# Generative Adversarial Network (GAN)

Goodfellow et al, “Generative Adversarial Networks”, NIPS 2014

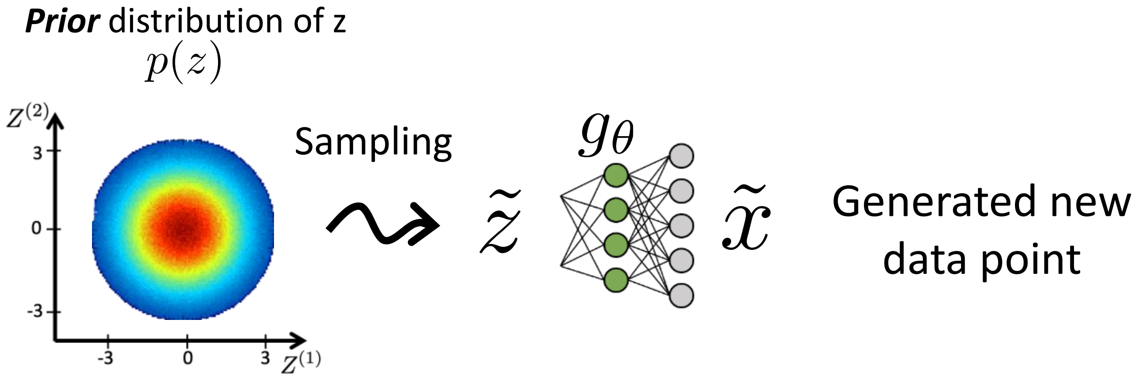
# What is a Generative Model?

"A generative model describes **how a dataset is generated**, in terms of a **probabilistic model**. By **sampling** from this model, we are **able to generate new data**."

Generative Deep Learning, by David Foster

**Generative** models: $g : \mathcal{Z} \rightarrow X$

*We assume: z "causes" x*



***Prior*** distribution of z

$p(z)$

Sampling

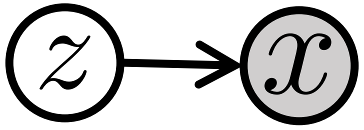
$\tilde{z}$ $g_\theta$ $\tilde{x}$

Generated new data point

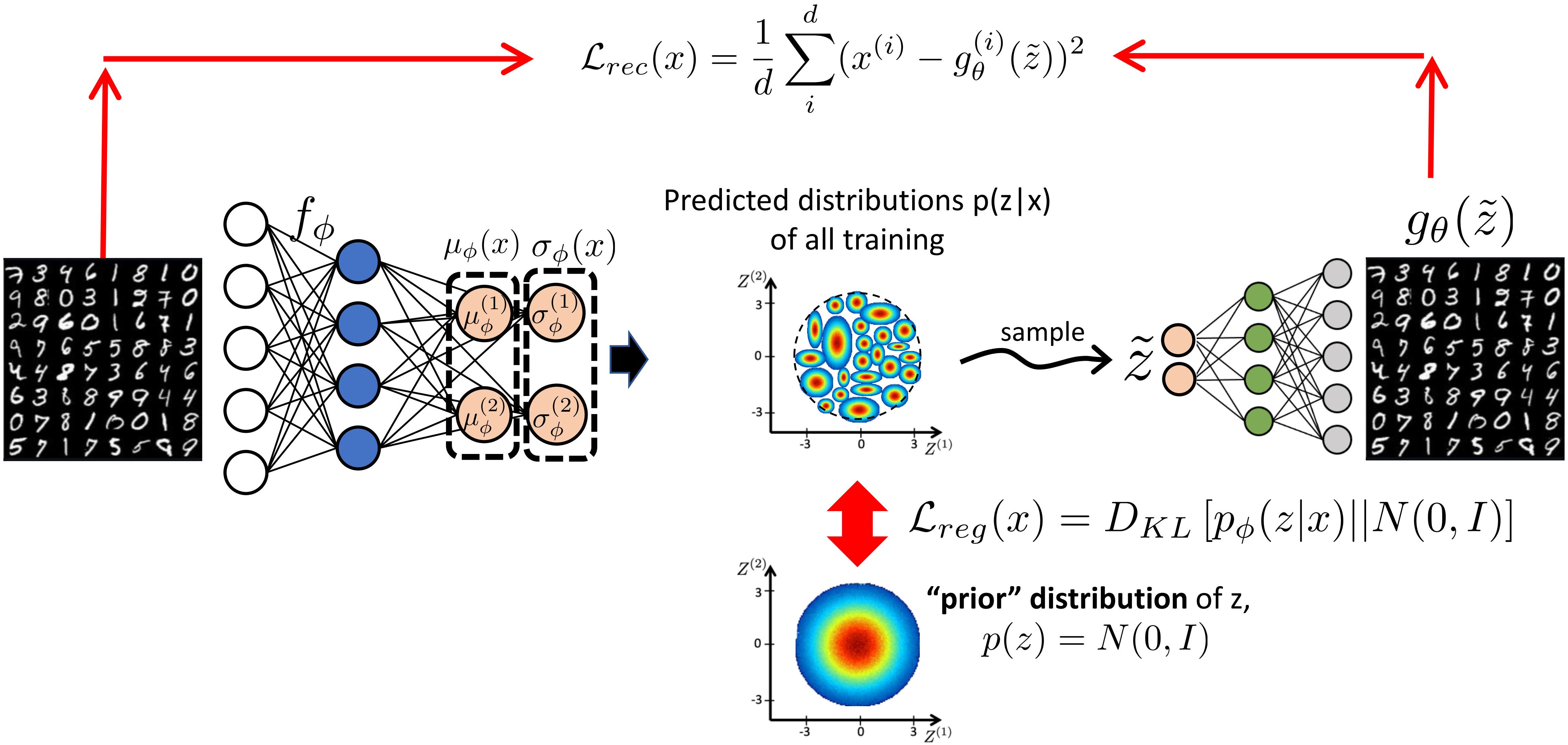
z: content, lighting, zoom …

x: the photo

z "causes" x



# VAEs: Training



$$\mathcal{L}_{rec}(x) = \frac{1}{d}\sum_{i}^{d}(x^{(i)} - g_\theta^{(i)}(\tilde{z}))^2$$

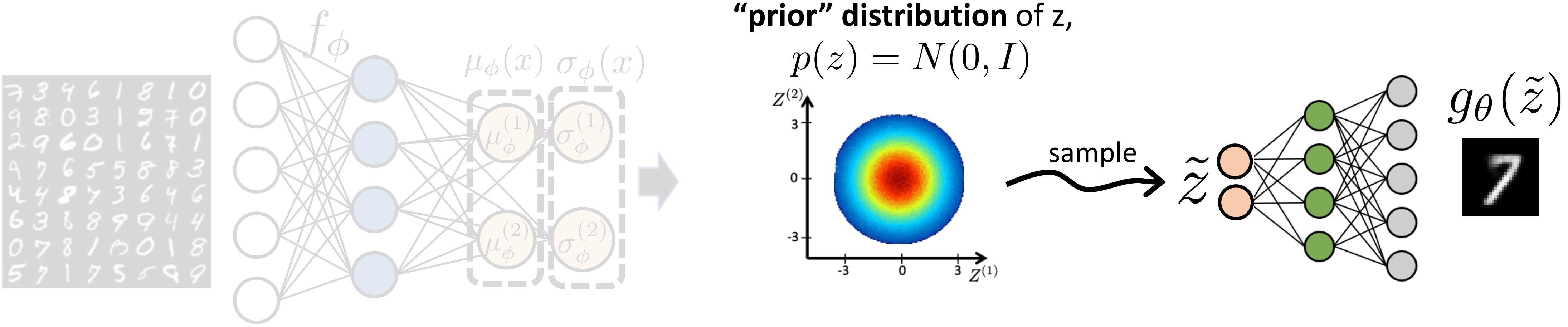
Predicted distributions p(z|x) of all training

