Generative Adversarial Network

Some slides were adated/taken from various sources, including Andrew Ng’s Coursera Lectures, CS231n: Convolutional Neural Networks for Visual Recognition lectures, Stanford University CS Waterloo Canada lectures, Aykut Erdem, et.al. tutorial on Deep Learning in Computer Vision, Ismini Lourentzou's lecture slide on "Introduction to Deep Learning", Ramprasaath's lecture slides, and many more. We thankfully acknowledge them. Students are requested to use this material for their study only and NOT to distribute it.

Outline

- Basics of GAN
- Training
- Cost function
- Drawbacks
- Different GAN architectures

GAN

- **Generative**
  - Learn a generative model
- **Adversarial**
  - Trained in an adversarial setting
- **Networks**
  - Use Deep Neural Networks

Generative Adversarial Network

- Definition: GAN are deep neural net architectures comprised of two neural networks, competing one against the other (thus adversarial).

- GAN are neural networks that are trained in an adversarial manner to generate data mimicking some distribution.

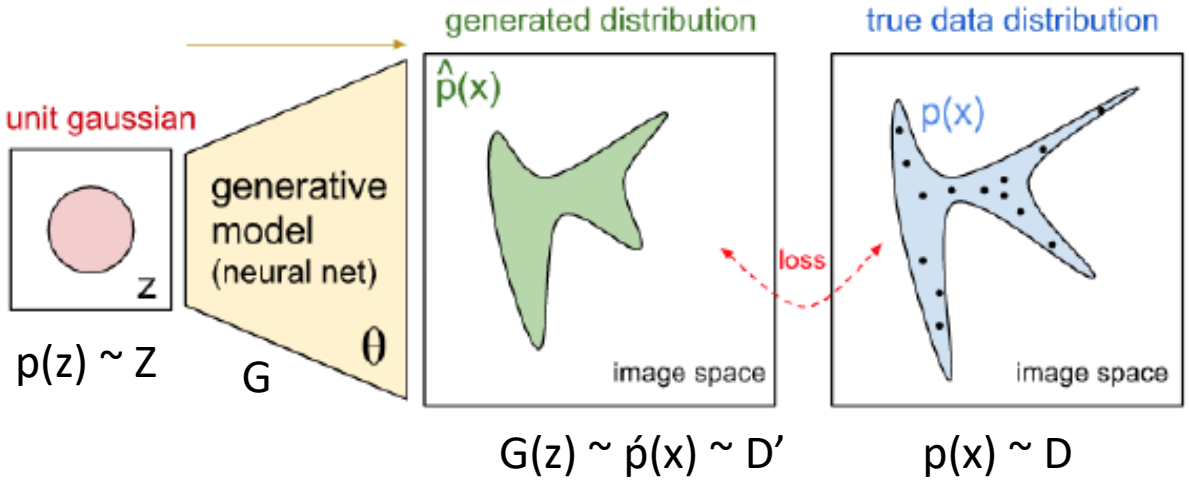
Two classes of models in machine learning

- Discriminative Models: It is learning by discriminating between two classes of data.
  - Classification (e.g. Face is fake or real)

- Generative Model: A generative model G to be trained on training data X sampled from some true distribution D is the one which given some standard random distribution Z produces a distribution D' which is close to D according to some closeness metric, mathematically

  $z \sim Z$ maps to a sample $G(z) \sim D'$

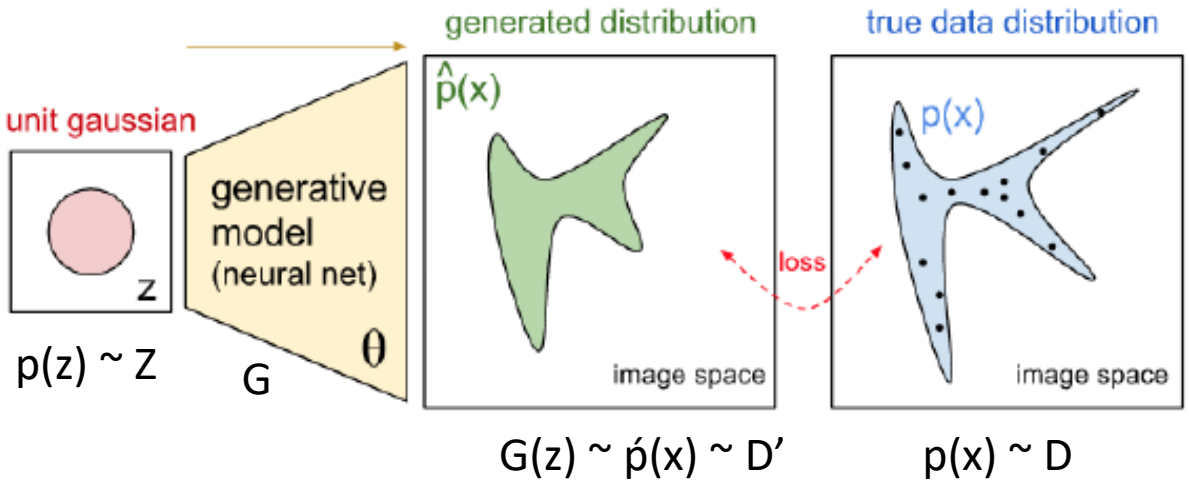
Generative Model



Let we have a set of samples (let say MNIST dataset) which form a distribution D which is not known to us. So, we have known samples but we don't know the distribution of those samples.

Goal: Approximate the distribution D and generate a data distribution (D') as close as possible to D.

How to achieve this goal: Given some standard distribution Z, use a generator which can be used to transfer the distribution Z to D',



Let Z is a random distribution with samples set z (zϵZ), we use a generator to produce some samples (called generative sample G(z)) having distribution D' which is as close as possible to distribution D with respect to some statistical distance (let say L1 norm, L2 norm etc.) . So, mathematically, G(z)~D'

What is G: Feed forward neural network (may be DNN). It produces the generative samples G(z) having distribution D'.

Loosely speaking, G(z) are also MNIST like samples but they are not present within D. They are synthetically generated following distribution D.

