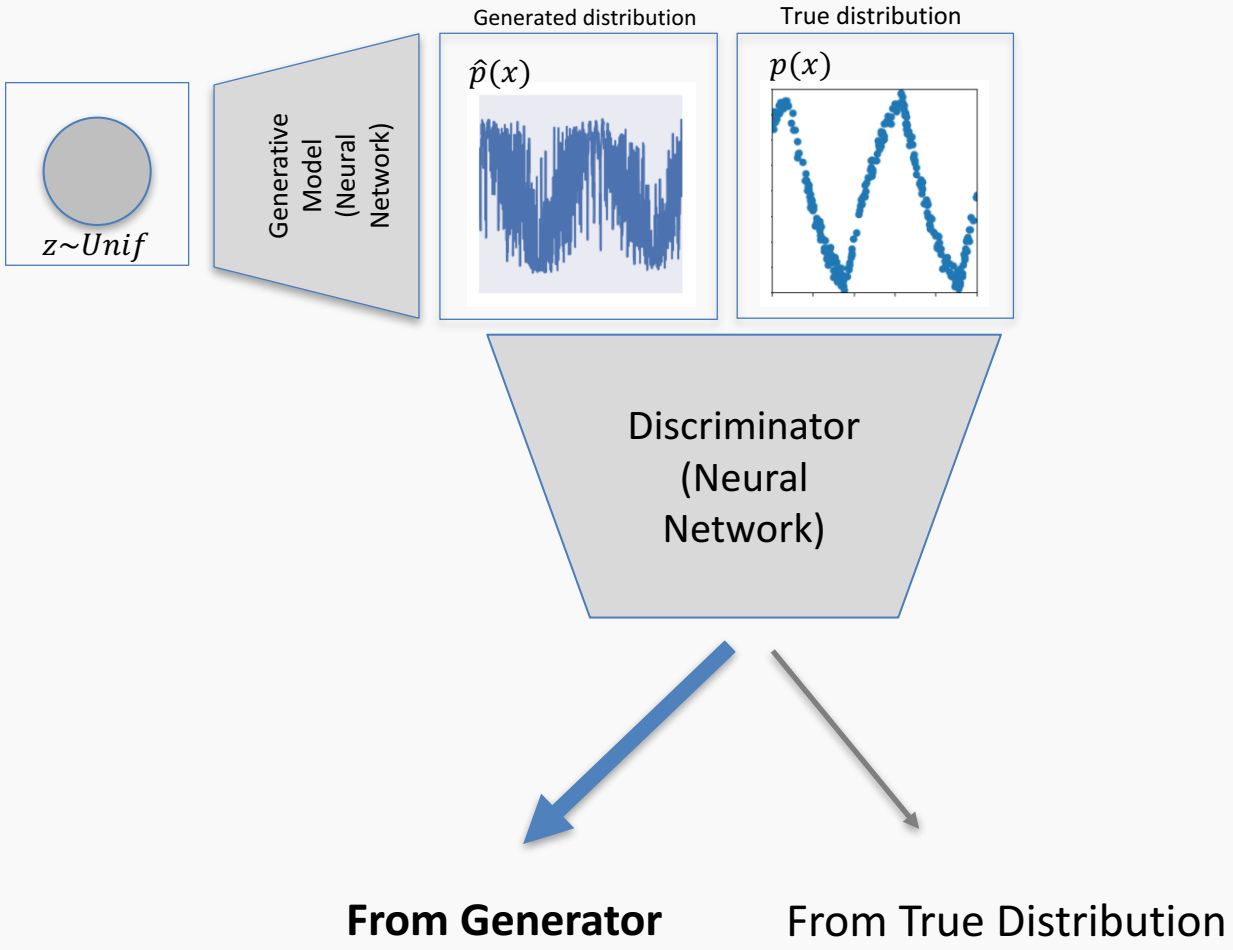
Generative Adversarial Networks



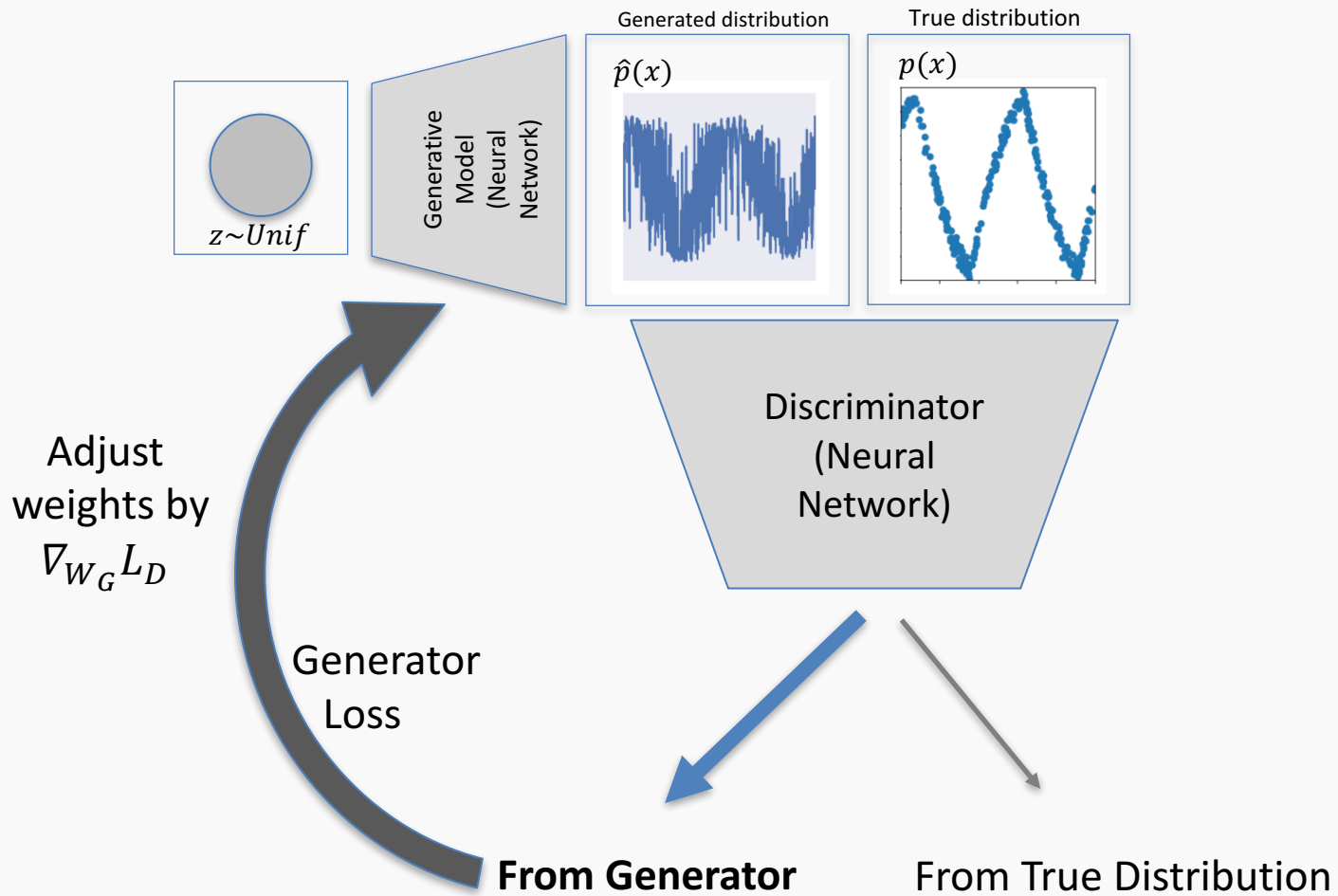
Generative Adversarial Networks