$$\mathcal{L}_{reg}(x) = D_{KL}\left[p_\phi(z|x)||N(0,I)\right]$$

**“prior” distribution** of z,

$p(z) = N(0, I)$

# VAEs: Generating after sampling from a prior p(z)



**“prior” distribution** of z,

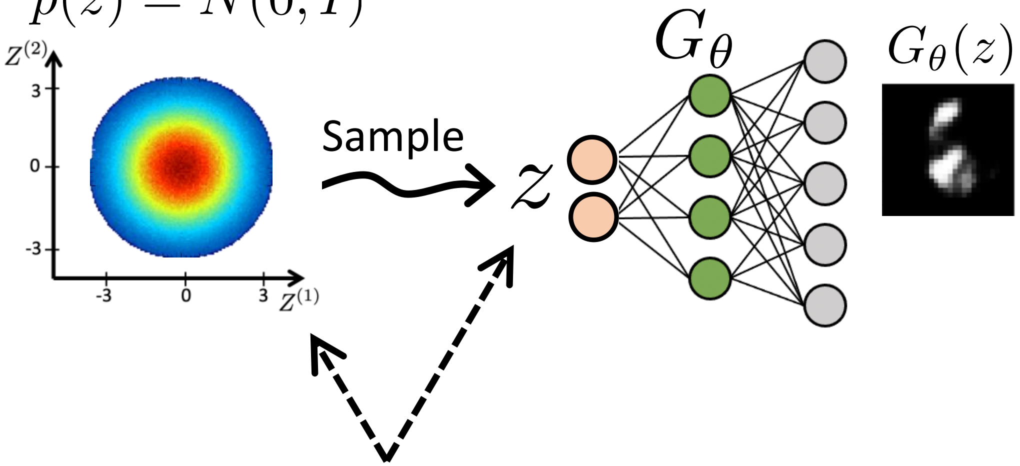
$p(z) = N(0, I)$

# Generative Adversarial Network

**“prior” distribution** of z,

$$p(z) = N(0, I)$$

**Generator**



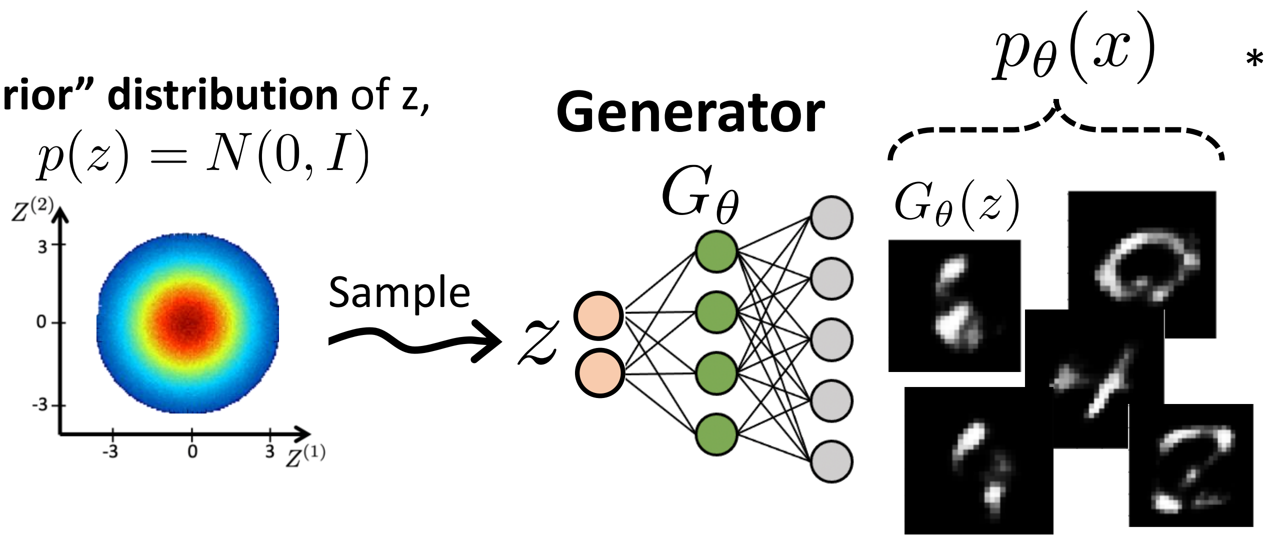
Sample

*Can be multi-dimensional*

# Generative Adversarial Network

**“prior” distribution** of z,

$p(z) = N(0, I)$



Sample

**Generator**

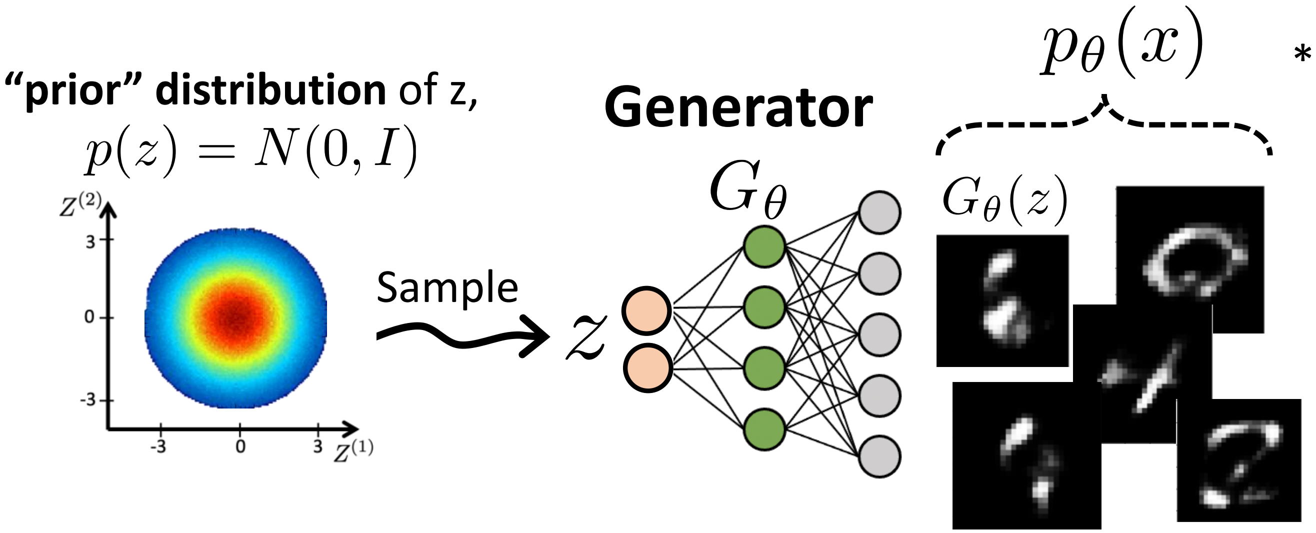
$G_\theta$

$G_\theta(z)$

$p_\theta(x)$

Generated data are samples from the *model’s* probability density function of x

# Generative Adversarial Network



**“prior” distribution** of z,

$p(z) = N(0, I)$

Sample

$z$

**Generator**

$G_\theta$

$G_\theta(z)$

$p_\theta(x)$

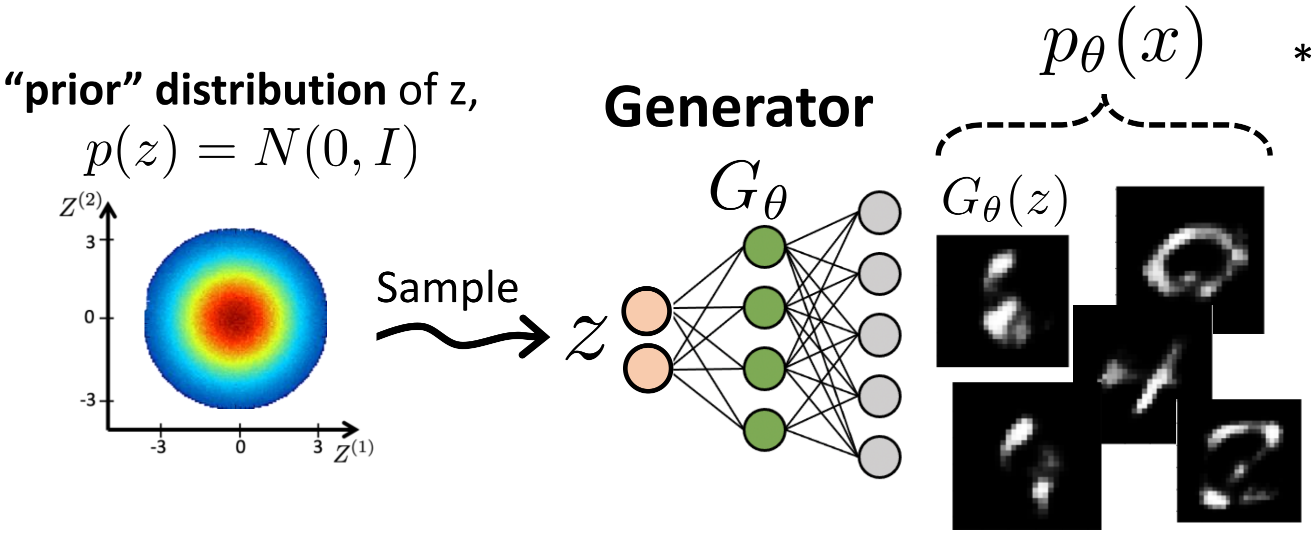
Generated data are samples from the *model’s* probability density function of x

$p_{data}(x)$

Real data
(training database)

Training data are samples from the *real* probability density function of x

# How can we learn $p_\theta(x) \approx p_{data}(x)$? *(Density estimation)*



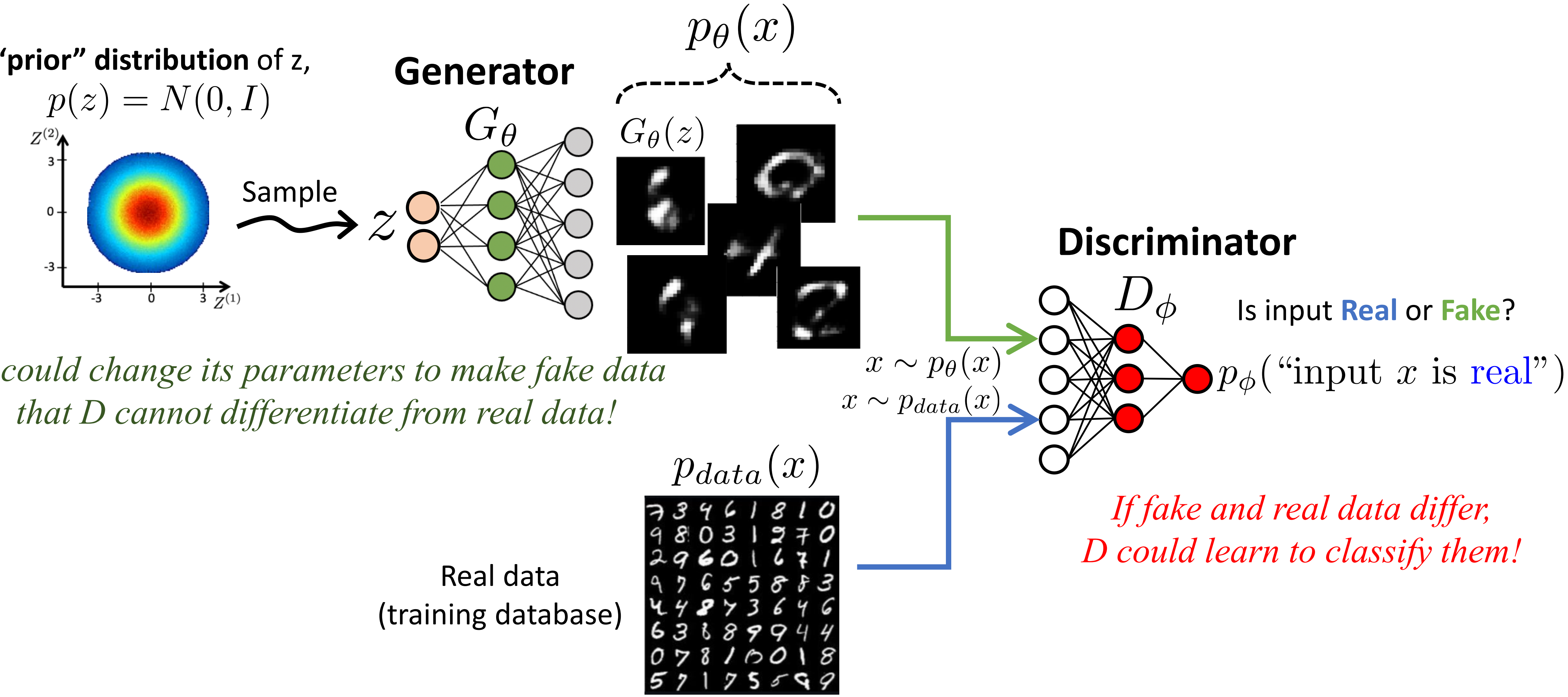
Generated data are samples from the *model’s* probability density function of x

$p_{data}(x)$

Real data (training database)

Training data are samples from the *real* probability density function of x

# How can we learn $p_\theta(x) \approx p_{data}(x)$? *(Density estimation)*



**“prior” distribution** of z,
$p(z) = N(0, I)$

Sample $\rightsquigarrow z$

**Generator**
$G_\theta$

$p_\theta(x)$
$G_\theta(z)$

**Discriminator**
$D_\phi$

Is input **Real** or **Fake**?

$x \sim p_\theta(x)$
$x \sim p_{data}(x)$

$p_\phi(\text{“input } x \text{ is real”})$

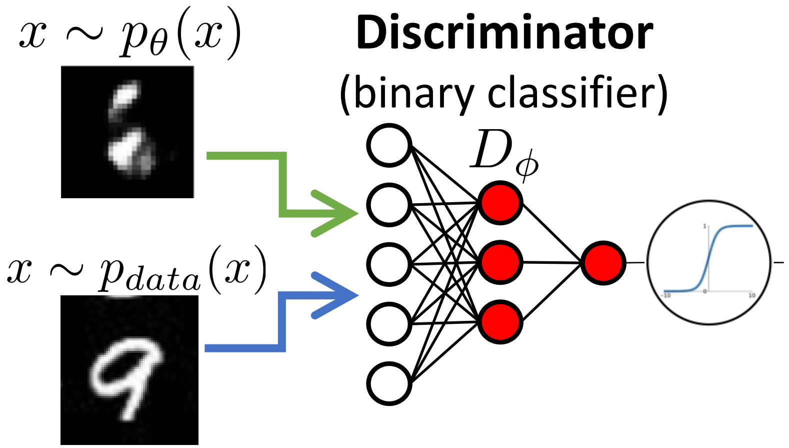
*G could change its parameters to make fake data that D cannot differentiate from real data!*

*If fake and real data differ, D could learn to classify them!*

$p_{data}(x)$

Real data
(training database)

# The Discriminator - Binary Classifier of inputs: Real or Fake?



$x \sim p_\theta(x)$

$x \sim p_{data}(x)$

**Discriminator**

(binary classifier)

$D_\phi$

Is input Real or Fake?

