Generative Models

Håkon Hukkelås | hakon.hukkelas@ntnu.no

Lecture Overview

- Unsupervised Learning

- Generative Models
  - PixelRNN and PixelCNN
  - Variational Autoencoders (VAE)
  - Generative Adversarial Networks

Supervised Learning

Data: (x,y)

x is image, y is label

Goal:

Learn a function to map x -> y

Examples:

Classification, regression, object detection, semantic segmentation, etc.

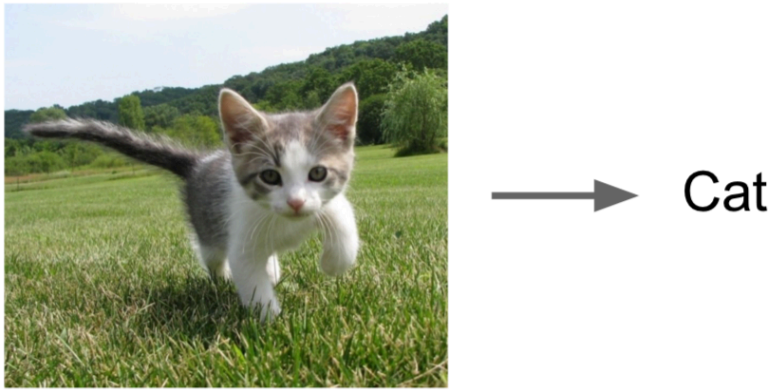
Slides credit: Stanford cs231n

Supervised Learning

Data: (x,y)
x is image, y is label

Goal:
Learn a function to map x -> y

Examples:
Classification, regression, object detection, semantic segmentation, etc.



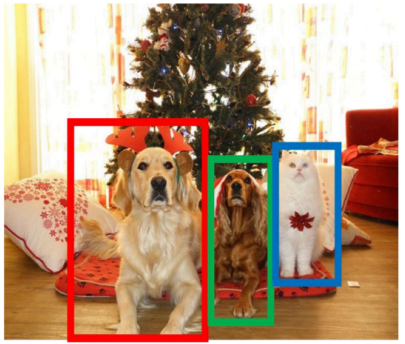
Classification

Supervised Learning

Data: (x,y)
x is image, y is label

Goal:
Learn a function to map x -> y

Examples:
Classification, regression, object detection, semantic segmentation, etc.



DOG, DOG, CAT

Object Detection

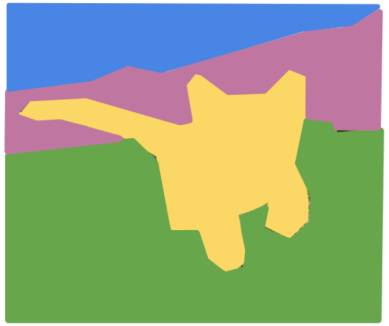
Slides credit: Stanford cs231n

Supervised Learning

Data: (x,y)
x is image, y is label

Goal:
Learn a function to map x -> y

Examples:
Classification, regression, object detection, semantic segmentation, etc.



GRASS, CAT,
TREE, SKY

Semantic Segmentation

Slides credit: Stanford cs231n

Unsupervised Learning

Data: x

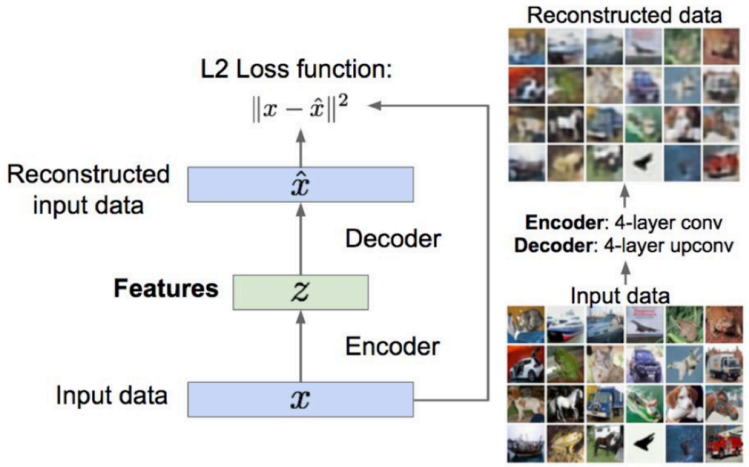
Just images, no labels!