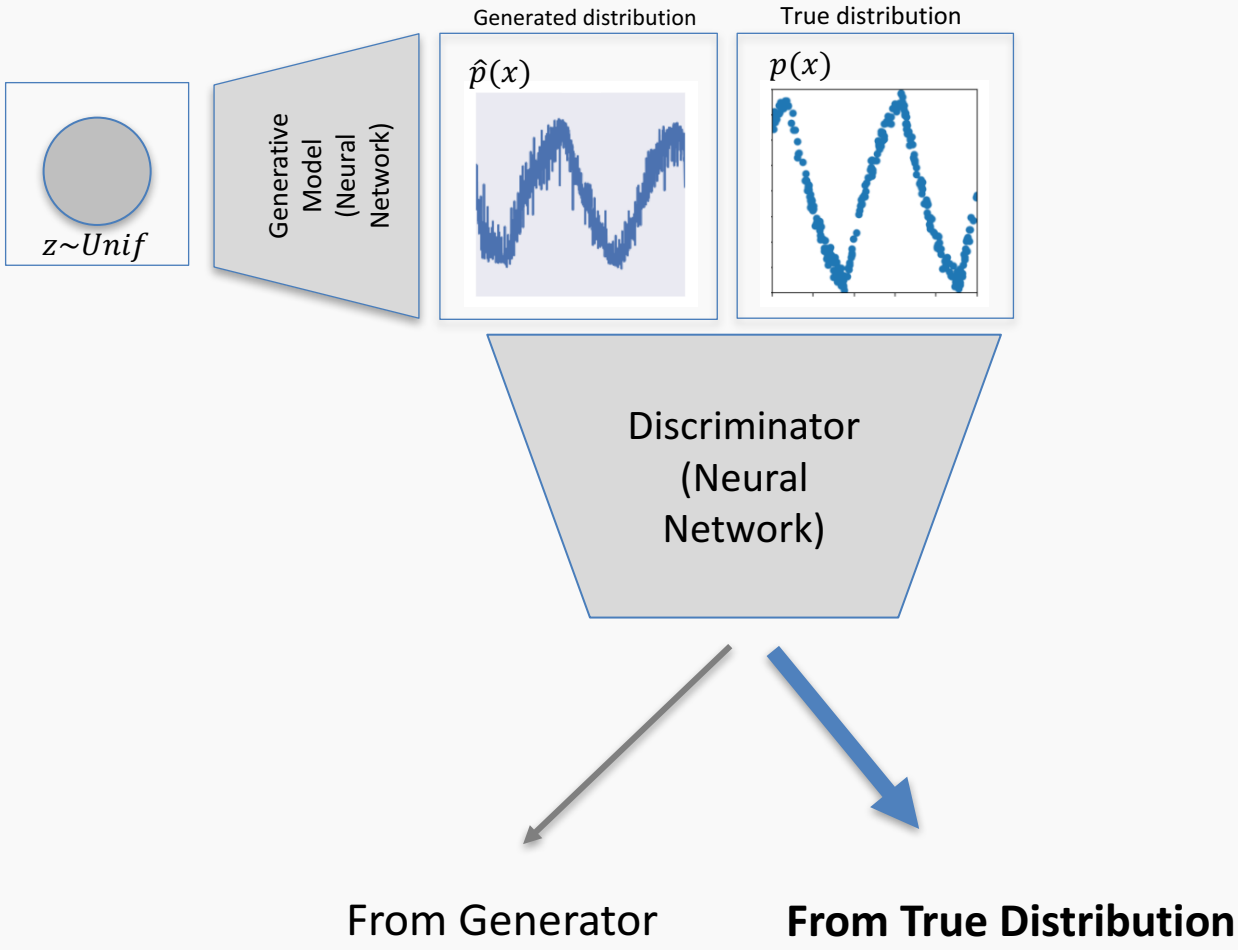
Generative Adversarial Networks



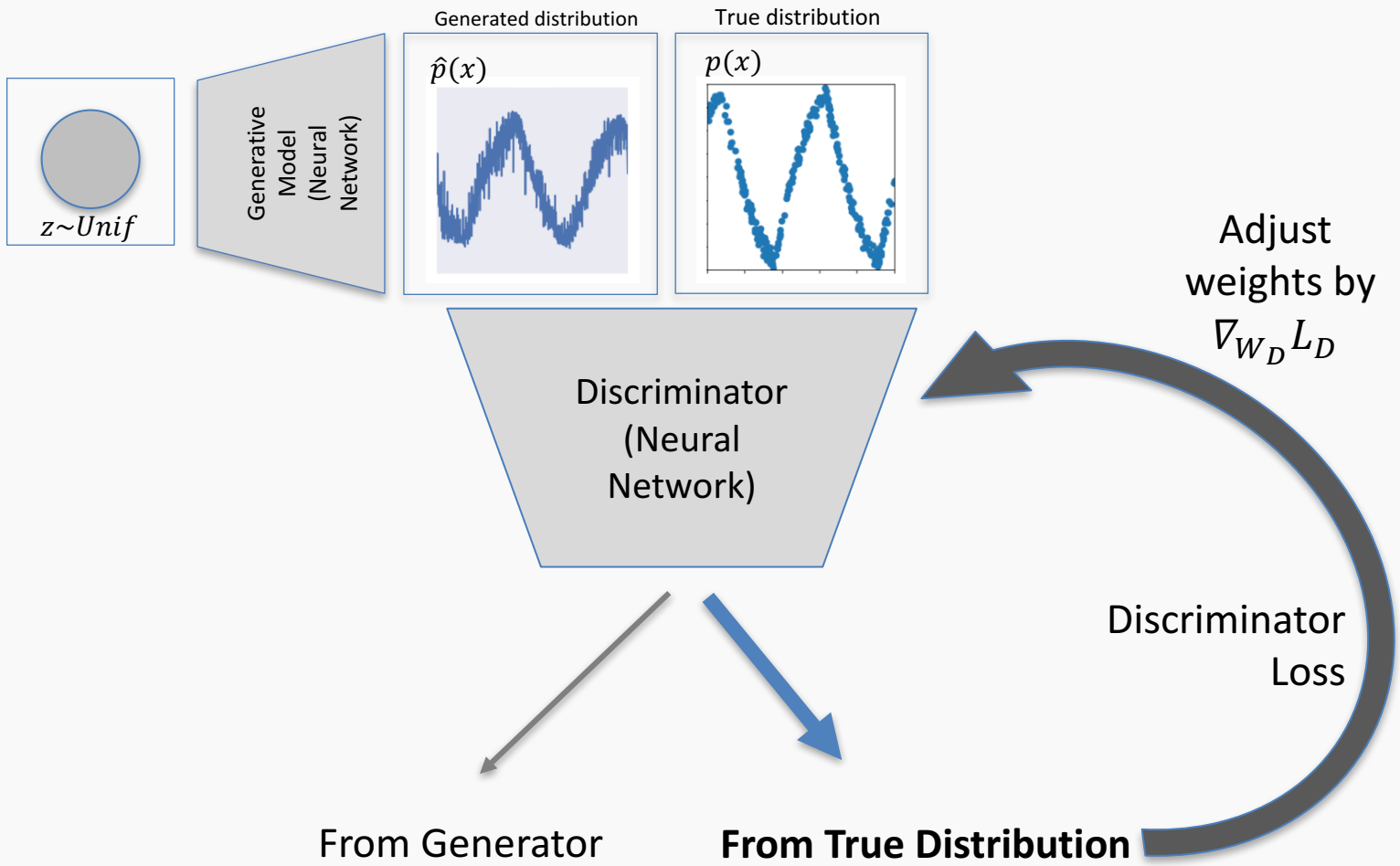
Generative Adversarial Networks



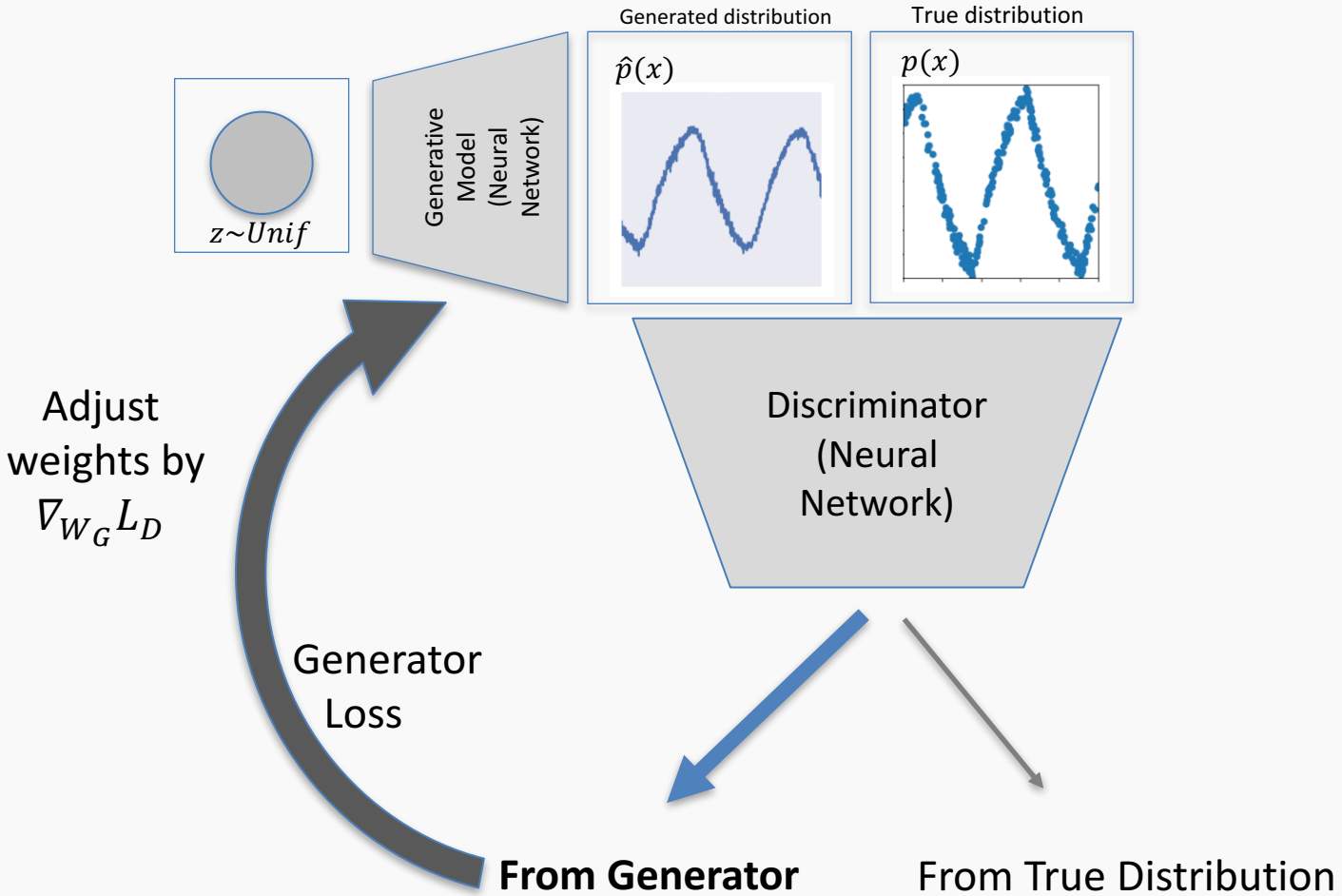