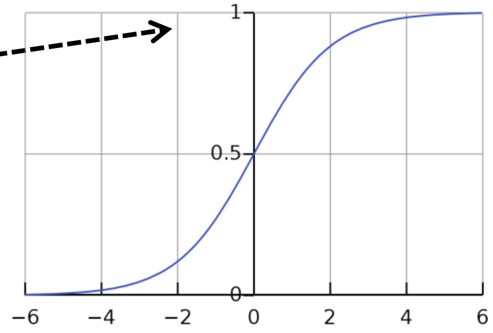
$$D_\phi(x) = p_\phi(x \text{ is real})$$

$D_\phi(x)\!=\!1$: D is certain $x$ is real

$D_\phi(x)\!=\!0$: D is certain $x$ is NOT real

**Sigmoid activation function**

$$s(\alpha) = \frac{1}{1+e^{-\alpha}}$$



$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{x\sim p_\theta(x)}\left[\log(1 - D_\phi(x))\right]$$

# The Discriminator - Binary Classifier of inputs: Real or Fake?

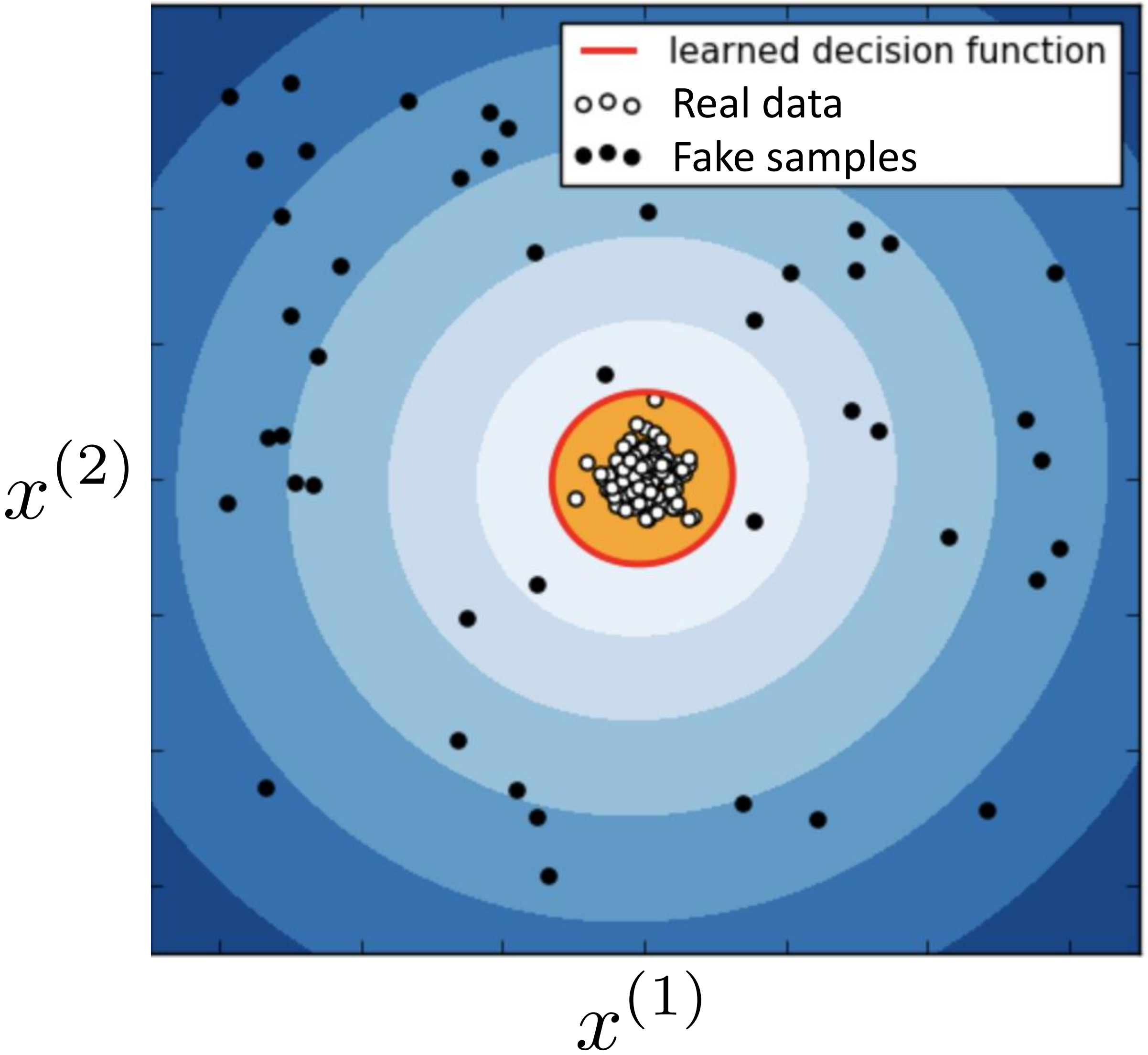
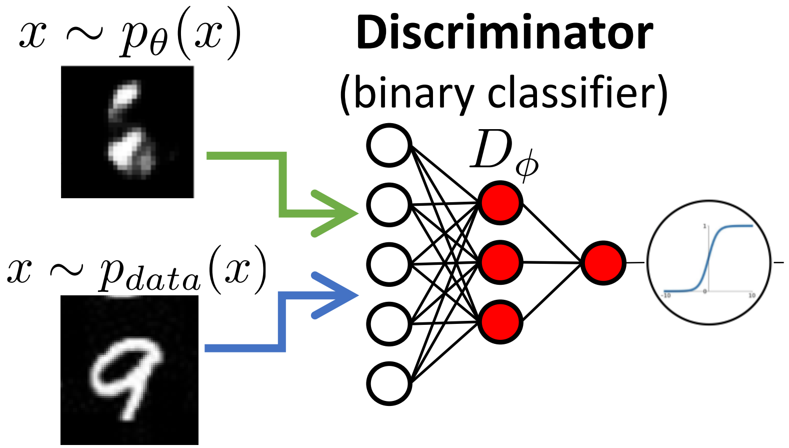


Image modified from Scipy

# The Discriminator - Binary Classifier of inputs: Real or Fake?



$x \sim p_\theta(x)$

$x \sim p_{data}(x)$

**Discriminator**

(binary classifier)

$D_\phi$

Is input **Real** or **Fake**?

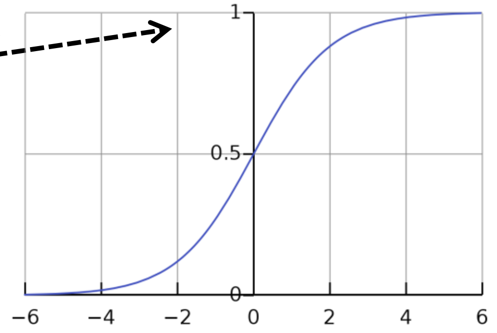
$$D_\phi(x) = p_\phi(x \text{ is real})$$

$D_\phi(x)\!=\!1$: D is certain $x$ is real

$D_\phi(x)\!=\!0$: D is certain $x$ is NOT real

**Sigmoid activation function**

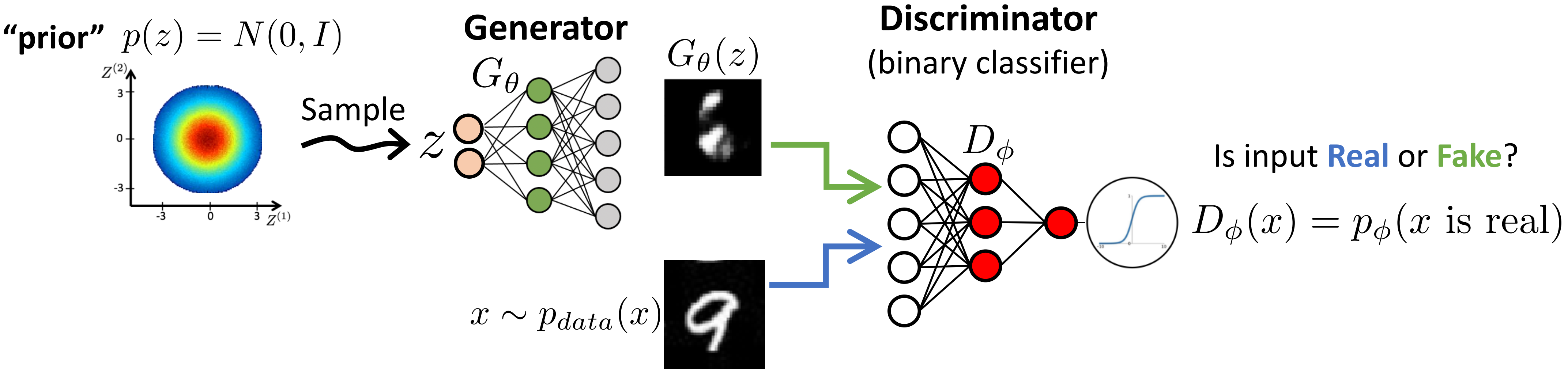
$$s(\alpha) = \frac{1}{1+e^{-\alpha}}$$



$$\phi' = \arg\max_{\phi} \mathbb{E}_{x \sim p_{data}(x)} \left[\log D_\phi(x)\right] + \mathbb{E}_{x \sim p_\theta(x)} \left[\log(1 - D_\phi(x))\right]$$

Discriminator learns a decision boundary between real and fake samples.
For each input, it predicts the probability that the input is real or fake.

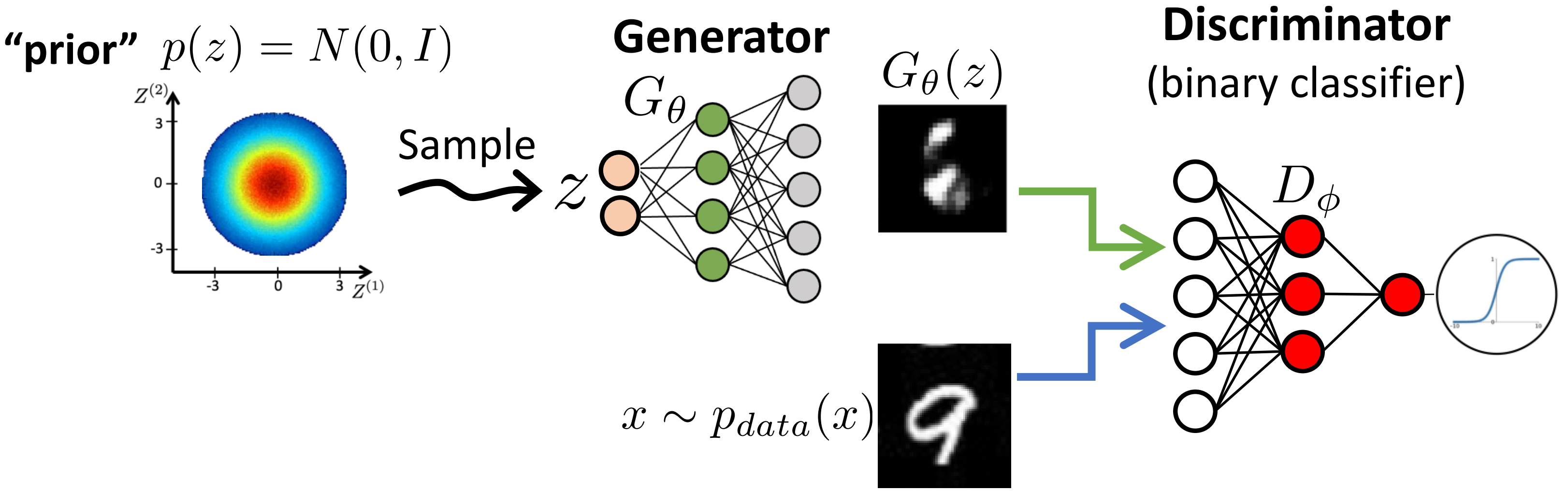
# The Discriminator - Binary Classifier of inputs: Real or Fake?



$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \underbrace{\mathbb{E}_{x\sim p_\theta(x)}}\left[\log(1 - D_\phi(x))\right]$$

$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1 - D_\phi(\overbrace{G_\theta(z)}))\right]$$

# The Discriminator - Binary Classifier of inputs: Real or Fake?



Is input Real or Fake?