Goal:

Learn som underlying hidden structure of data

Examples:

Clustering, dimensionality reduction, feature learning, density estimation, etc.



Autoencoders
(Feature learning)

Slides credit: Stanford cs231n

Supervised vs Unsupervised Learning

Data: (x,y)
x is image, y is label

Goal:
Learn a function to map x -> y

Examples:
Classification, regression, object detection, semantic segmentation, etc.

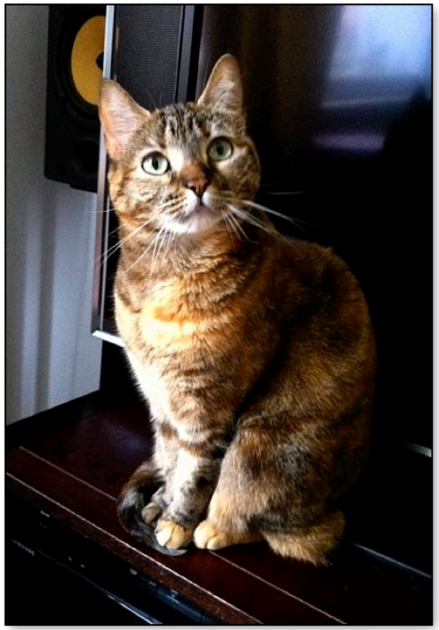
Data: x
Just images, no labels!

Goal:
Learn som underlying hidden structure of data

Examples:
Clustering, dimensionality reduction, feature learning, density estimation, etc.

Slides credit: Stanford cs231n

Classification vs Generation

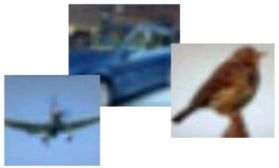


“What is in the picture?”

EASY

Generative Models

Given training data, generate new samples from same distribution



Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Slides credit: Stanford cs231n

Why Generative Models?

- Application areas:
  - Realistic samples of art work
  - Super-resolution
  - Colorization
  - Generate new datasets?
- Enable inference of latent representations that can be useful as general features

Super Resolution

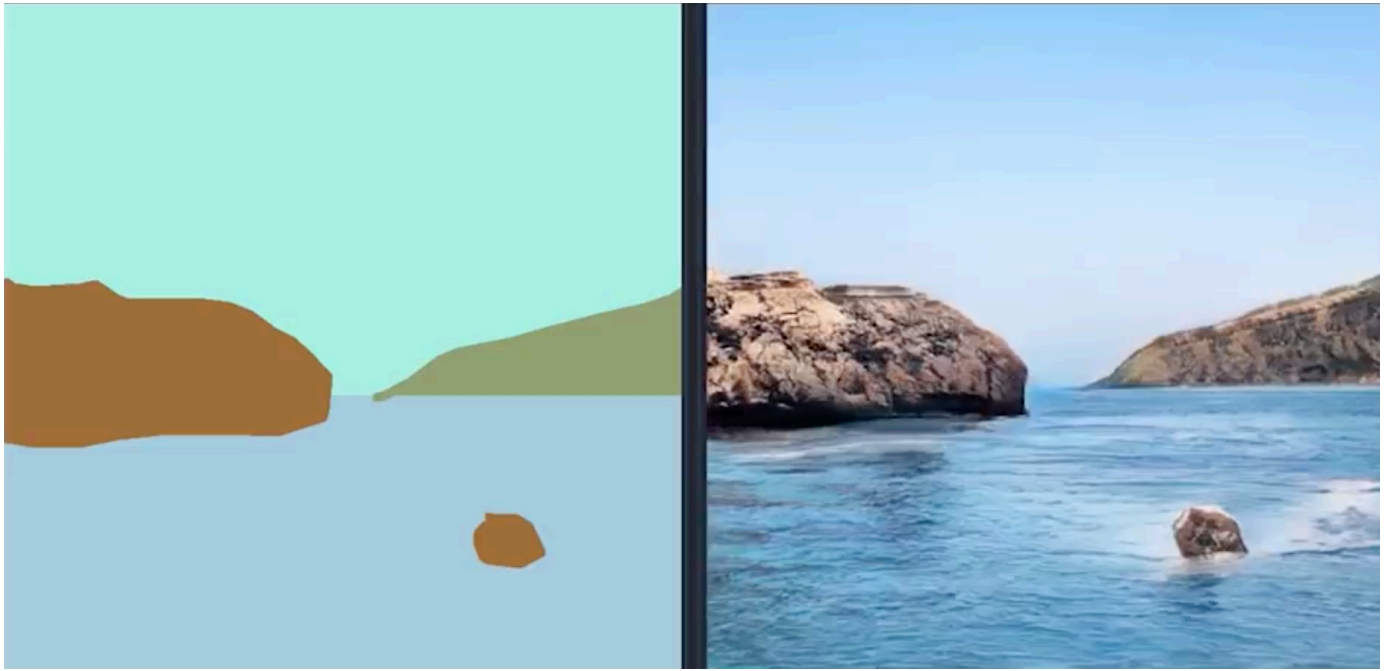
"GENERATING HIGH FIDELITY IMAGES WITH SUBSCALE PIXEL NETWORKS AND MULTIDIMENSIONAL UPSCALING", Menick et. al.