Generative Adversarial Networks



Generative Adversarial Networks



Generative Adversarial Networks

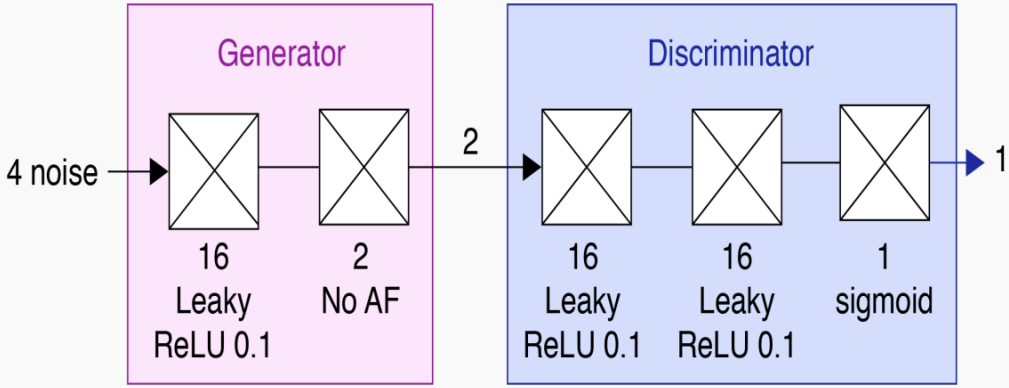