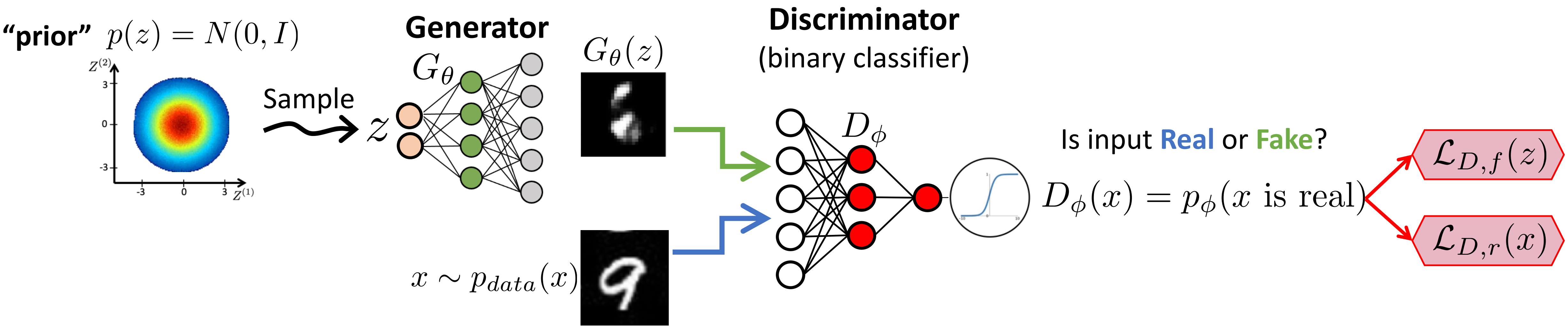
$$D_\phi(x) = p_\phi(x \text{ is real})$$

$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \underbrace{\mathbb{E}_{x\sim p_\theta(x)}}\left[\log(1 - D_\phi(x))\right]$$

$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1 - D_\phi(\overbrace{G_\theta(z)}))\right]$$

$$\phi' = \arg\min_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[-\log D_\phi(x)\right] + \mathbb{E}_{z\sim p(z)}\left[-\log(1 - D_\phi(G_\theta(z)))\right]$$

# The Discriminator - Binary Classifier of inputs: Real or Fake?



$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{x\sim p_\theta(x)}\left[\log(1 - D_\phi(x))\right]$$

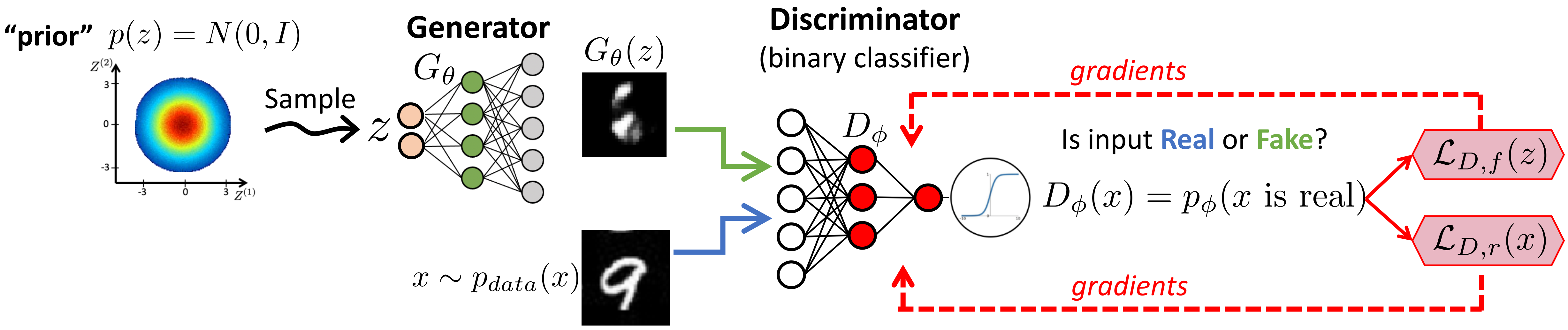
$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1 - D_\phi(G_\theta(z)))\right]$$

$$\phi' = \arg\min_\phi \mathbb{E}_{x\sim p_{data}(x)}\underbrace{\left[-\log D_\phi(x)\right]}_{\mathcal{L}_{D,r}(x)} + \mathbb{E}_{z\sim p(z)}\underbrace{\left[-\log(1 - D_\phi(G_\theta(z)))\right]}_{\mathcal{L}_{D,f}(z)}$$

**Can be understood as if D learns by minimizing 2 losses:**

- One for real samples (make D(x) go to 1)
- One for fake samples (make D(G(z)) go to 0)

# The Discriminator - Binary Classifier of inputs: Real or Fake?



**"prior"** $p(z) = N(0, I)$

Sample

$z$

**Generator**

$G_\theta$

$G_\theta(z)$

$x \sim p_{data}(x)$

**Discriminator**

(binary classifier)

$D_\phi$

Is input Real or Fake?

$D_\phi(x) = p_\phi(x \text{ is real})$

$\mathcal{L}_{D,f}(z)$

$\mathcal{L}_{D,r}(x)$

*gradients*

*gradients*

$$\phi' = \arg\max_{\phi} \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{x\sim p_\theta(x)}\left[\log(1 - D_\phi(x))\right]$$

$$\phi' = \arg\max_{\phi} \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1 - D_\phi(G_\theta(z)))\right]$$

$$\phi' = \arg\min_{\phi} \mathbb{E}_{x\sim p_{data}(x)}\underbrace{\left[-\log D_\phi(x)\right]}_{\mathcal{L}_{D,r}(x)} + \mathbb{E}_{z\sim p(z)}\underbrace{\left[-\log(1 - D_\phi(G_\theta(z)))\right]}_{\mathcal{L}_{D,f}(z)}$$

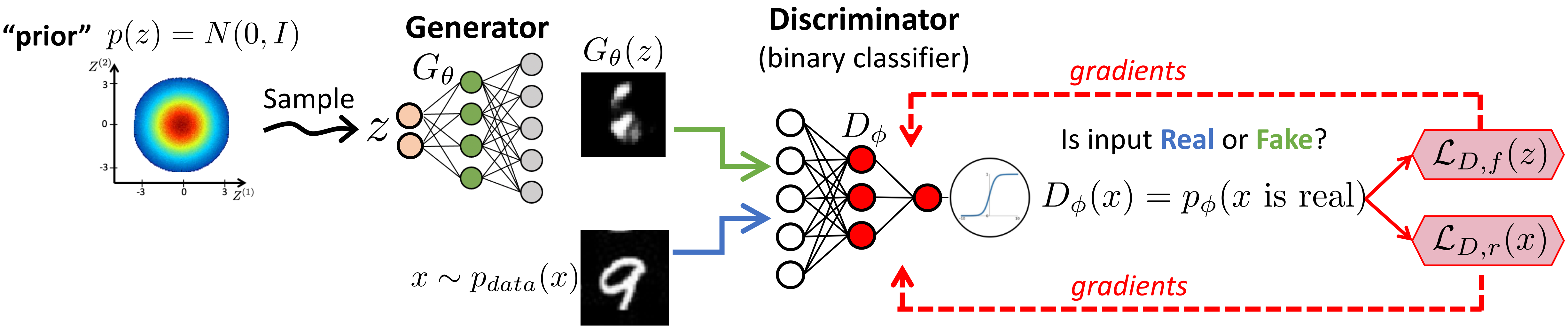
**Can be understood as if D learns by minimizing 2 losses:**

- One for real samples (make D(x) go to 1)
- One for fake samples (make D(G(z)) go to 0)

**Both losses are functions of D's parameters** $\phi$

- By backpropagation, we can learn optimal D.
- Note: All this for "fixed" G parameters.

# The Discriminator - Binary Classifier of inputs: Real or Fake?



**"prior"** $p(z) = N(0, I)$

Sample

**Generator**

$G_\theta$

$z$

$G_\theta(z)$

$x \sim p_{data}(x)$

**Discriminator**

(binary classifier)

$D_\phi$

*gradients*

Is input Real or Fake?

$D_\phi(x) = p_\phi(x \text{ is real})$

$\mathcal{L}_{D,f}(z)$

$\mathcal{L}_{D,r}(x)$

*gradients*

$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}[\log D_\phi(x)] + \underbrace{\mathbb{E}_{x\sim p_\theta(x)}}[\log(1-D_\phi(x))]$$

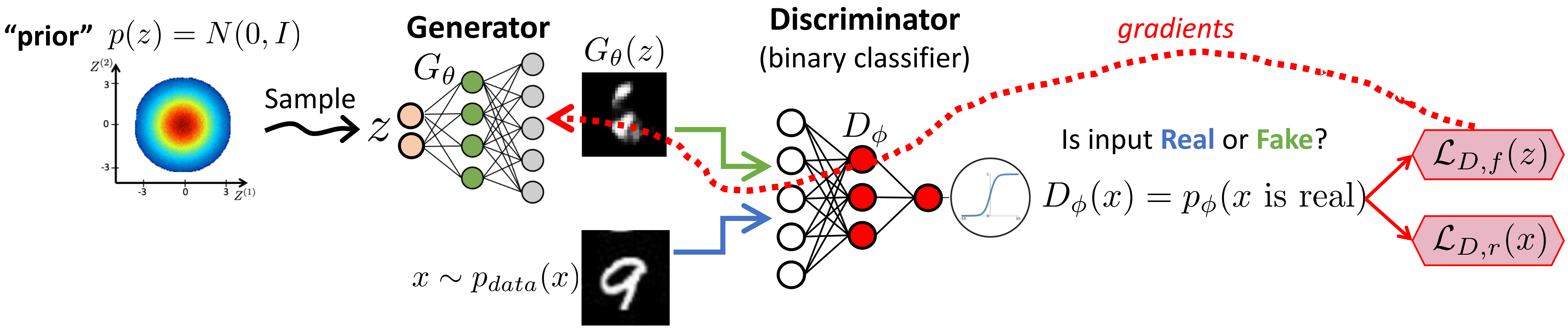
$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}[\log D_\phi(x)] + \mathbb{E}_{z\sim p(z)}[\log(1-D_\phi(\overbrace{G_\theta(z)}))]$$

$$\phi' = \arg\min_\phi \mathbb{E}_{x\sim p_{data}(x)} \underbrace{[-\log D_\phi(x)]}_{\mathcal{L}_{D,r}(x)} + \mathbb{E}_{z\sim p(z)} \underbrace{[-\log(1-D_\phi(G_\theta(z)))]}_{\mathcal{L}_{D,f}(z)}$$

**In practice**, SGD with **2 batches**: batch of ($N_{D,r}$) **real** data and batch of ($N_{D,f}$) **fake** samples:

$$\phi' = \arg\min_\phi \frac{1}{N_{D,r}} \sum_i^{N_{D,r}} -\log D_\phi(x_i) + \frac{1}{N_{D,f}} \sum_j^{N_{D,f}} -\log(1-D_\phi(G_\theta(z_j)))$$

# The Discriminator - Binary Classifier of inputs: Real or Fake?



$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \underbrace{\mathbb{E}_{x\sim p_\theta(x)}}\left[\log(1 - D_\phi(x))\right]$$

$$\phi' = \arg\max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1 - D_\phi(\overbrace{G_\theta(z)}))\right]$$

$$\phi' = \arg\min_\phi \mathbb{E}_{x\sim p_{data}(x)}\underbrace{\left[-\log D_\phi(x)\right]}_{\mathcal{L}_{D,r}(x)} + \mathbb{E}_{z\sim p(z)}\underbrace{\left[-\log(1 - D_\phi(G_\theta(z)))\right]}_{\mathcal{L}_{D,f}(z)}$$