Style Transfer

"Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization", Xun Huang et. al.

What is the potential of this?

Semantic Map | Generated Image



https://nvlabs.github.io/SPADE/

Taxonomy of Generative Models

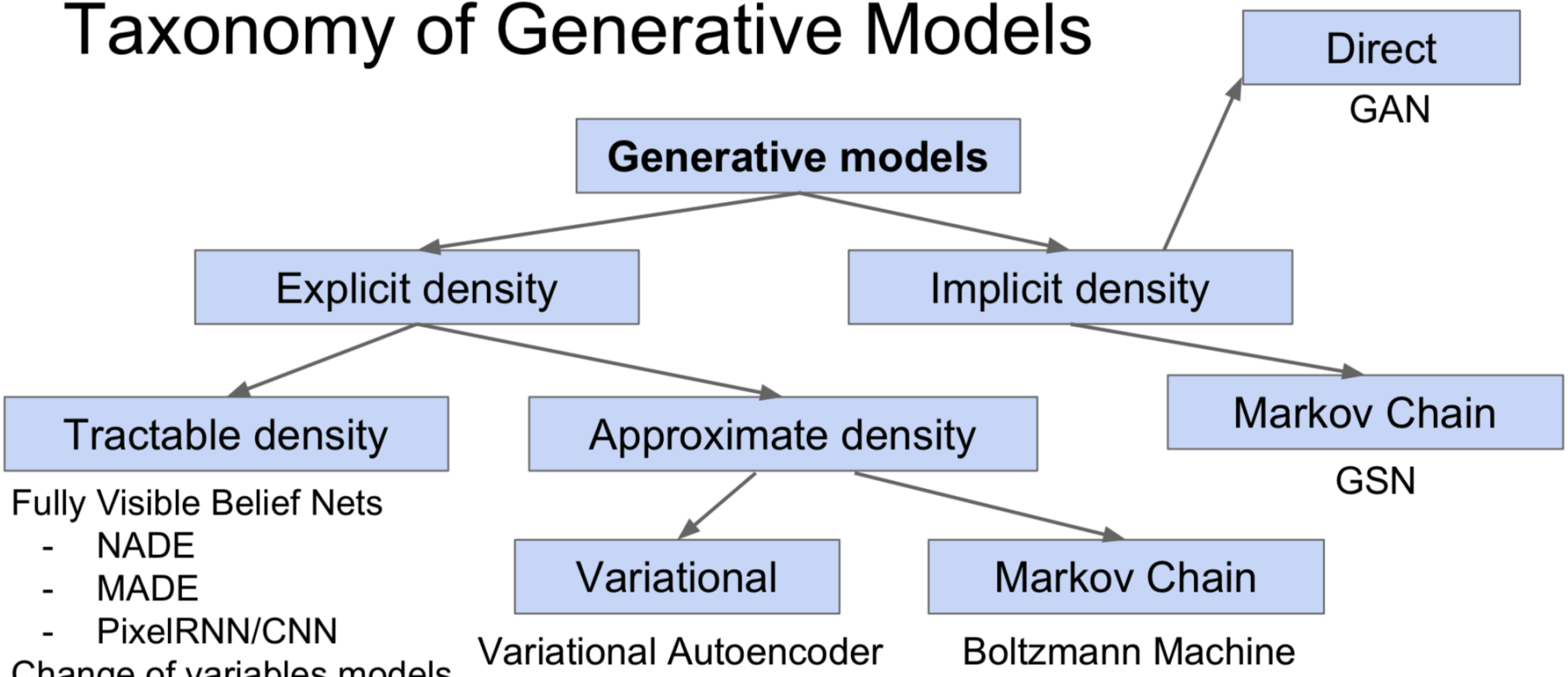


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

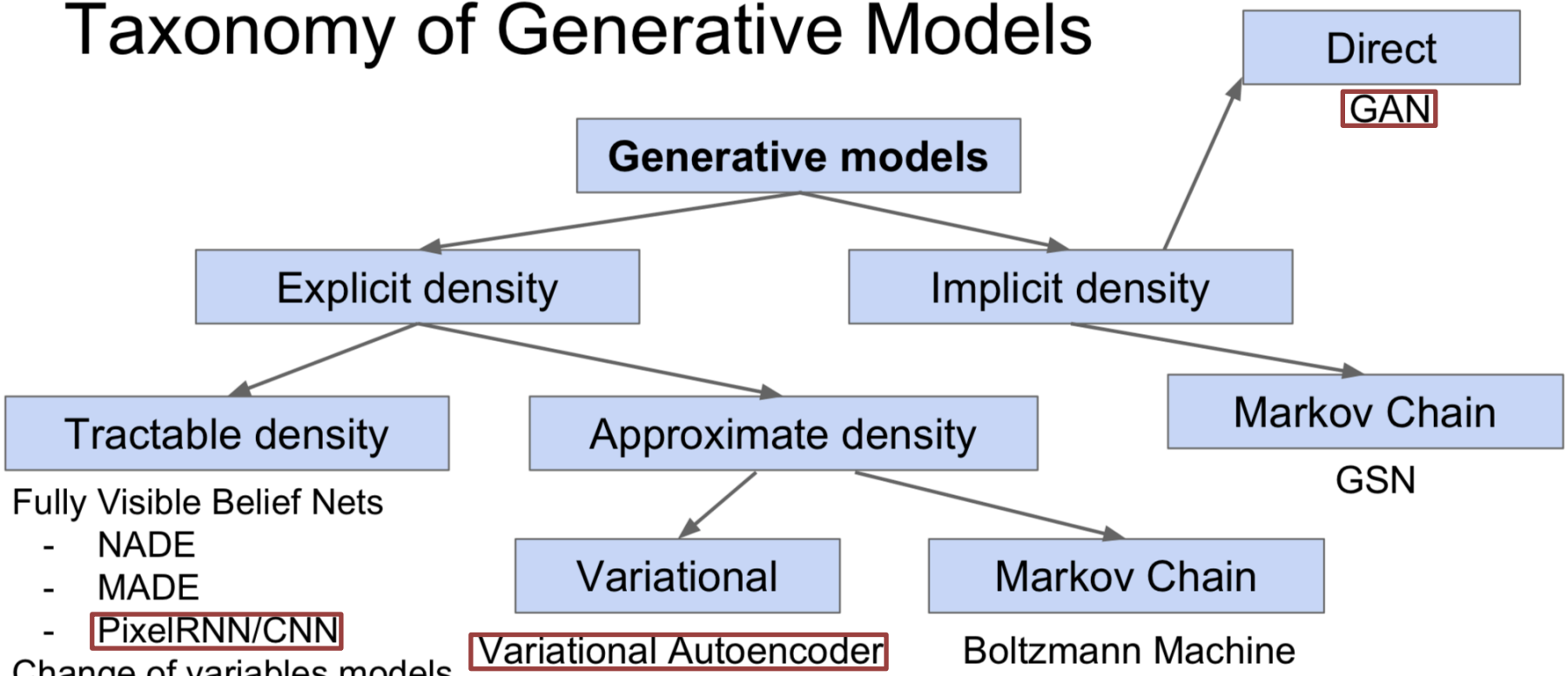