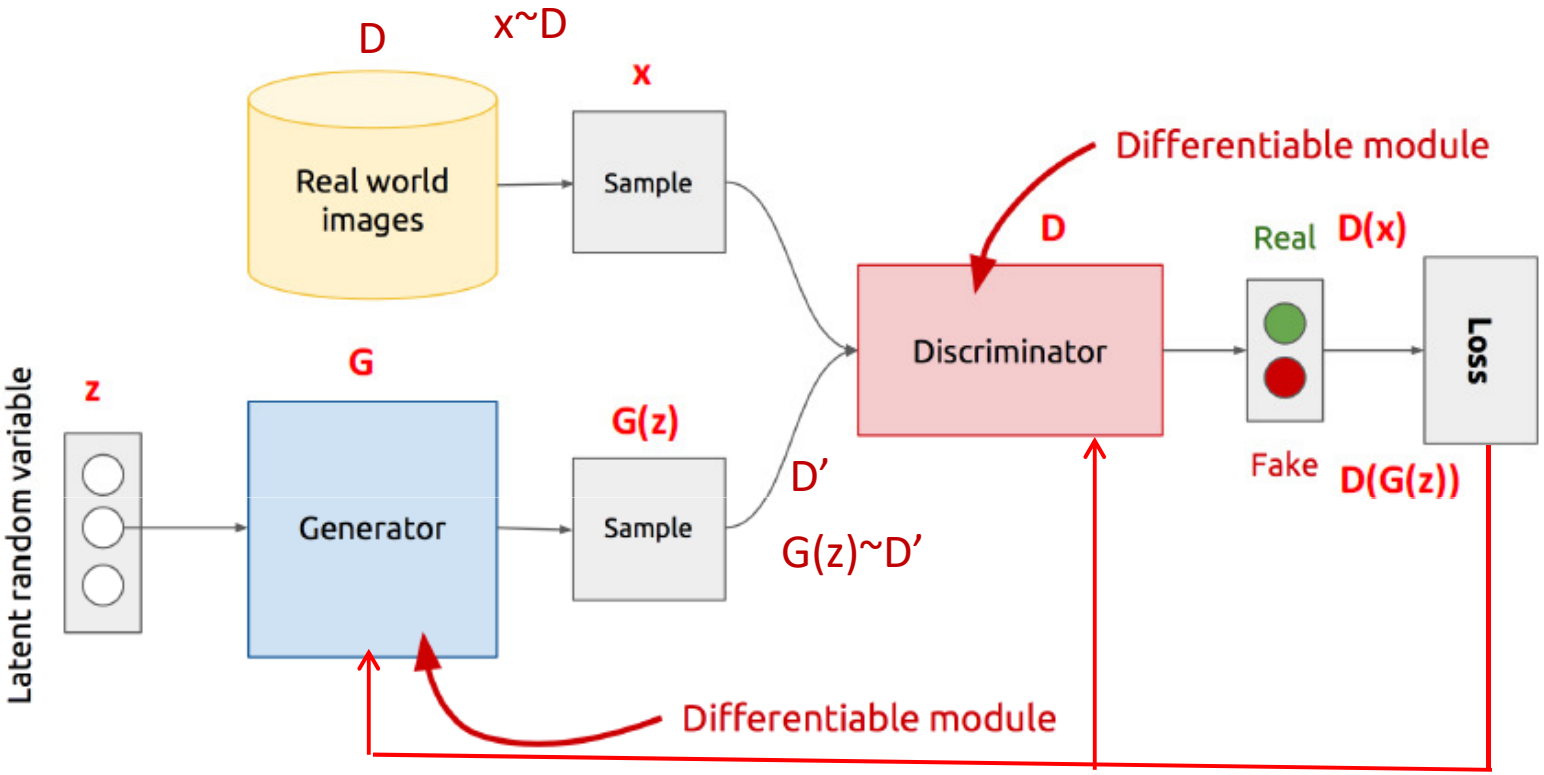
Why Generative Model

- **We've only seen discriminative models so far**
  - Given an image **X**, predict a label **Y**
  - Estimates **P(Y|X)**

- **Discriminative models have several key limitations**
  - Can't model **P(X)**, i.e. the probability of seeing a certain image
  - Thus, can't sample from **P(X)**, i.e. **can't generate new images**

- **Generative models (in general) cope with all of above**
  - Can model **P(X)**
  - Can generate new images