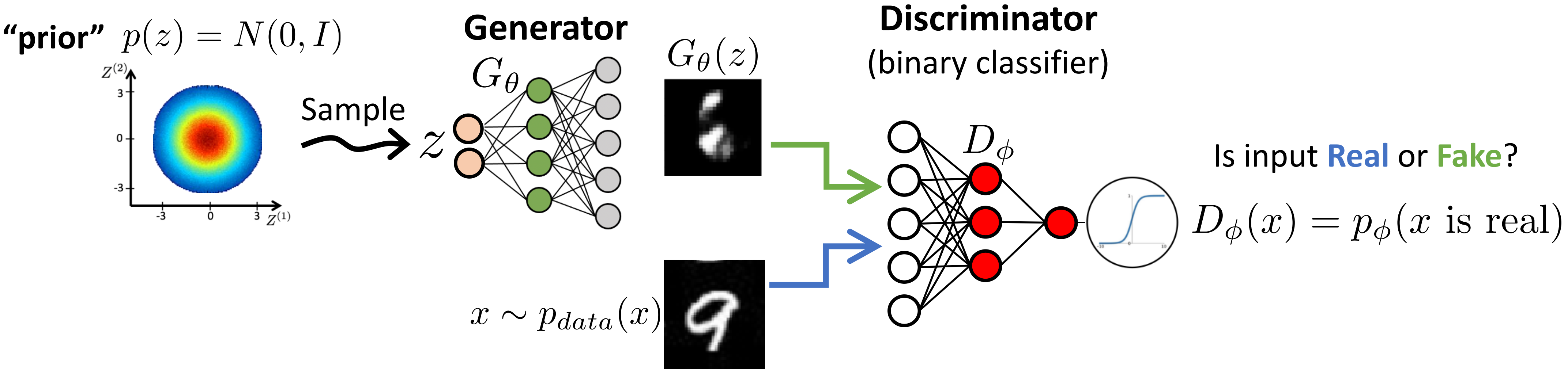
Note-1: The term over fake samples is **also a function of the Generator** parameters $\theta$

Note-2: The generated output G(z) **is “just” another activation map**, on top of which we connect the Discriminator.

Therefore, we **can backpropagate through the generated output G(z).**

We can use this to alter parameters of G to change this term in the opposite direction and **confuse** the Discriminator!

# Adversarial Training



Is input Real or Fake?

$D_\phi(x) = p_\phi(x \text{ is real})$

$$\{\theta', \phi'\} = \arg\min_\theta \max_\phi \mathbb{E}_{x\sim p_{data}(x)} [\log D_\phi(x)] + \underbrace{\mathbb{E}_{x\sim p_\theta(x)}} [\log(1 - D_\phi(x))]$$

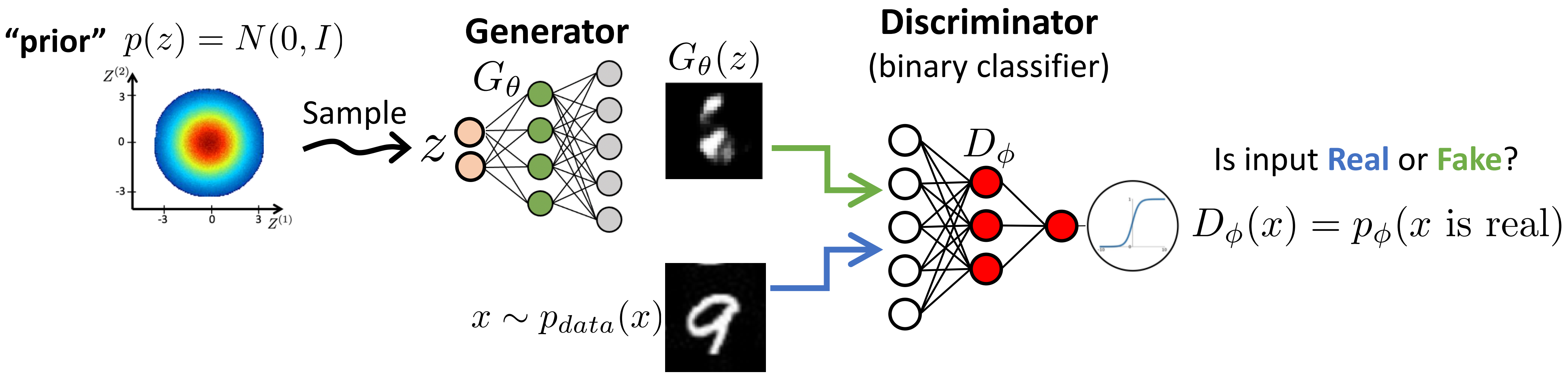
$$\{\theta', \phi'\} = \arg\min_\theta \max_\phi \underbrace{\mathbb{E}_{x\sim p_{data}(x)} [\log D_\phi(x)] + \mathbb{E}_{z\sim p(z)} [\log(1 - D_\phi(\overbrace{G_\theta(z)}))]}_{J_{GAN}(\theta, \phi)}$$

Objective function of the Min-Max game

**Two-player Min-Max game** for optimization (from Game Theory):

- For given (fixed) G, D tries to **maximize** D’s accuracy of separating Reals from Fakes
- For given (fixed) D, G tries to **minimize** D’s accuracy of separating Reals from Fakes
- Therefore called “adversarial”.

# Adversarial Training: Loss of G



$$\{\theta', \phi'\} = \arg\min_\theta \max_\phi \mathbb{E}_{x \sim p_{data}(x)}[\log D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

For **fixed Discriminator** parameters $\phi$ , we **train Generator** parameters $\theta$:

$$\theta' = \arg\min_\theta \mathbb{E}_{x \sim p_{data}(x)}[\log D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_\phi(G_{\underbrace{\theta}_{\text{Function of } \theta}}(z)))]$$

$$\theta' = \arg\min_\theta \mathbb{E}_{z \sim p(z)}\underbrace{[\log(1 - D_\phi(G_\theta(z)))]}_{\mathcal{L}_G(z)}$$

Loss (theoretical) of Generator

**In practice**, SGD with a batch of fake samples (size $N_G$):

$$\theta' = \arg\min_\theta \frac{1}{N_G} \sum_i^{N_G} \log(1 - D_\phi(G_\theta(z_i)))$$

# Training GANs Algorithm (Theoretical $L_G$)

```
$\theta \leftarrow$ random initialization
$\phi \leftarrow$ random initialization
for $t = 1$ to $T$ do:
    for $k = 1$ to $K$ do:   # Train D for K iterations of SGD (often K=1)
        $\{x_i\}_1^{N_{D,r}}$, sampling $x_i \sim X_{train}$
        $\{z_i\}_1^{N_{D,f}}$, sampling $z_i \sim N(0, I)$
        $L_D(\phi,\theta) = L_{D,r}(\phi) + L_{D,f}(\theta,\phi) = -\frac{1}{N_{D,r}}\sum_i^{N_{D,r}} \log D_\phi(x_i) - \frac{1}{N_{D,f}}\sum_i^{N_{D,f}} \log(1 - D_\phi(G_\theta(z_i)))$
        $\phi \leftarrow \phi - \alpha \frac{\partial L_D(\phi,\theta)}{\partial \phi}$
    # Train G for 1 iteration of SGD
    $\{z_i\}_1^{N_G}$, where $z_i \sim N(0, I)$
    $L_G(\phi,\theta) = \frac{1}{N_G}\sum_i^{N_G} \log(1 - D_\phi(G_\theta(z_i)))$
    $\theta \leftarrow \theta - \beta \frac{\partial L_G(\phi,\theta)}{\partial \theta}$
return $\theta, \phi$
```

Opposites

# How does adversarial training achieve $p_\theta(x) \approx p_{data}(x)$?

$$\{\theta', \phi'\} = \min_\theta \max_\phi \underbrace{\mathbb{E}_{x\sim p_{data}(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{x\sim p_\theta(x)}\left[\log(1 - D_\phi(x))\right]}_{J_{GAN}(\theta, \phi)}$$

Objective function of the min-max game

It has been proven that if we assume D has infinite capacity (can learn anything), then for given $\theta$ (G params), **optimal D** is:

$$D_{\phi^*}(x) = \frac{p_{data}(x)}{p_{data}(x) + p_\theta(x)}$$

*Proof: Goodfellow et al, "Generative Adversarial Nets", 2014, sec 4.1.*

Then, by substituting $D_{\phi'}(x)$ to the GAN's total objective function $J_{GAN}(\theta, \phi)$ we have:

$$\begin{aligned} J_{GAN}(\theta, \phi^*) &= \mathbb{E}_{x\sim p_{data}(x)}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_\theta(x)}\right] + \mathbb{E}_{x\sim p_\theta(x)}\left[\log(1 - \frac{p_{data}(x)}{p_{data}(x) + p_\theta(x)})\right] \\ &= \mathbb{E}_{x\sim p_{data}(x)}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_\theta(x)}\right] + \mathbb{E}_{x\sim p_\theta(x)}\left[\log(\frac{p_\theta(x)}{p_{data}(x) + p_\theta(x)})\right] \\ &= \underbrace{2D_{JS}\left[p_{data}(x)||p_\theta(x)\right]}_{\text{Jensen-Shannon divergence between } p_{data}(x) \text{ and } p_\theta(x)} - 2\log 2 \end{aligned}$$

$$D_{JS}\left[p_{data}||p_\theta\right] = 0 \Leftrightarrow p_\theta(x) = p_{data}(x)$$

# How does adversarial training achieve $p_\theta(x) \approx p_{data}(x)$?

The previous theoretical results means:

*If* D was perfectly optimized at every SGD iteration before updating G,
then minimizing $J_{GAN}$ with $G_\theta$ *would* lead to minimization of JS divergence.

This in turn means the generator *would* replicate learn the density of real data "perfectly",
i.e. $p_\theta(x) \approx p_{data}(x)$

But, in practice:
a) we don't optimize D perfectly in each SGD iteration (only K updates, and SGD local minima),
b) we often use another loss for G, rather than what we showed before (which was aligned with theory)…

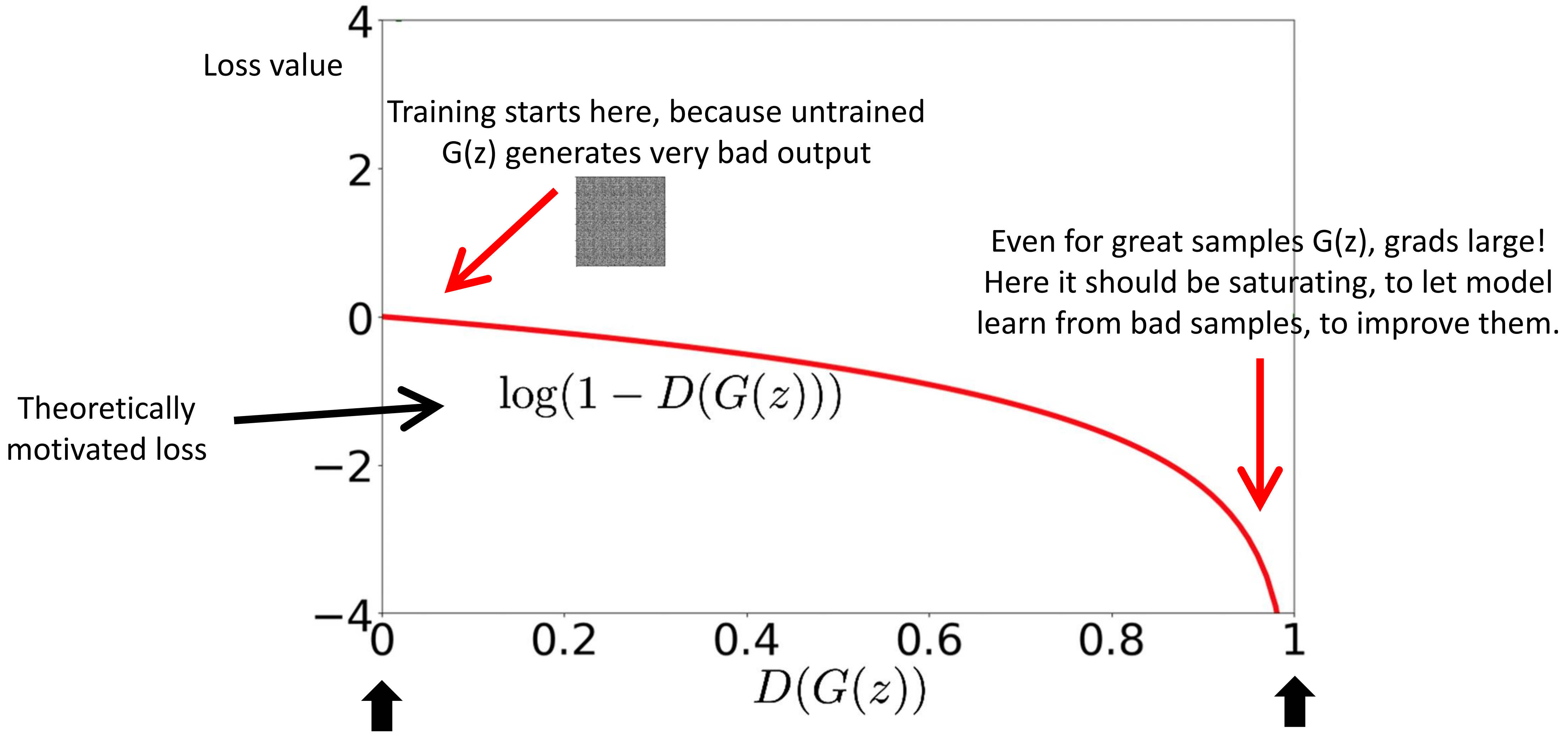
Regardless, this theoretical result gives a nice intuition about how GANs work.