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PixelRNN and PixelCNN

Fully Visible Belief Network

Explicit density model

Use chain rule to decompose likelihood of an image x into a product of 1-d distributions

$$p(x) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels

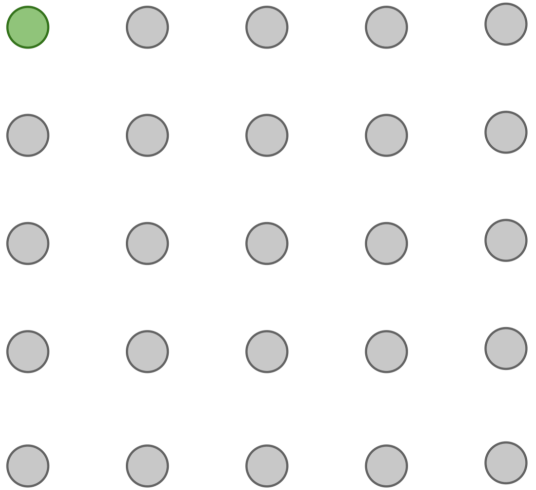
Then maximize likelihood of data

Slides credit: Stanford cs231n

PixelRNN

Generating pixels starting from corner

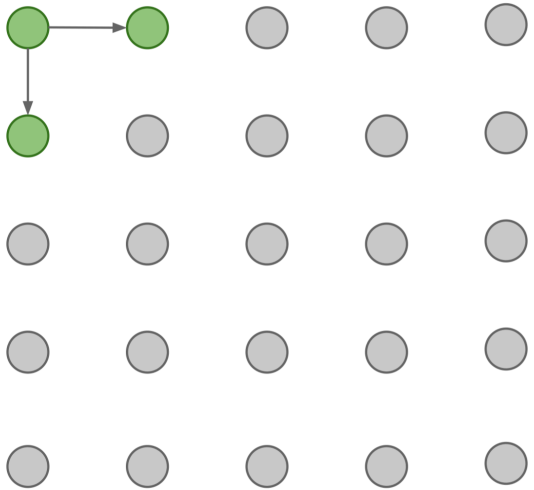
Dependency on previous pixels modelled as an RNN (LSTM)



PixelRNN

Generating pixels starting from corner

Dependency on previous pixels modelled as an RNN (LSTM)

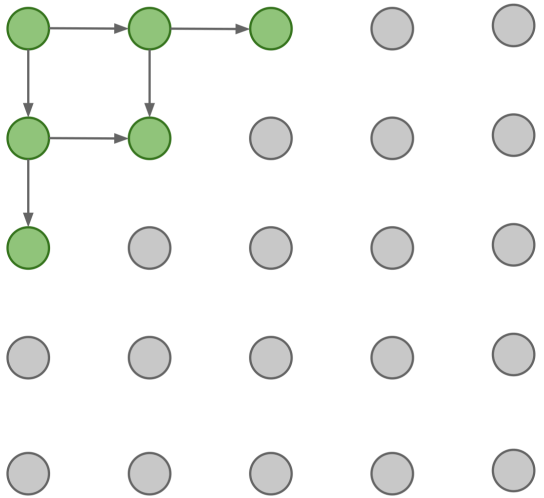


Slides credit: Stanford cs231n

PixelRNN

Generating pixels starting from corner

Dependency on previous pixels modelled as an RNN (LSTM)