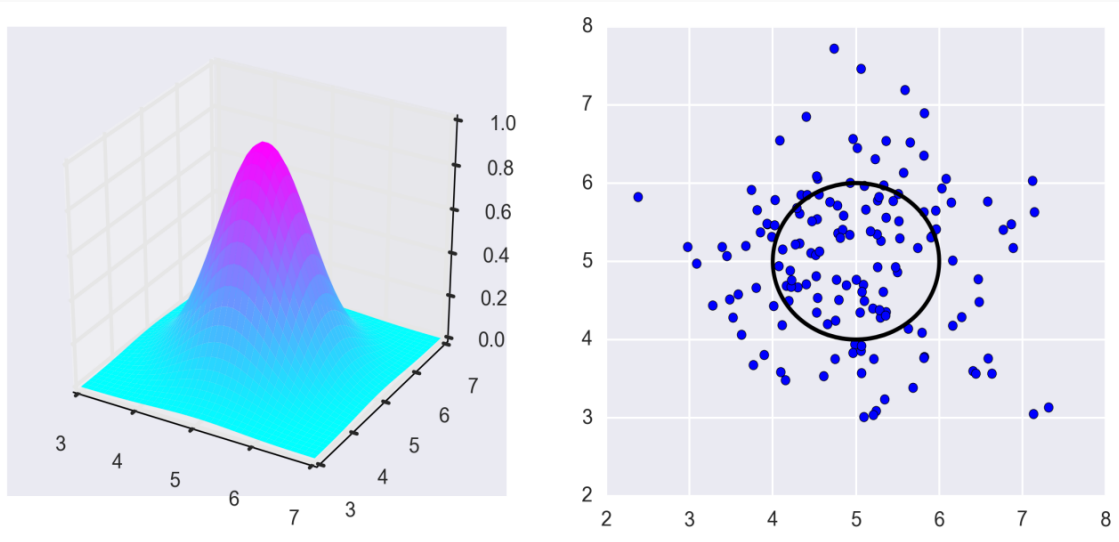
Building GANS: Fully Connected Case

Let's build a FC simple GAN to generate points in a 2-dimensional Gaussian Distribution.