# Training GANs Algorithm (Theoretical $L_G$)

```
$\theta \leftarrow$ random initialization
$\phi \leftarrow$ random initialization
for $t = 1$ to $T$ do:
    for $k = 1$ to $K$ do:   # Train D for K iterations of SGD (often K=1)
        $\{x_i\}_1^{N_{D,r}}$, sampling $x_i \sim X_{train}$
        $\{z_i\}_1^{N_{D,f}}$, sampling $z_i \sim N(0, I)$
        $L_D(\phi, \theta) = L_{D,r}(\phi) + L_{D,f}(\theta, \phi) = -\frac{1}{N_{D,r}} \sum_i^{N_{D,r}} \log D_\phi(x_i) - \frac{1}{N_{D,f}} \sum_i^{N_{D,f}} \log(1 - D_\phi(G_\theta(z_i)))$
        $\phi \leftarrow \phi - \alpha \frac{\partial L_D(\phi, \theta)}{\partial \phi}$
    # Train G for 1 iteration of SGD
    $\{z_i\}_1^{N_G}$, where $z_i \sim N(0, I)$
    $L_G(\phi, \theta) = \frac{1}{N_G} \sum_i^{N_G} \log(1 - D_\phi(G_\theta(z_i)))$
    $\theta \leftarrow \theta - \beta \frac{\partial L_G(\phi, \theta)}{\partial \theta}$
return $\theta, \phi$
```

Opposites

# Losses for training the Generator G



Loss value

Training starts here, because untrained G(z) generates very bad output

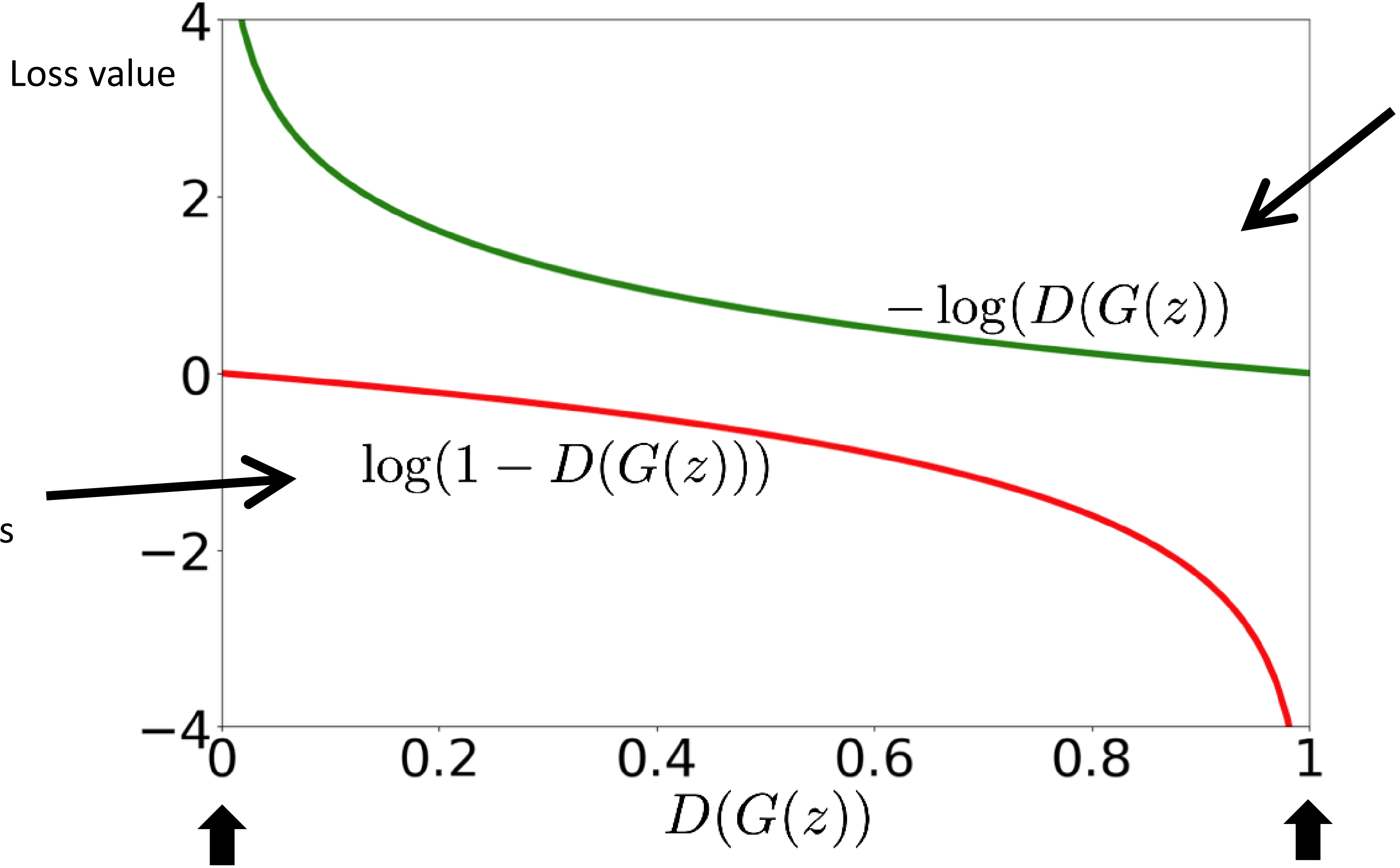
Even for great samples G(z), grads large! Here it should be saturating, to let model learn from bad samples, to improve them.

Theoretically motivated loss

D is certain that input G(z) is Fake (correctly)

D is certain that input G(z) is Real (it got tricked by G)

# Losses for training the Generator G



Commonly used in practice

Theoretically motivated loss

D is certain that input G(z) is Fake (correctly)

D is certain that input G(z) is Real (it got tricked by G)

# Losses of G: are they comparable?

**Theoretically motivated loss for G** (min max game minimizing JS divergence):

$$\mathcal{L}_G(z) = \log(1 - D_\phi(G_\theta(z)))$$

$$\theta' = \arg\min_\theta \mathbb{E}_{z\sim p(z)} \left[\underbrace{\log(1 - D_\phi(G_\theta(z)))}_{1 - \underbrace{D_\phi(G_\theta(z_j))}_{D_\phi(G_\theta(z_j))}}\right]$$

Minimize the log

Minimize (because log convex)

Maximize

**Practical loss for G:**

$$\mathcal{L}_G(z) = -\log D_\phi(G_\theta(z))$$

$$\theta' = \arg\min_\theta \mathbb{E}_{z\sim p(z)} \left[\underbrace{-\log D_\phi(G_\theta(z))}_{\log \underbrace{D_\phi(G_\theta(z))}_{D_\phi(G_\theta(z))}}\right]$$

Minimize the -log

Maximize the log

Maximize

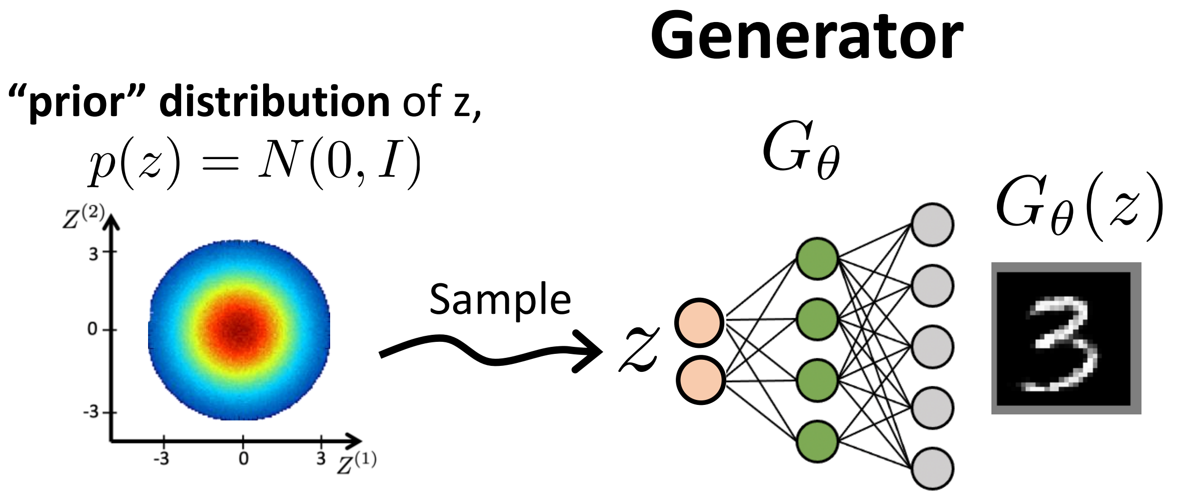
They both try to maximize the probability that D will think fake samples are real. But using **different gradients to get there**.

# Training GANs Algorithm (Practical $L_G$)

```
$\theta \leftarrow$ random initialization
$\phi \leftarrow$ random initialization
for $t = 1$ to $T$ do:
    for $k = 1$ to $K$ do:   # Train D for K iterations of SGD (often K=1)
        $\{x_i\}_1^{N_{D,r}}$, sampling $x_i \sim X_{train}$
        $\{z_i\}_1^{N_{D,f}}$, sampling $z_i \sim N(0, I)$
        $L_D(\phi, \theta) = L_{D,r}(\phi) + L_{D,f}(\theta, \phi) = -\frac{1}{N_{D,r}} \sum_i^{N_{D,r}} \log D_\phi(x_i) - \frac{1}{N_{D,f}} \sum_i^{N_{D,f}} \log(1 - D_\phi(G_\theta(z_i)))$
        $\phi \leftarrow \phi - \alpha \frac{\partial L_D(\phi, \theta)}{\partial \phi}$
    # Train G for 1 iteration of SGD
    $\{z_i\}_1^{N_G}$, where $z_i \sim N(0, I)$
    $L_G(\phi, \theta) = \frac{1}{N_G} \sum_i^{N_G} - \log D_\phi(G_\theta(z_i))$
    $\theta \leftarrow \theta - \beta \frac{\partial L_G(\phi, \theta)}{\partial \theta}$
return $\theta, \phi$
```

“Practical” version of Generator’s loss.