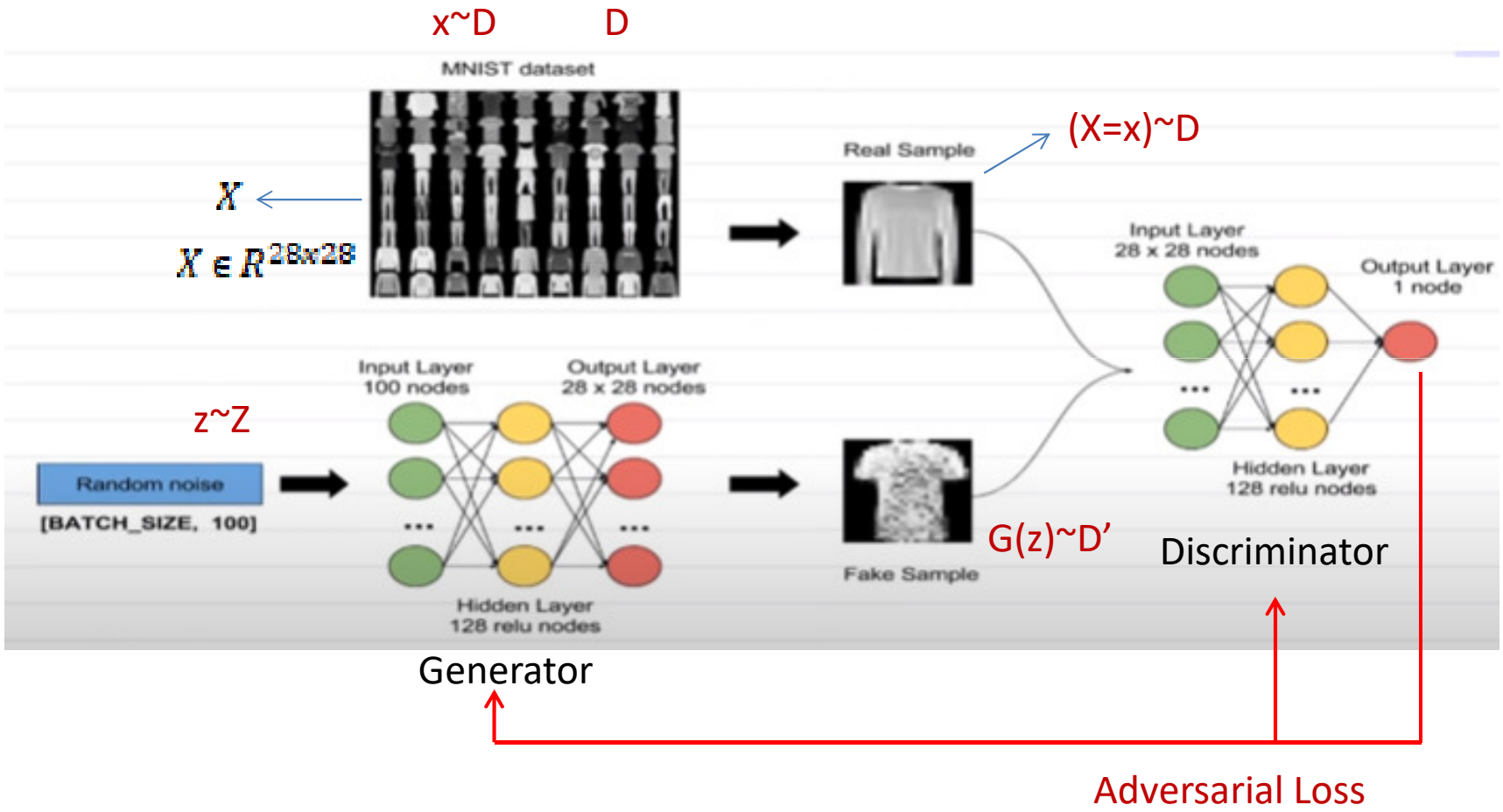
GAN Architecture



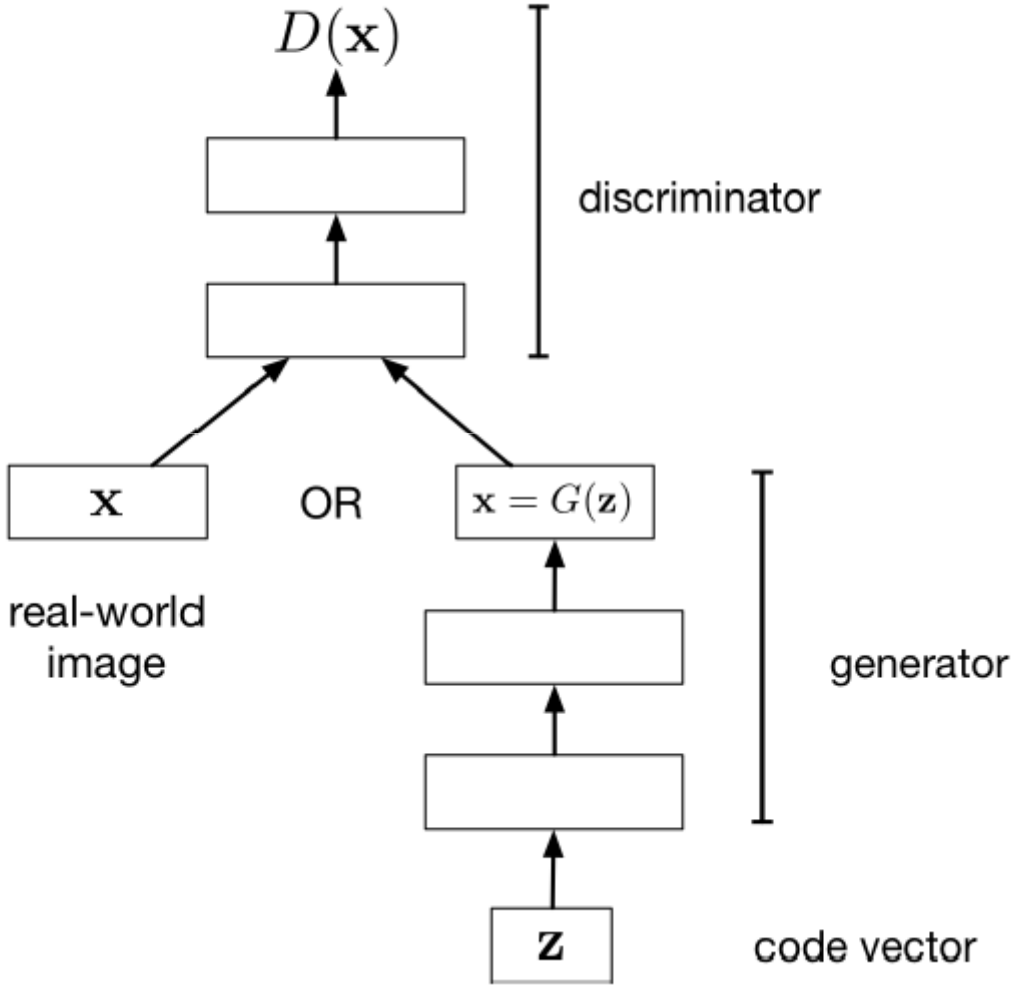
Z is some random noise (Gaussian/Uniform).

Z can be thought as the latent representation of the image.