- **Generator**
  - Takes 4 random numbers
  - Generates a coordinate pair drawn from a specific 2-D Gaussian

- **Discriminator**
  - Takes an input point in the form of a coordinate pair
  - Determines whether the point is drawn from a specific 2-D Gaussian

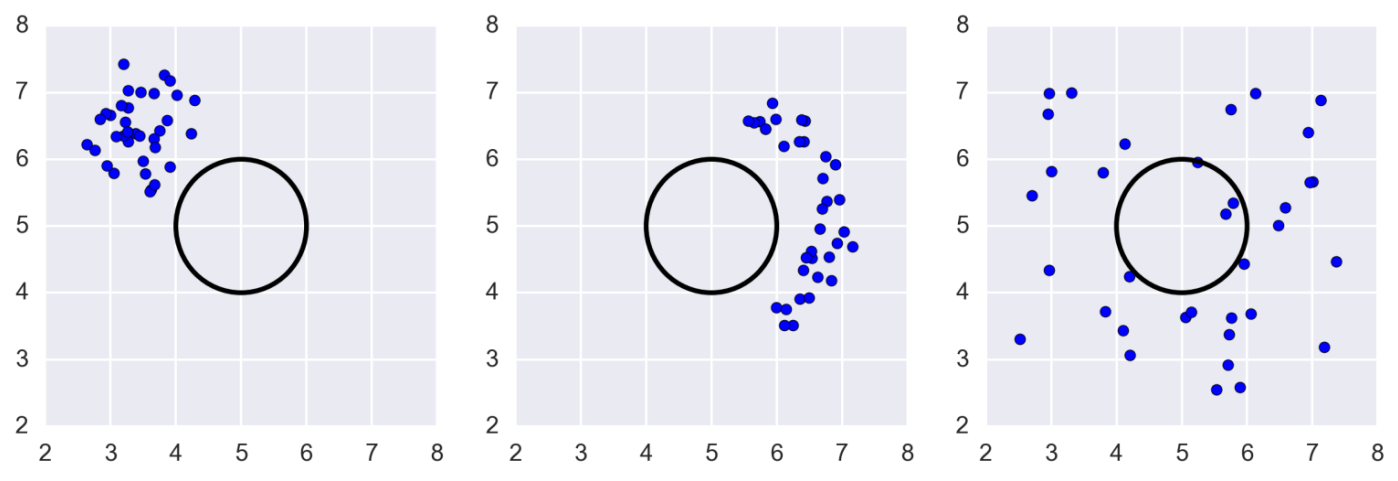




Building GANS: Fully Connected Case

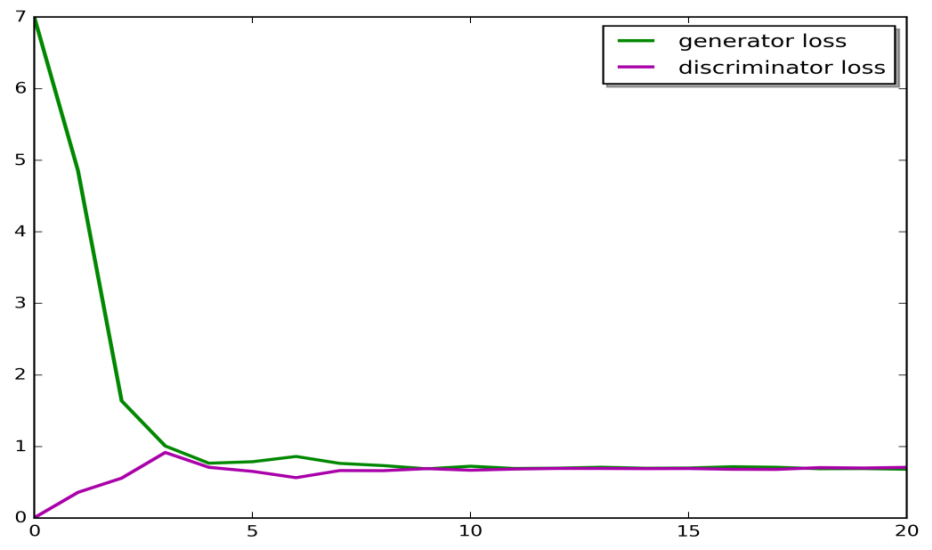
Train the Networks based on their ability to generate/discriminate batches of points drawn from the distribution.