# After training: G maps any point of z from prior p(z) to a "realistic" x=G(z), where all G(z) form $p_\theta(x) \approx p_{data}(x)$



**Generator**

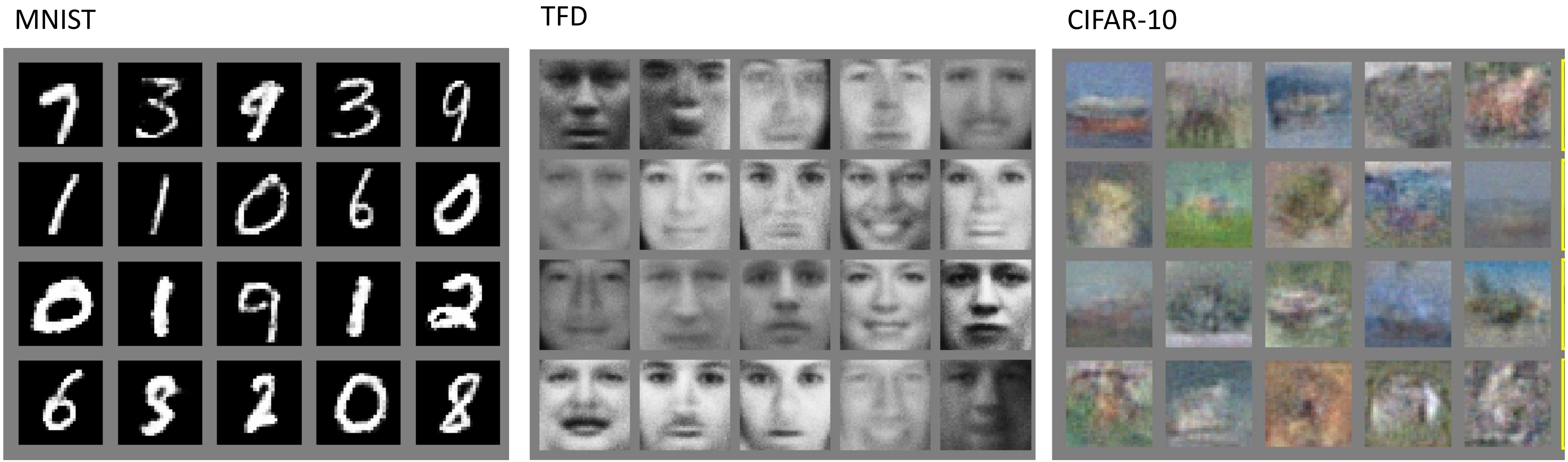
**"prior" distribution** of z,
$p(z) = N(0, I)$

Sample

**To generate new data from a GAN**:
Step 1. sample *z* from prior *N(0,I)*
Step 2. "Decode" using *G*, obtaining *G(z)*, the generated sample.

The process is "exactly" the same as if generating using prior and decoder of a trained VAE.
*(Note: Discriminator is not used for generation)*

# Generated samples from a basic GAN - 2014



MNIST

TFD

CIFAR-10

From: Goodfellow et al, ”Generative Adversarial Nets”, 2014

# Deep Convolutional GAN (DCGAN) - 2016

Radford et al, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ICLR 2016

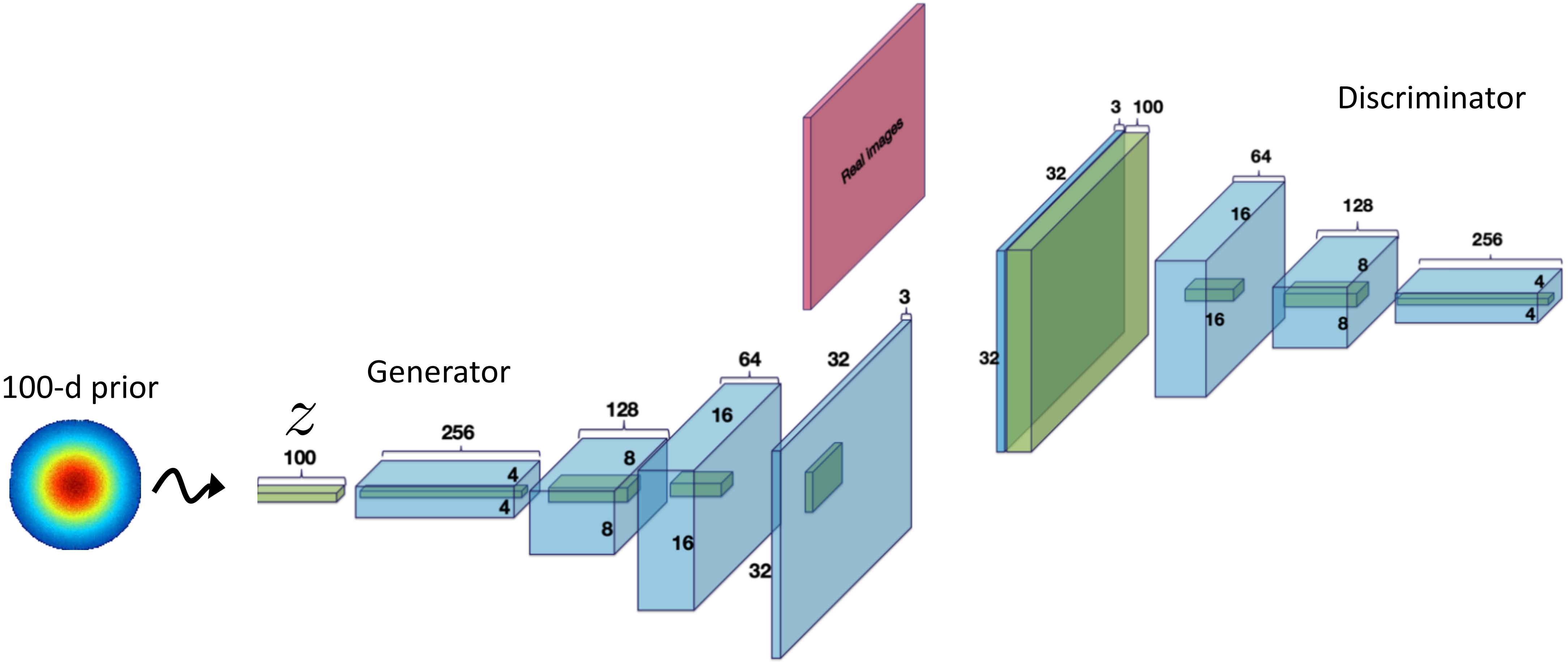
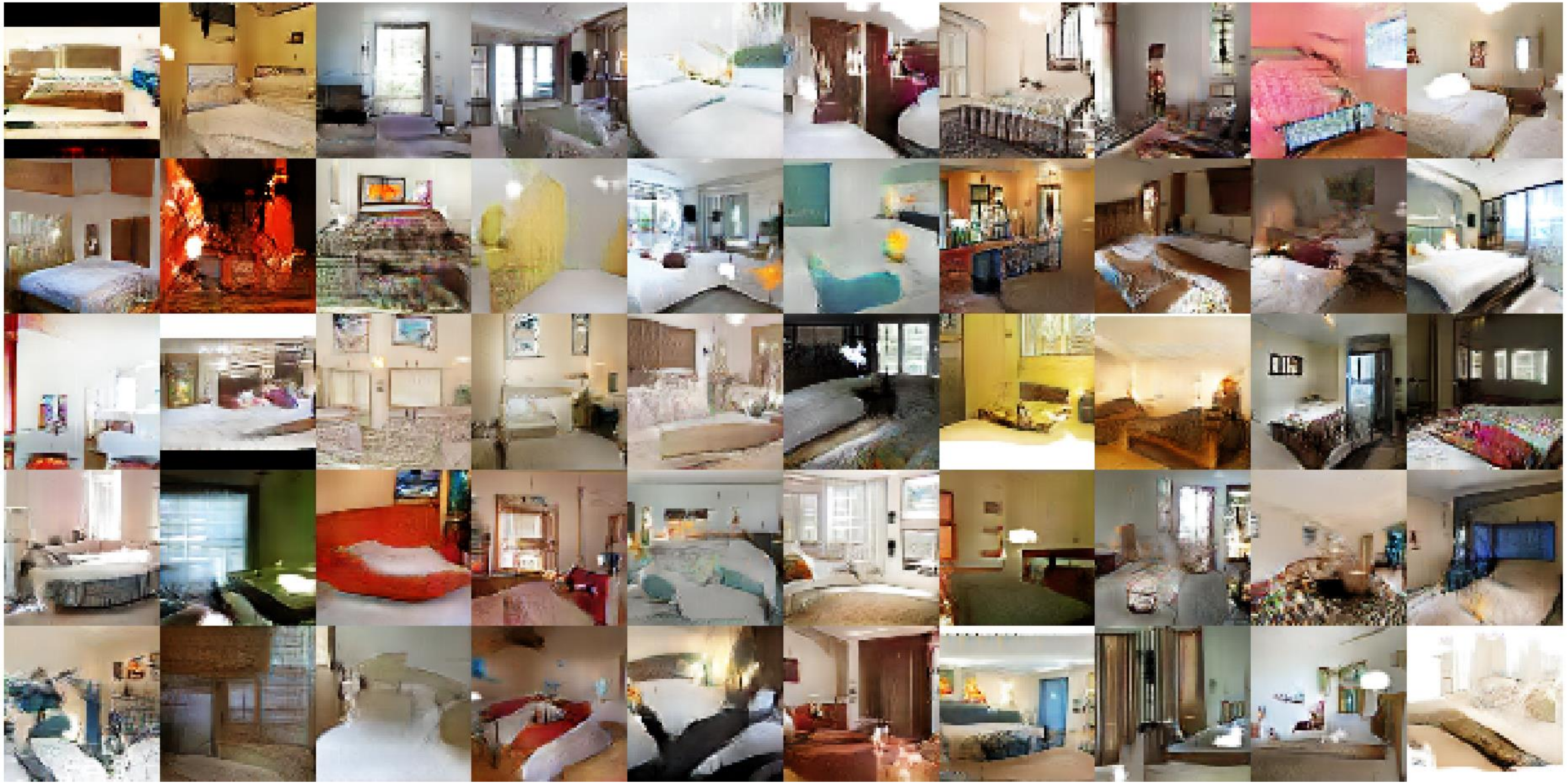


Image modified from Satrfall6 ([link](link))