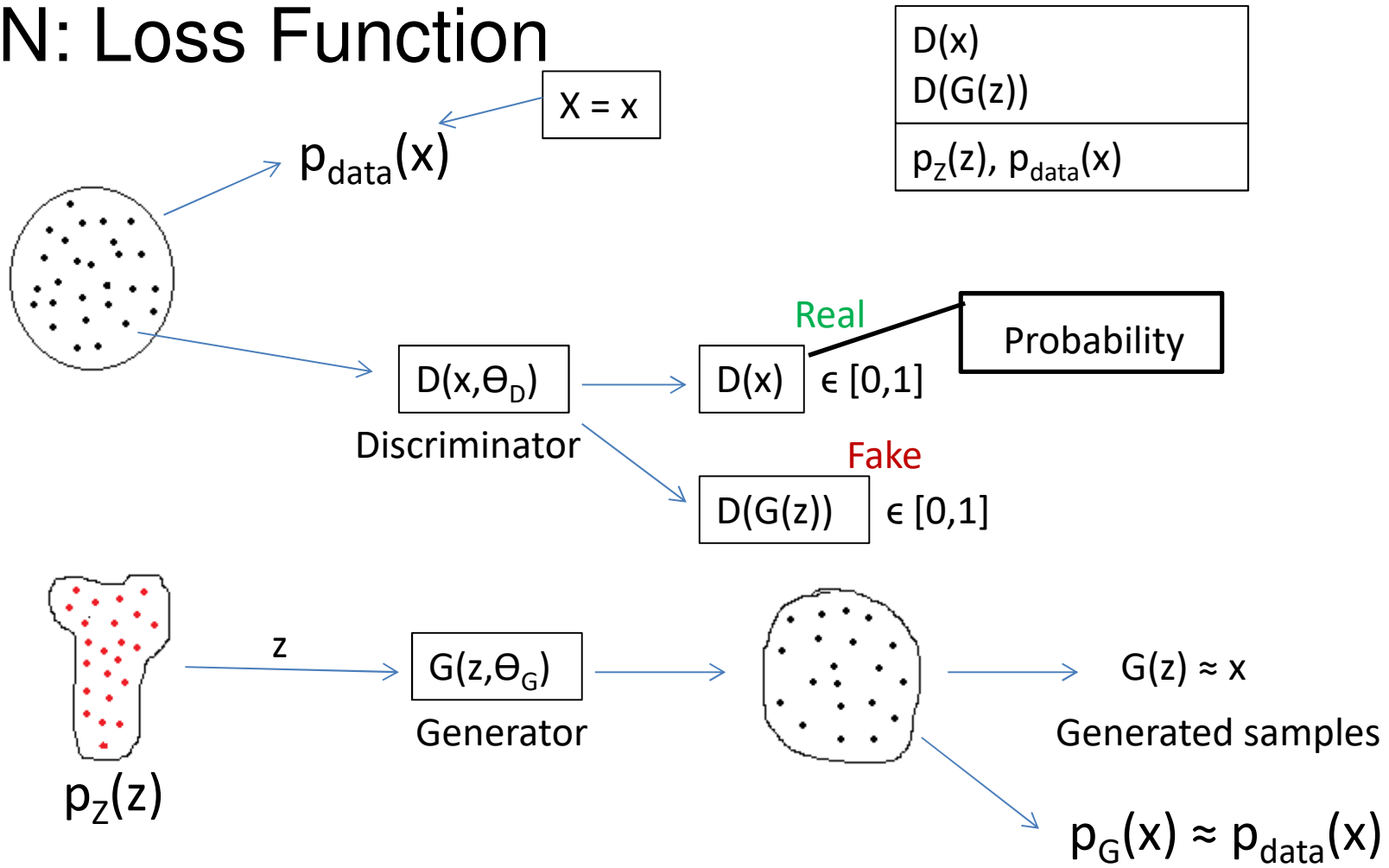
Example: MNIST Fashion



GAN Architecture



GAN: Loss Function



GAN: Loss Function

Binary Cross Entropy: $L(\hat{y}, y) = y log\, \hat{y} + (1 - y) \log (1 - \hat{y})$

The label for the data coming from $p_{data}(x)$ is y=1 & $\hat{y}$ = D(x), so putting them we obtain

L(D(x),1) = log(D(x)) ---- (A)

And the data coming from generator, the label is y=0 & $\hat{y}$ = D(G(z))

So in that case,

L(D(G(z)),0) = (1-0)log(1-D(G(z))) ---- (B)

GAN: Loss Function (Discriminator)

Now, the objective of the discriminator is to correctly classify fake vs the real dataset. For this (A) and (B) should be maximal.

$$L(D(x),1) = \log(D(x)) \text{ ---- (A)}$$

$$L(D(G(z)),0) = (1-0)\log(1-D(G(z))) \text{ ---- (B)}$$

Now, how can we maximize both A and B?

By making D(x) = 1 we can maximize A and

By making D(G(z)) = 0, we can maximize B

$$\max \{\log(D(x)) + \log(1-D(G(z)))\}$$

GAN: Loss Function (Generator)

- The main objective of the Generator is to fool the discriminator i.e. to make D(G(z))=1 (why?)
- For Discriminator, all the values coming from the Generator should have y=0 (Fake).
- So, to fool the Discriminator, Generator's role is to produce such images which looks like real i.e. y=1
- Now $\hat{y}$ = log(1-D(G(z)))
- L(D(G(z)),0) = (1-0)log(1-$\hat{y}$)= (1) log(1-D(G(z)))
- So, to fool the discriminator, we have to make D(G(z))=1
- And how can we do that, by making

$$\min \{\log(D(x)) + \log(1-D(G(z)))\}$$

- The above expression forces to make D(G(z))=1 which we want

GAN: Loss Function

Combining two objective functions, we can write

$$\min_{G}\max_{D} \{log\, D(x) + \log\,(1 - D(G(z))\}$$

The above equation is for one data sample. For all the samples, we have to take the expectation of the above expression. So the final expression become:

$$\min_{G}\max_{D} V(D,G) = \{\underset{x\sim p_{data}(x)}{E}[log\, D(x)] + \underset{z\sim p_z(z)}{E}[\log\,(1 - D(G(z))]\}$$

GoodFellow' 2014

GAN: Loss Function

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Ian Goodfellow et. al, Generative Adversarial Nets, NIPS 2014