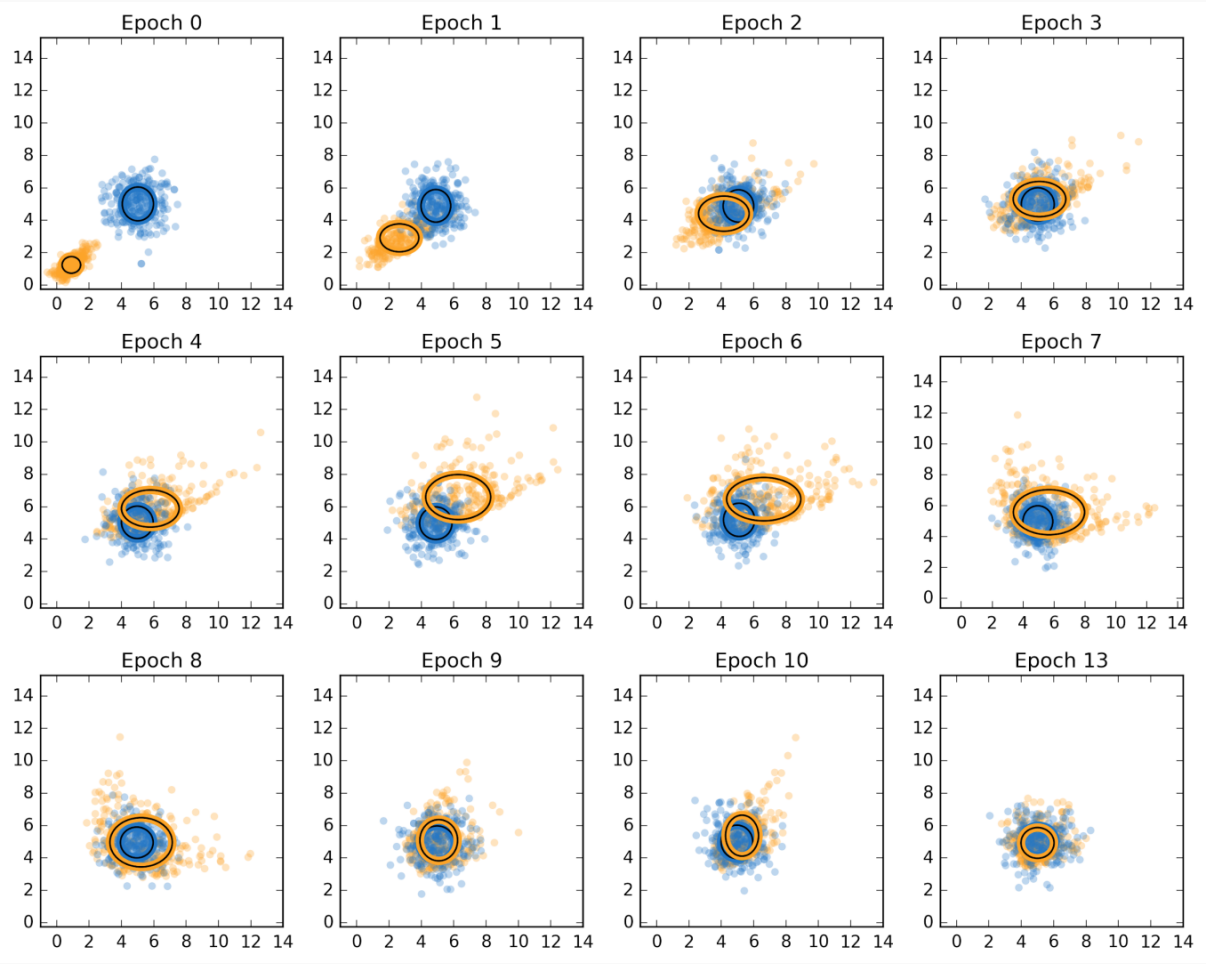
Are these batches of points drawn from the right distribution?



Building GANS: Fully Connected Case

As the generator and discriminator loss converges, the batch of points generated by the generator (in the yellow) approaches the real batch of points (in the blue)

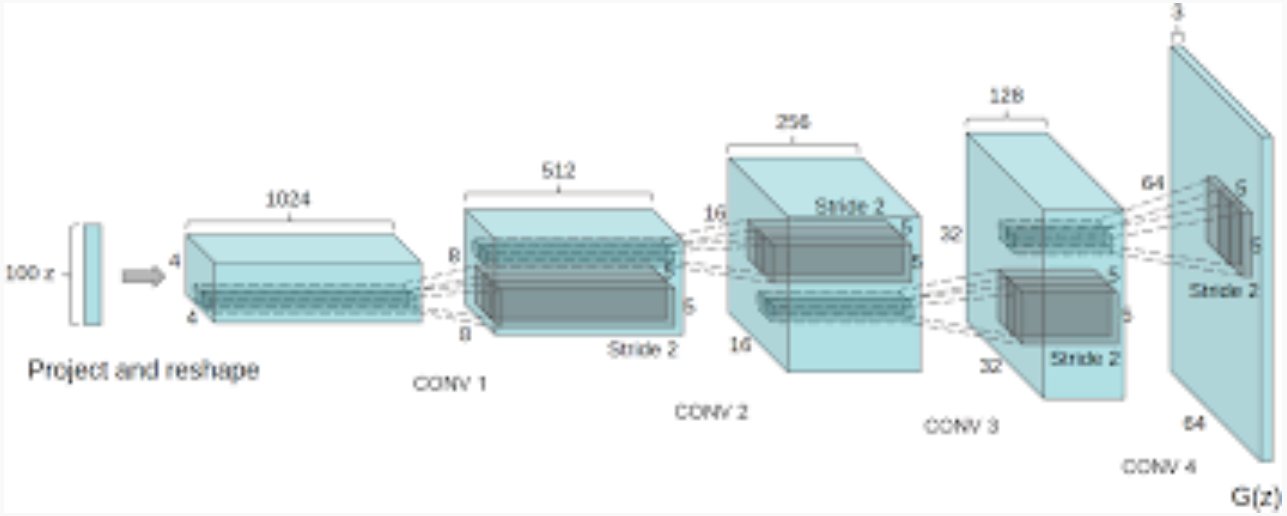




Building GANS: DCGAN

Deep Convolutional GAN (**DCGAN**)
-- Alex Radford et al. 2016

- Eliminate fully connected layers.
- Max Pooling BAD! Replace all max pooling with convolutional stride
- Use transposed convolution for upsampling.
- Use Batch normalization