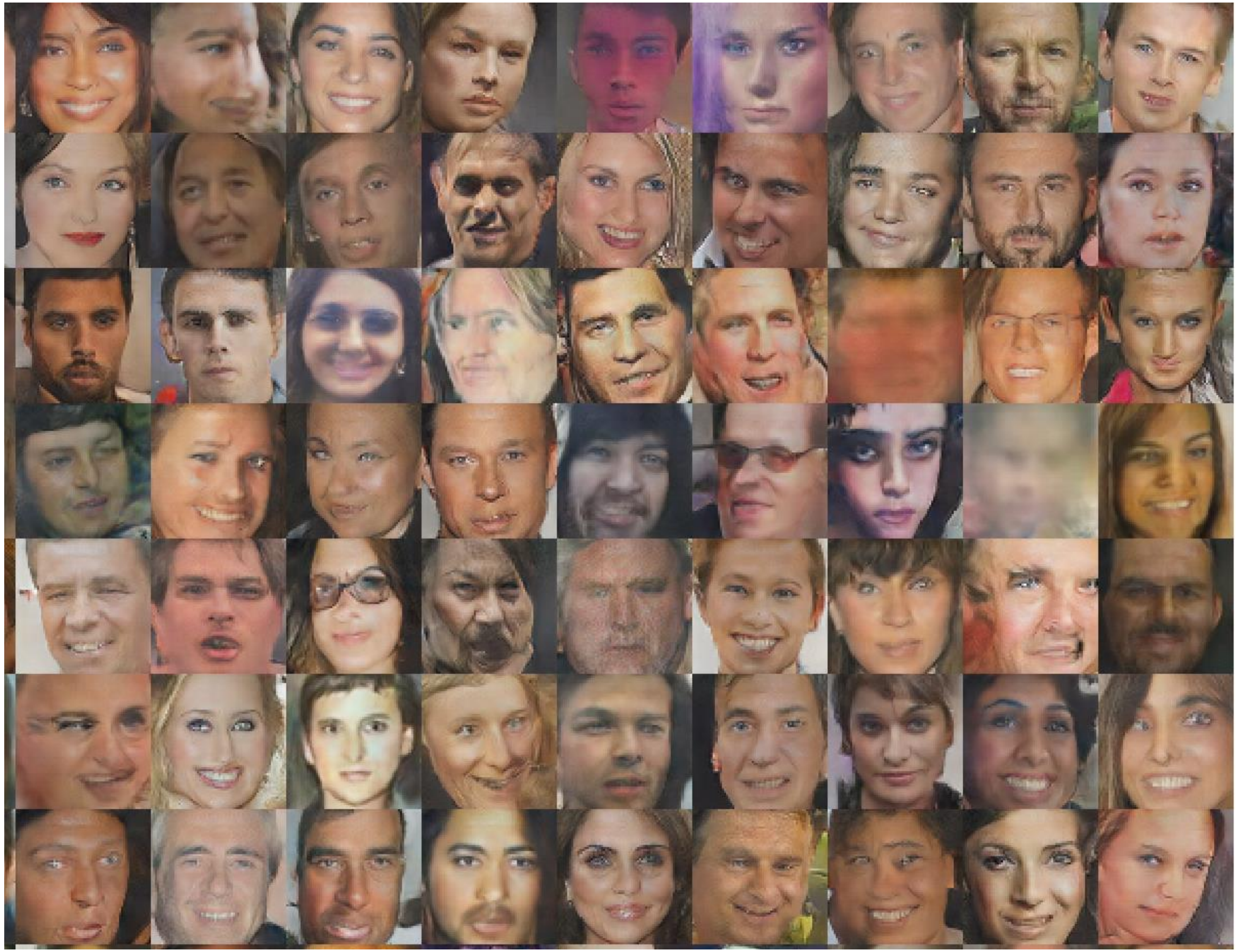
# Deep Convolutional GAN (DCGAN) - 2016

Radford et al, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ICLR 2016



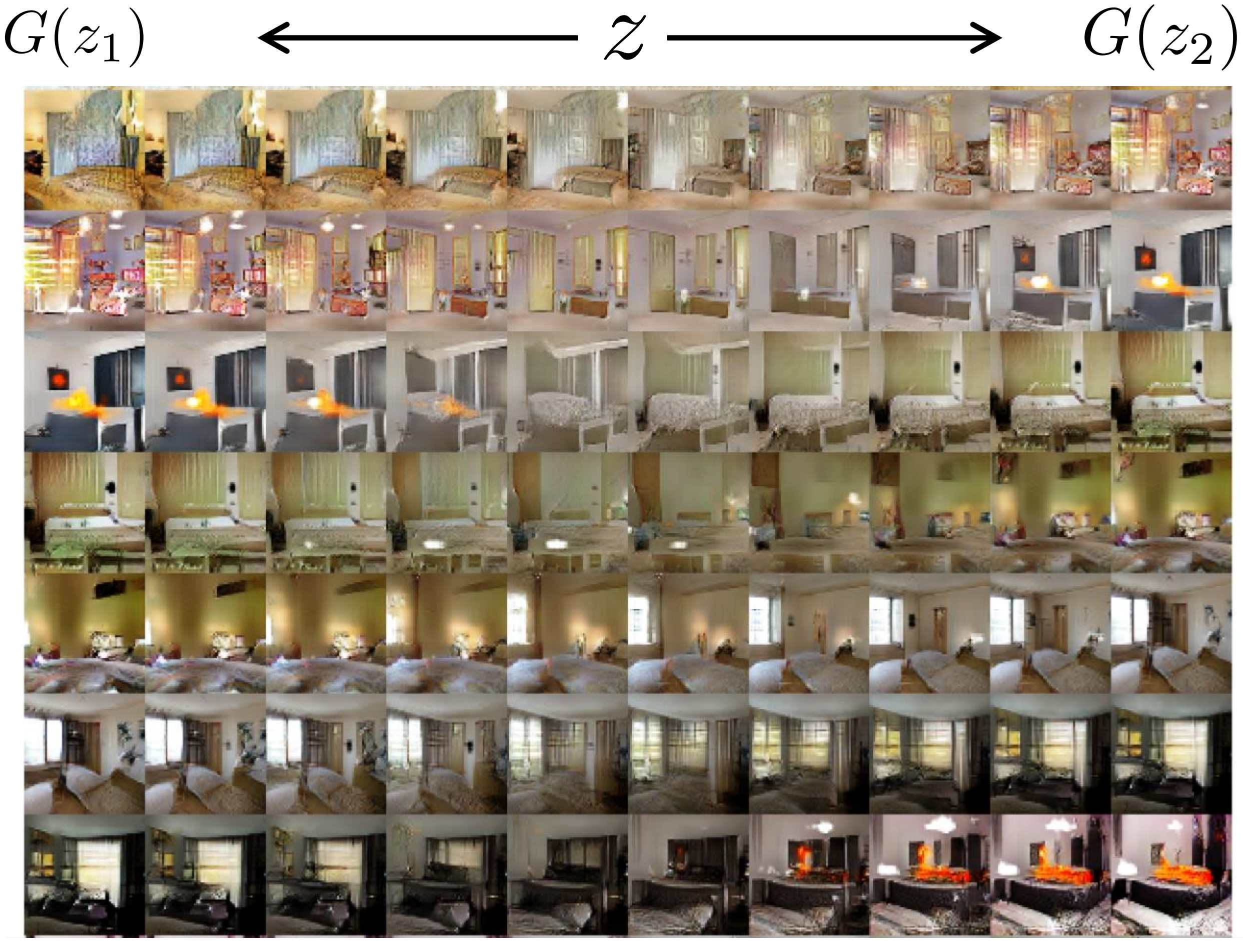
# Deep Convolutional GAN (DCGAN) - 2016

Radford et al, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ICLR 2016



# DCGAN: Interpolation by decoding z between 2 values

Radford et al, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ICLR 2016



$z_1$, $z_2$ are different (randomly sampled) per row

Observe that results change ”smoothly” (similar to VAEs).

There is no theoretical explanation why this happens in GANs (in contrast to VAEs), but it is often empirically observed.

# BigGAN - 2019

Brock et al, Large scale GAN training for high fidelity natural image synthesis, 2019

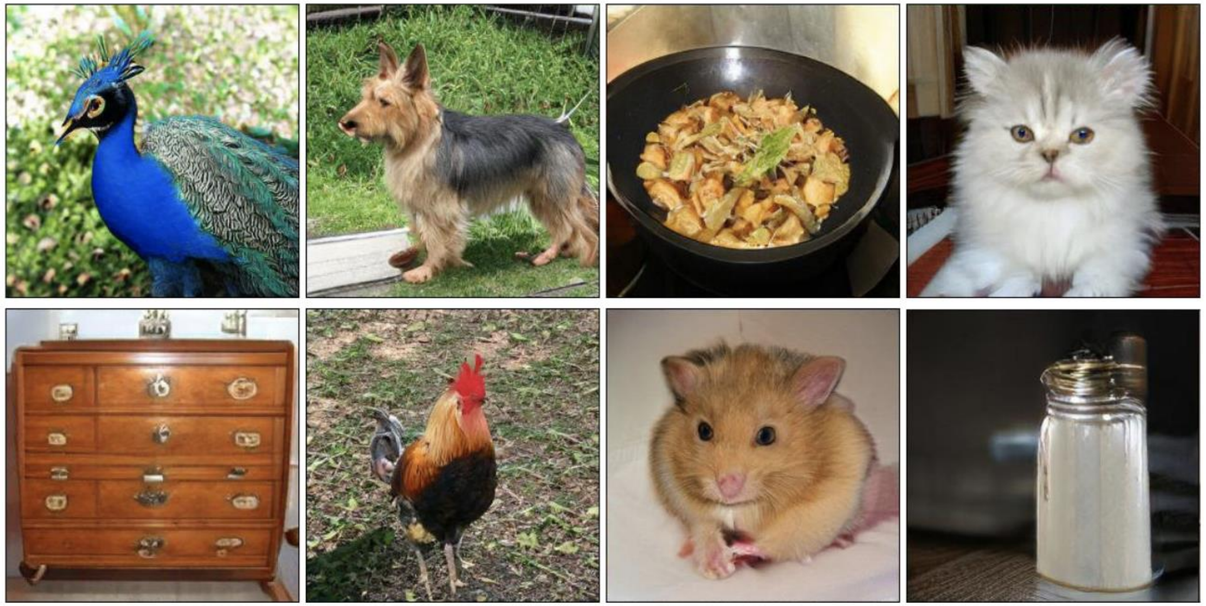


Figure 5: Samples generated by our BigGAN model at 256×256 resolution.

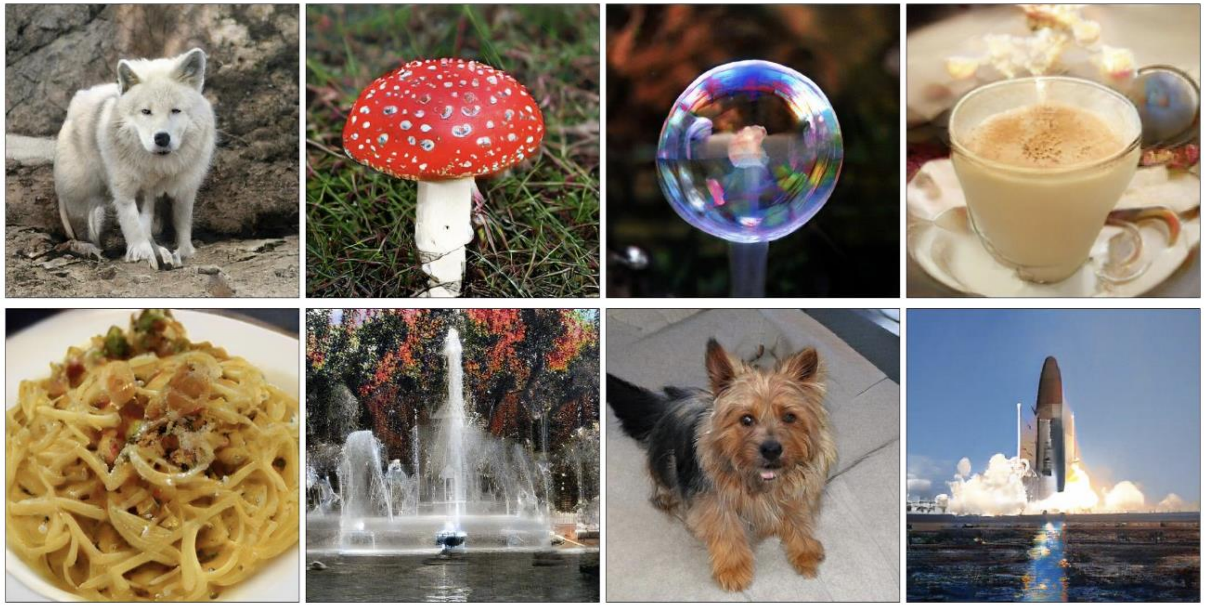


Figure 6: Samples generated by our BigGAN model at 512×512 resolution.

# Thank you very much