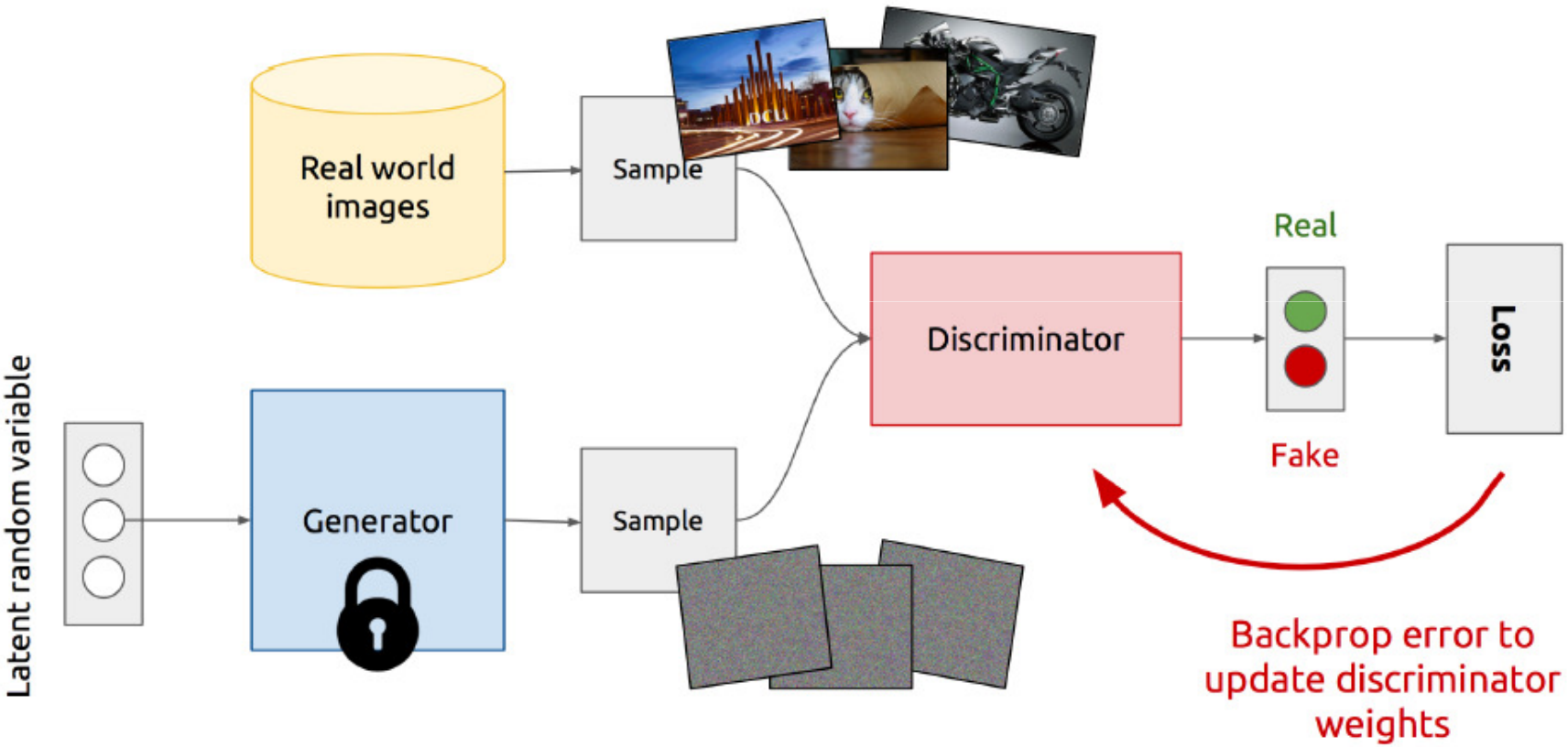
GAN: Formulation

$$\min_G \max_D V(D, G)$$

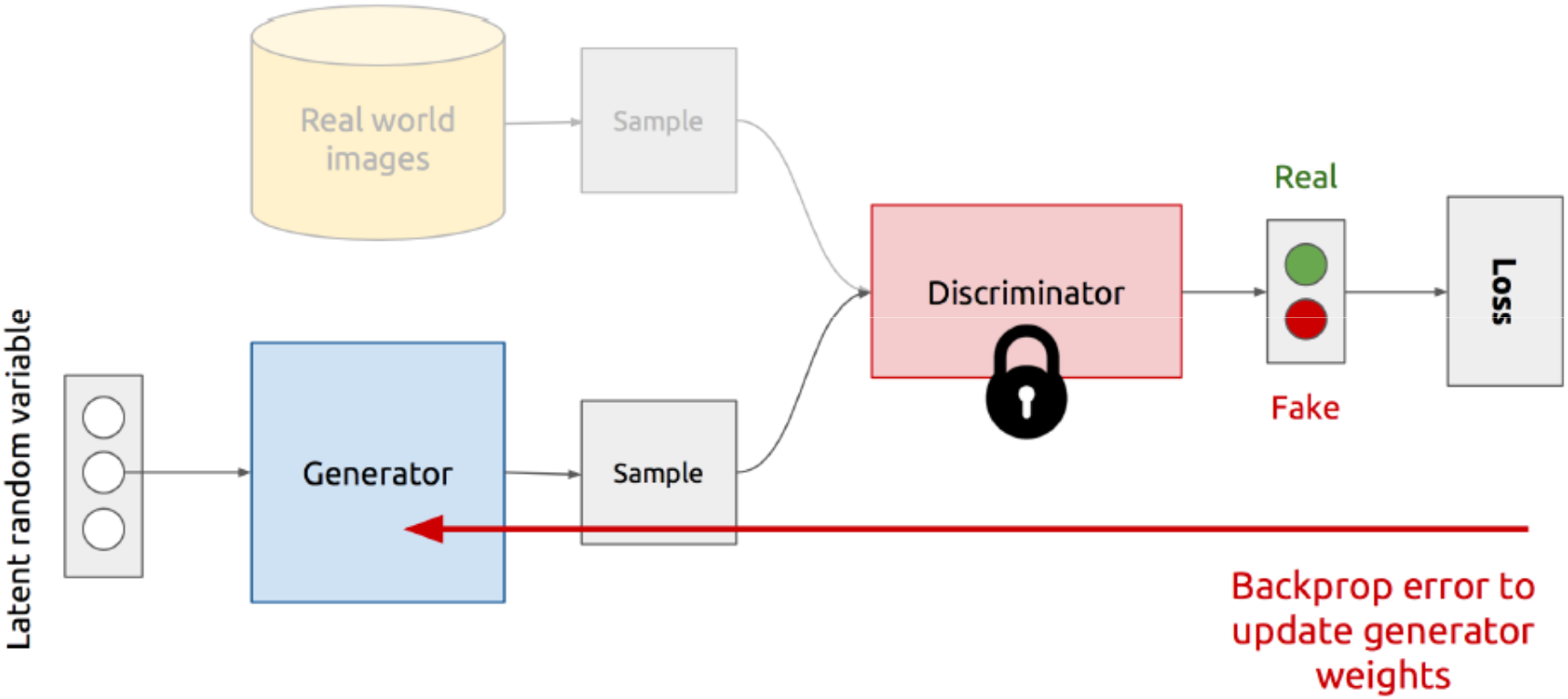
- It is formulated as a **minimax game**, where:
  - The Discriminator is trying to maximize its reward $\boldsymbol{V(D, G)}$
  - The Generator is trying to minimize Discriminator's reward (or maximize its loss)

$$V(D, G) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

Training GAN: Discriminator



Training GAN: Generator



Training GAN:

Update discriminator by Ascending the GD because it's a maximum optimization to be carried out with respect to the D