Building GANS: DCGAN

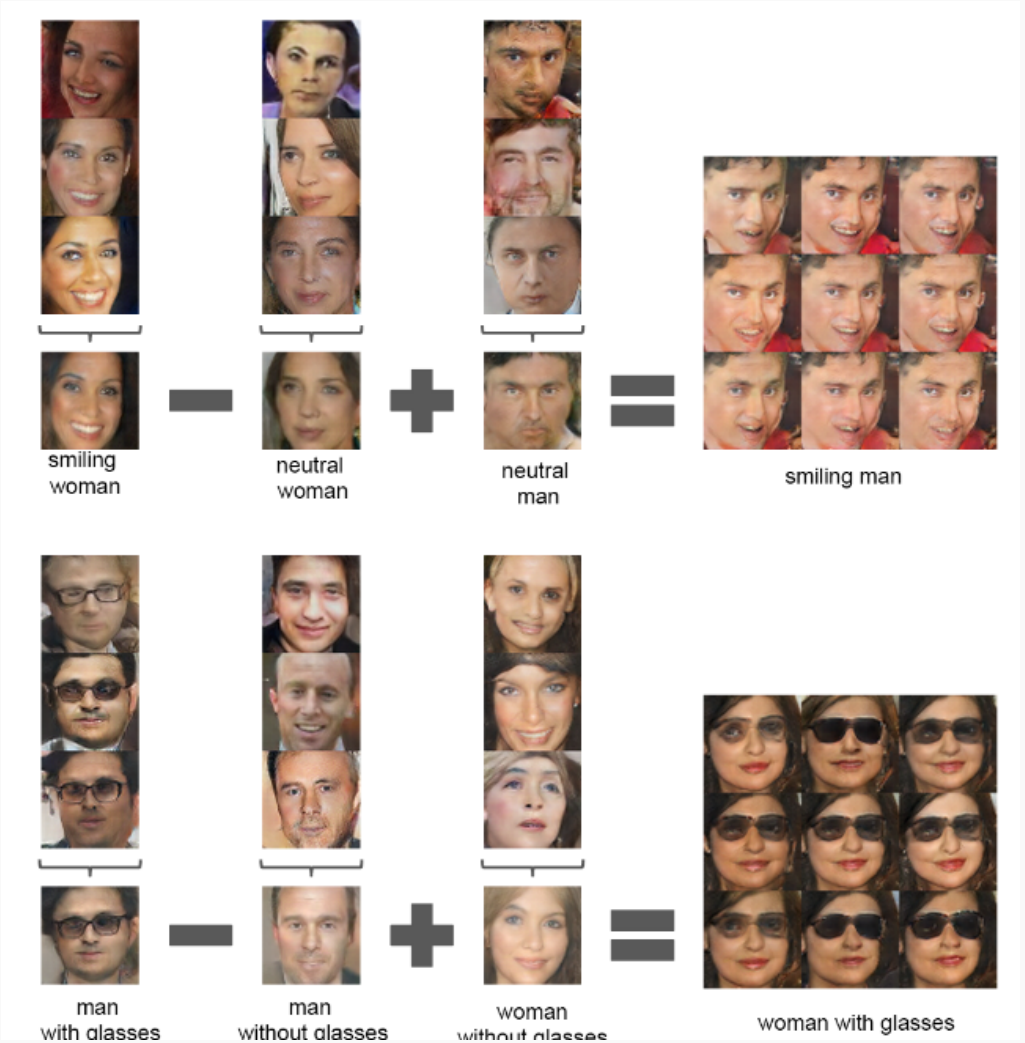
DCGAN on MNIST

Generated digits

Building GANS: DCGAN

DCGAN
Interpretable
Vector Arithmetic

Parallels to
Autoencoder Case



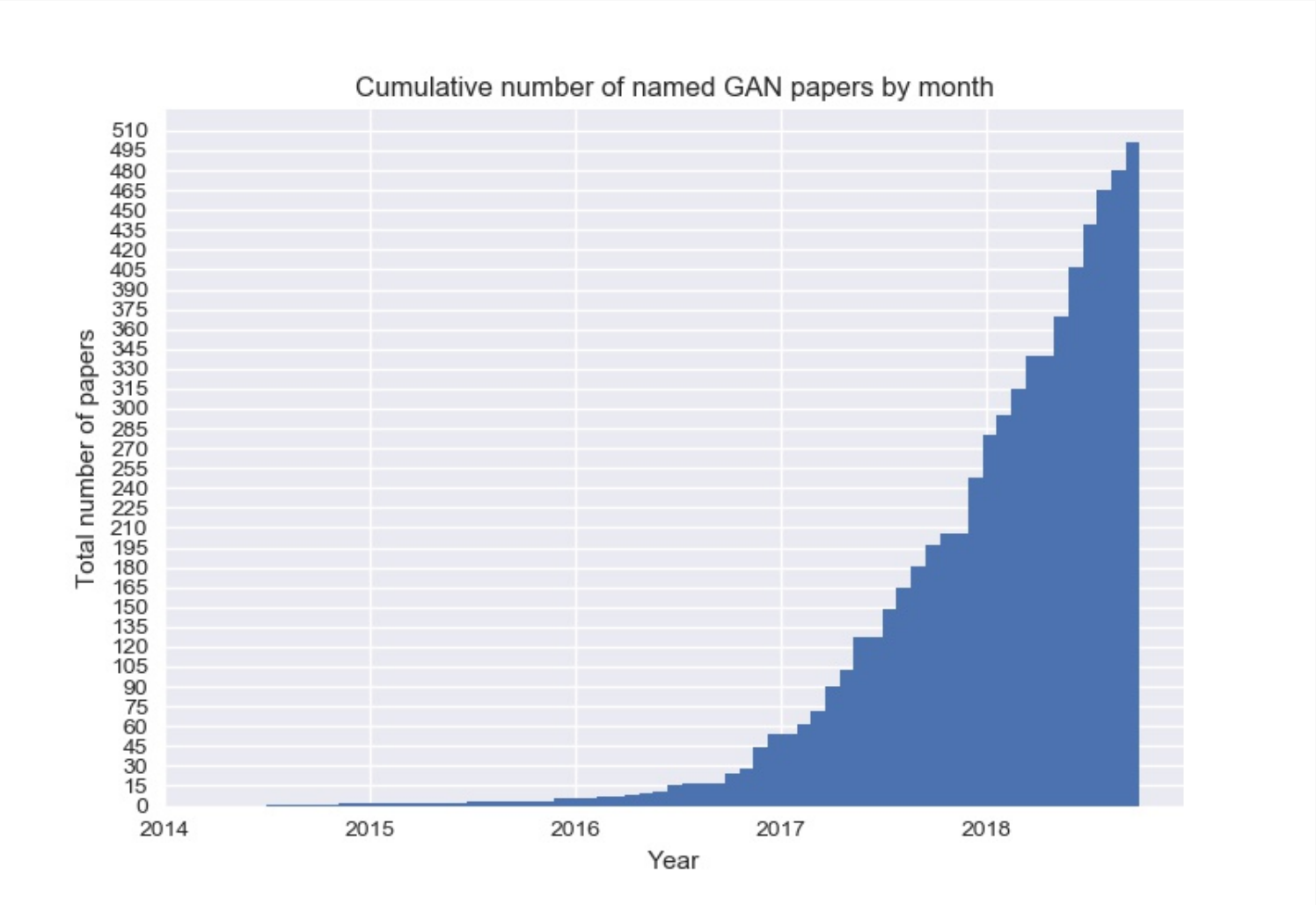
https://arxiv.org/pdf/1511.06434v2.pdf

Evolution of GANs

5 Years of Improvement in Artificially Generated Faces

https://twitter.com/goodfellow_ian/status/969776035649675265?lang=en

Evolution of GANs



Ian Goodfellow
@goodfellow_ian

Follow

One of my favorite samples from the Progressive GANs paper is this one from the "cat" category. Apparently some of the cat training photos were memes with text. The GAN doesn't know what text is so it has made up new text-like imagery in the right place for a meme caption.

11:41 AM - 3 Dec 2017

GAN Rules of Thumb (GANHACKs)

Normalize the inputs

- normalize the images between -1 and 1
- tanh as the last layer of the generator output

Use Spherical Z

- Don’t sample from a Uniform distribution
- When doing interpolations, do the interpolation via a great circle, rather than a straight line from point A to point B
- Tom White's Sampling Generative Networks ref code https://github.com/dribnet/plat has more details

GAN Rules of Thumb (GANHACKs)

BatchNormalization

- Construct different mini-batches for real and fake, i.e. each mini-batch needs to contain only all real images or all generated images.
- when batchnorm is not an option use instance normalization (for each sample, subtract mean and divide by standard deviation).