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

Discriminator updates

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

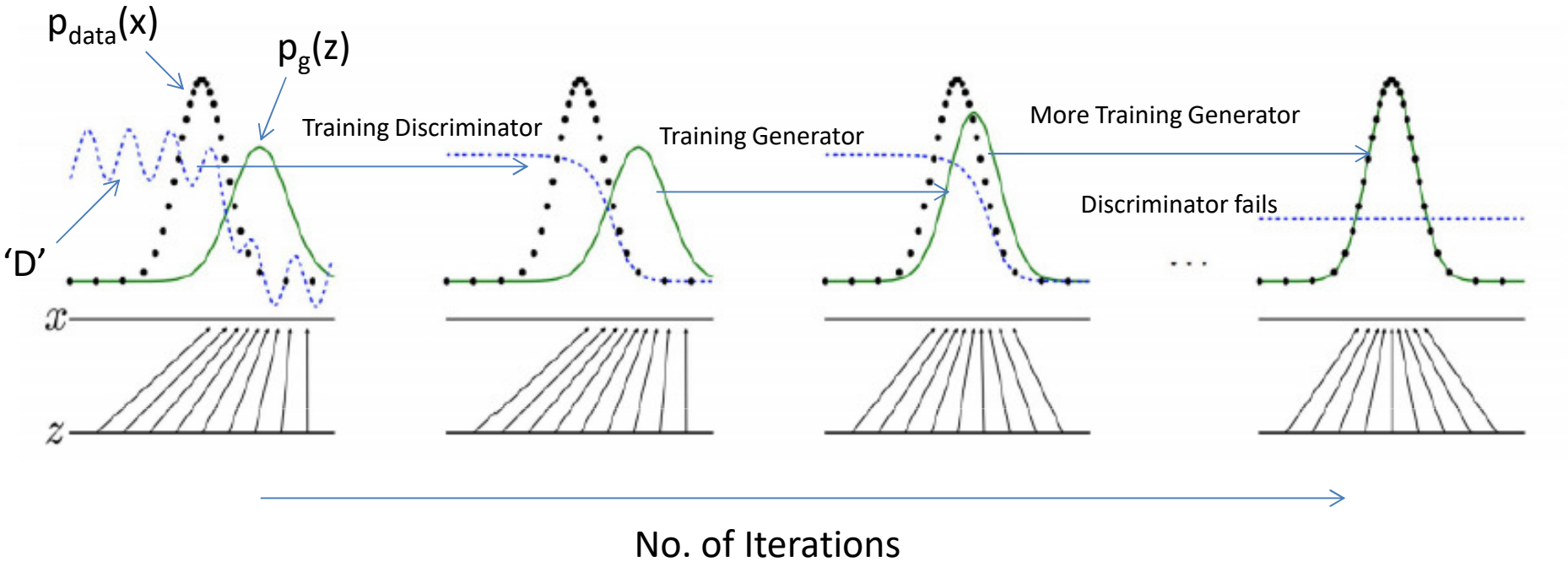
Generator updates

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Ascending the GD because it's a minimum optimization to be carried out with respect to the G

Training GAN:



Vanishing Gradient for Generator

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

$$\nabla_{\theta_G} V(D,G) = \nabla_{\theta_G} \mathbb{E}_{z\sim q(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

- $\nabla_a \log\left(1-\sigma(a)\right) = \frac{-\nabla_a \sigma(a)}{1-\sigma(a)} = \frac{-\sigma(a)\left(1-\sigma(a)\right)}{1-\sigma(a)} = -\sigma(a) = -D\left(G(z)\right)$
- Gradient goes to 0 if D is confident, i.e. $D\left(G(z)\right) \to 0$

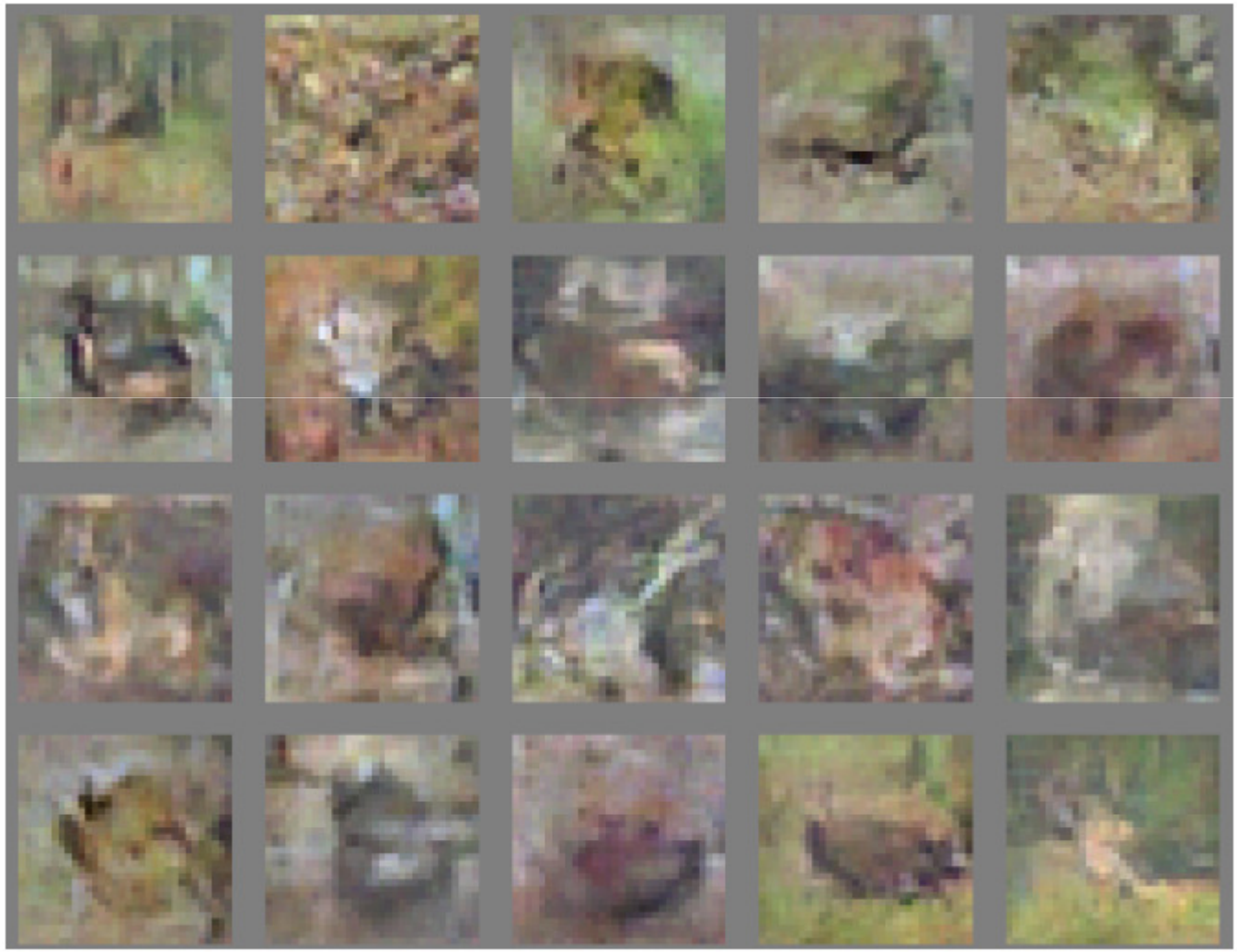
- Minimize $-\mathbb{E}_{z\sim q(z)}\left[\log D\left(G(z)\right)\right]$ for **Generator** instead (keep Discriminator as it is)

  Or maximize $\mathbb{E}_{z\sim q(z)}\left[\log D\left(G(z)\right)\right]$ for **Generator**

Some Examples: Faces

Goodfellow, Ian, et al. **"Generative adversarial nets."** *Advances in neural information processing systems*. 2014.

Some Examples: CIFAR



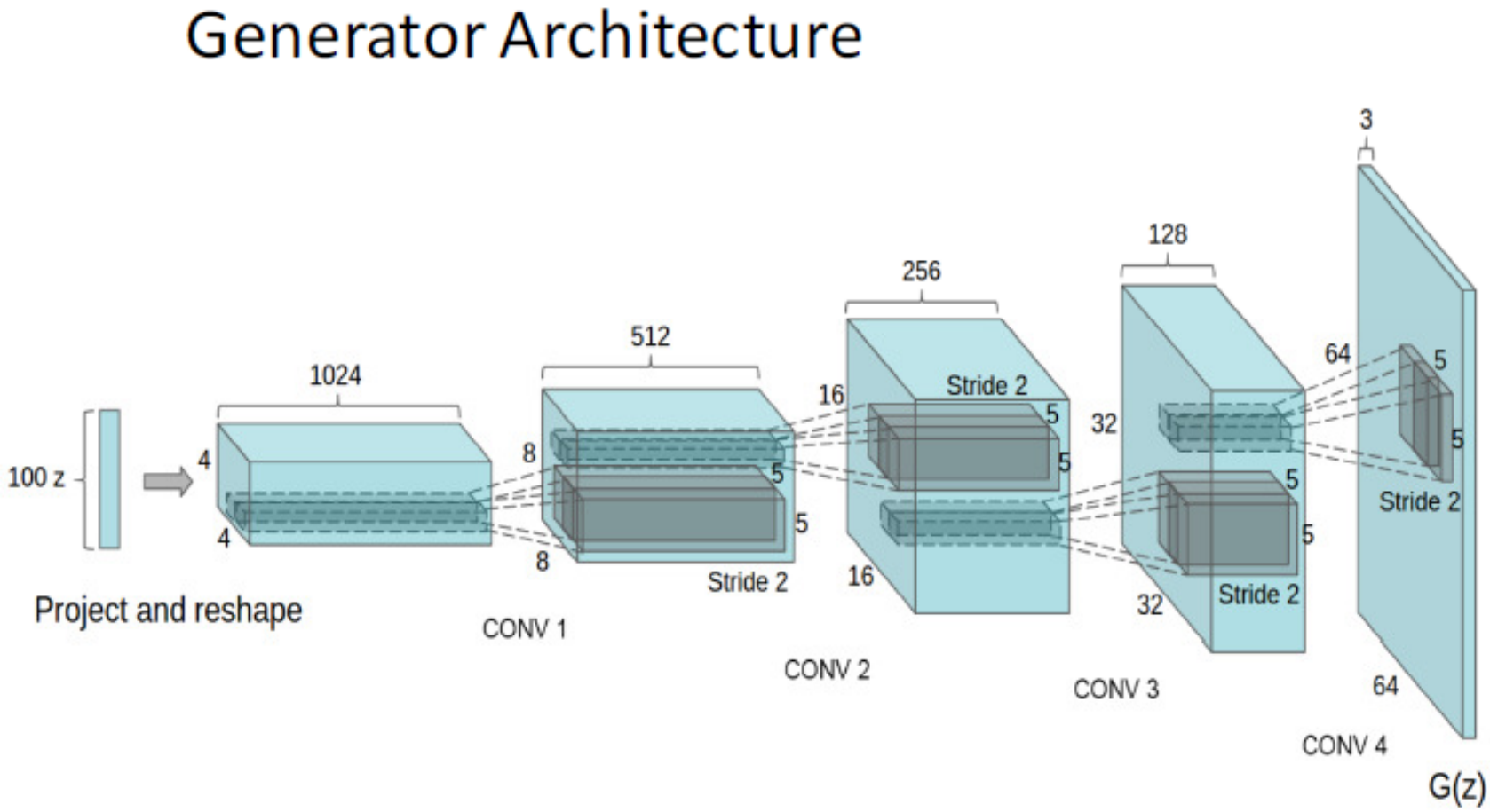
Goodfellow, Ian, et al. **"Generative adversarial nets."** *Advances in neural information processing systems*. 2014.

Deep Convolution GAN (DCGAN)

- Key ideas
  - Replace FC hidden layers with convolutions
  - Generator: Use fractional stride convolution
  - Use batch Normalization after each layer
  - Inside Generator
    - Use ReLU for hidden layers
    - Use TanH for output layer

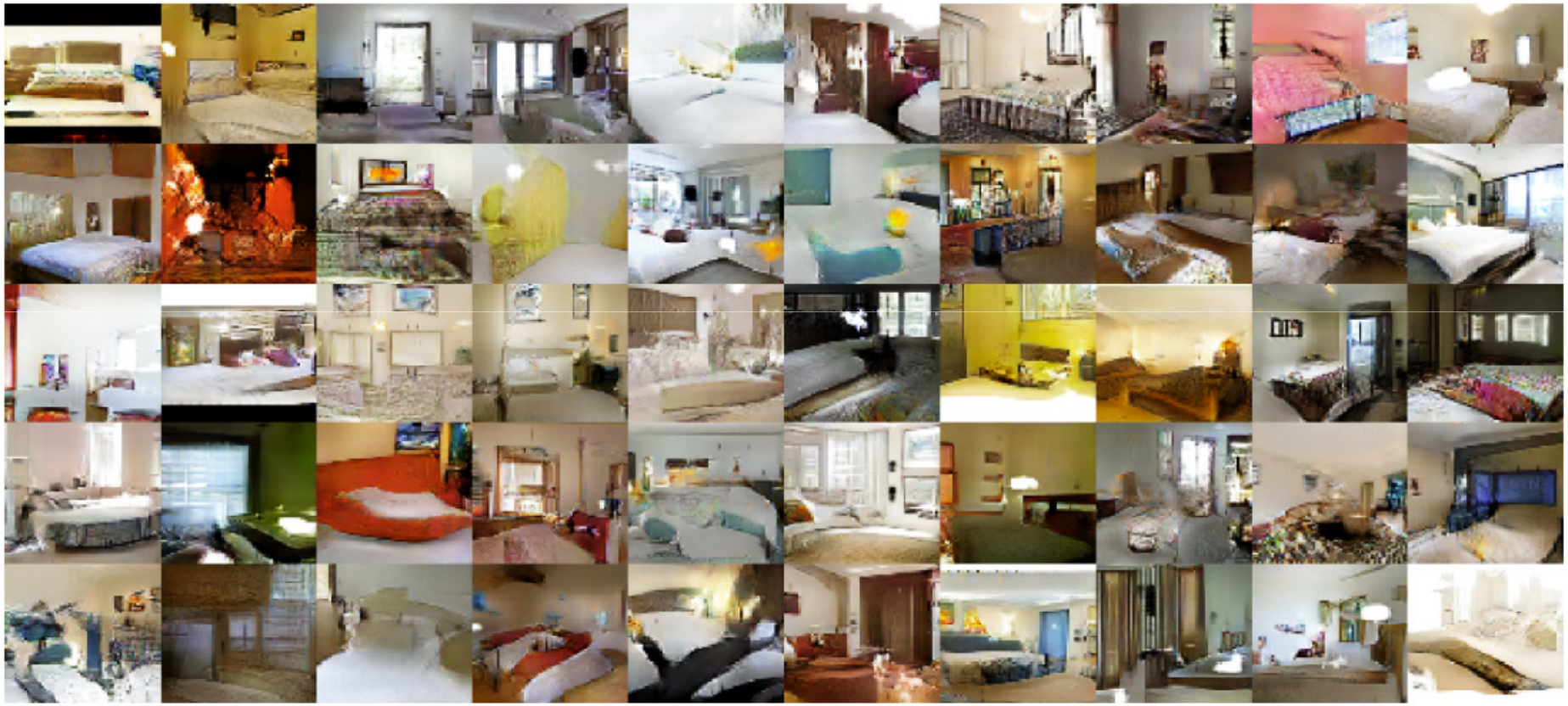
Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks."