Avoid Sparse Gradients: ReLU, MaxPool

- the stability of the GAN game suffers if you have sparse gradients
- LeakyReLU = good (in both G and D)
- For Downsampling, use: Average Pooling, Conv2d + stride
- For Upsampling, use: PixelShuffle, ConvTranspose2d + stride
    - PixelShuffle: https://arxiv.org/abs/1609.05158

GAN Rules of Thumb (GANHACKs)

Use Soft and Noisy Labels

- Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2, and if it is a fake sample, replace it with 0.0 and 0.3 (for example).
  - Salimans et. al. 2016
- make the labels the noisy for the discriminator: occasionally flip the labels when training the discriminator

See GANHACKs (https://github.com/soumith/ganhacks) for more tips

Outline

Review of AE and VAE

GANS

- Motivation
- Formalism
- Training

Game Theory, minmax

Challenges:

- Big Samples
- Modal collapse

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 "winner."

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 a-sec on Wednesday.

Outline

Review of AE and VAE

GANS

- Motivation
- Formalism
- Training

Game Theory, minmax

Challenges:

- Big Samples
- Modal collapse

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 GAN or 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 to 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.

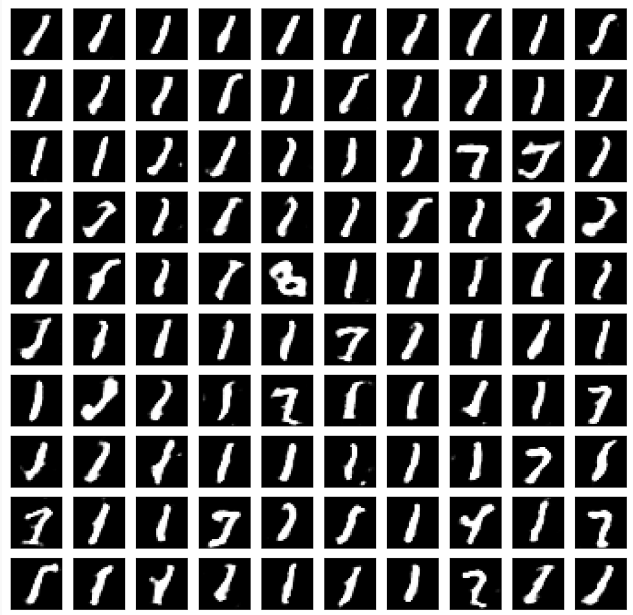
This problem of producing just one successful output over and over is called **modal collapse**.

[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

Next?

W-GANS

Adversarial NN