Deep Convolution GAN (DCGAN)



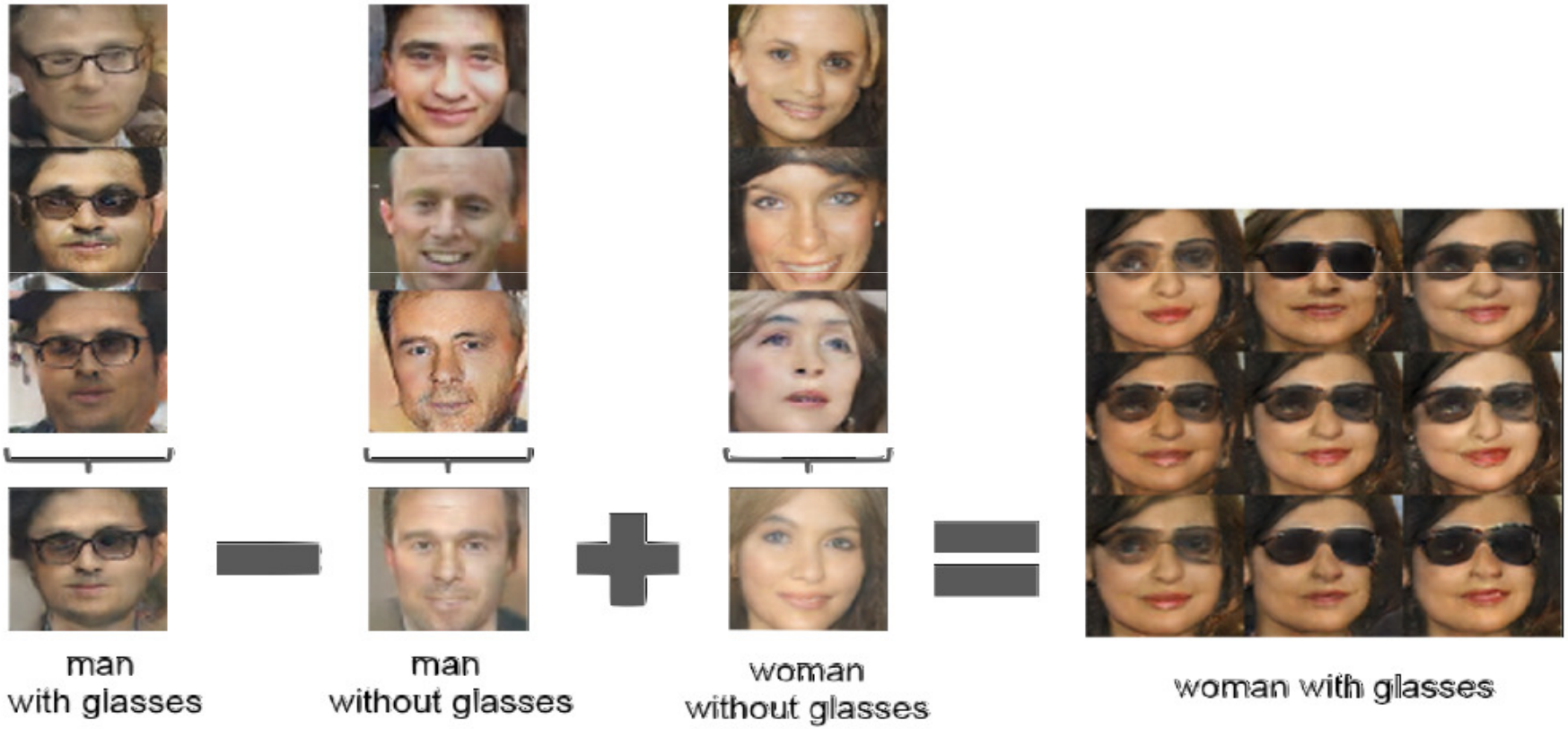
Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks."

DCGAN: Bedroom Images



Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks."

Latent vectors capture interesting patterns



Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks."

Advantages of GAN

- **Plenty of existing work on Deep Generative Models**
  - Boltzmann Machine
  - Deep Belief Nets
  - Variational AutoEncoders (VAE)
- **Why GANs?**
  - Sampling (or generation) is straightforward.
  - Training doesn't involve Maximum Likelihood estimation.
  - Robust to Overfitting since Generator never sees the training data.
  - Empirically, GANs are good at capturing the modes of the distribution.

Problems with GANs

- **Probability Distribution is Implicit**
  - Not straightforward to compute P(X).
  - Thus **Vanilla GANs** are only good for Sampling/Generation.
- **Training is Hard**
  - Non-Convergence
  - Mode-Collapse

Training: Non Convergence

- **Deep Learning models (in general) involve a single player**
  - The player tries to maximize its reward (minimize its loss).
  - Use SGD (with Backpropagation) to find the optimal parameters.
  - SGD has convergence guarantees (under certain conditions).
  - **Problem:** With non-convexity, we might converge to local optima.

$$\min_G L(G)$$

- **GANs instead involve two (or more) players**
  - Discriminator is trying to maximize its reward.
  - Generator is trying to minimize Discriminator's reward.

$$\min_G \max_D V(D, G)$$

- SGD was not designed to find the Nash equilibrium of a game.
- **Problem:** We might not converge to the Nash equilibrium at all.

Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

Training: Non Convergence

$$\min_x \max_y V(x, y)$$

Let $V(x, y) = xy$

• State 1:	x > 0	y > 0	V > 0	Increase y	Decrease x
• State 2:	x < 0	y > 0	V < 0	Decrease y	Decrease x
• State 3:	x < 0	y < 0	V > 0	Decrease y	Increase x
• State 4 :	x > 0	y < 0	V < 0	Increase y	Increase x
• State 5:	x > 0	y > 0	V > 0 == State 1	Increase y	Decrease x

to continue...