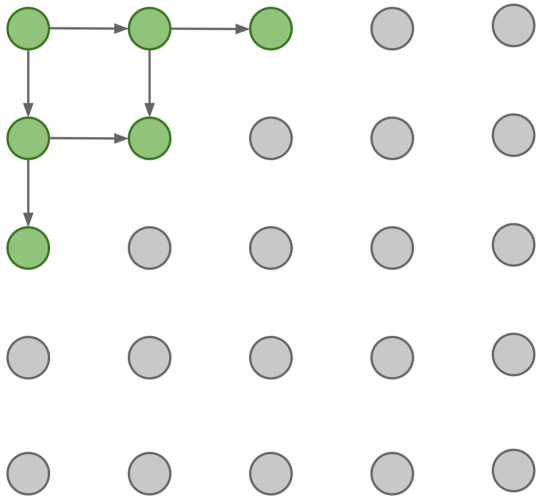


PixelRNN

Generating pixels starting from corner

Dependency on previous pixels modelled as an RNN (LSTM)

Drawback: sequential generation is slow

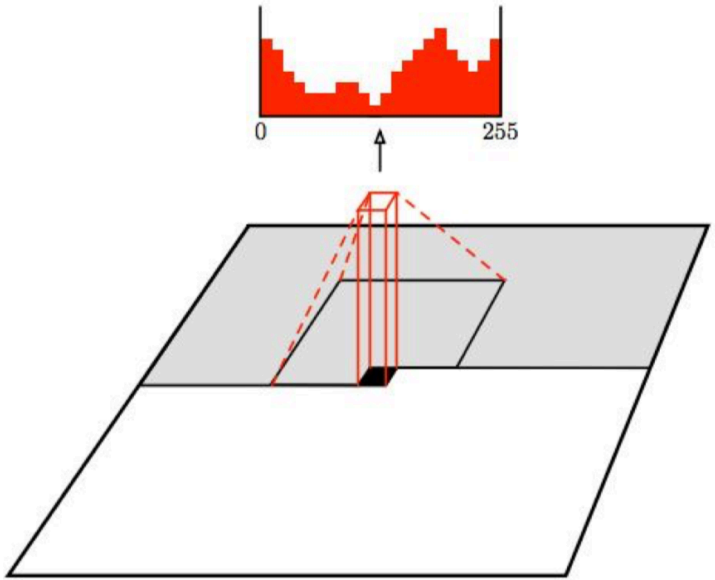


Slides credit: Stanford cs231n

PixelCNN

Generating pixels starting from corner

Dependency on previous pixels is now modelled using a CNN over context region



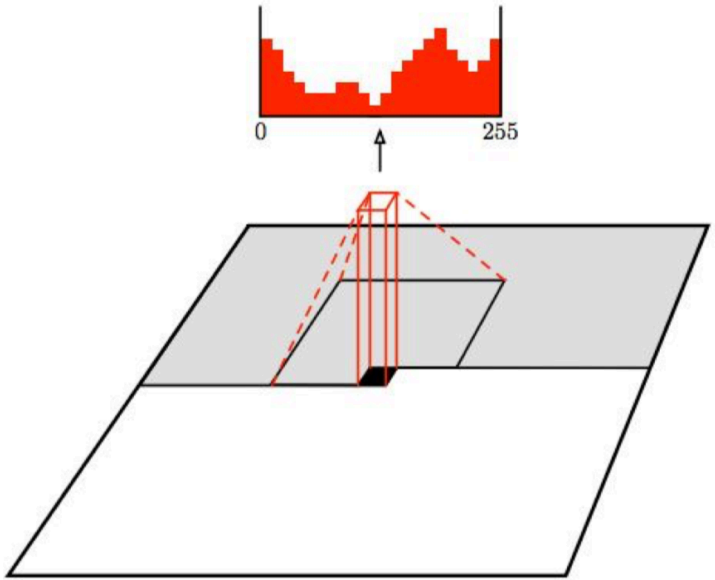
Slides credit: Stanford cs231n

PixelCNN

Generating pixels starting from corner

Dependency on previous pixels is now modelled using a CNN over context region

Softmax loss at each pixel



Slides credit: Stanford cs231n

PixelCNN

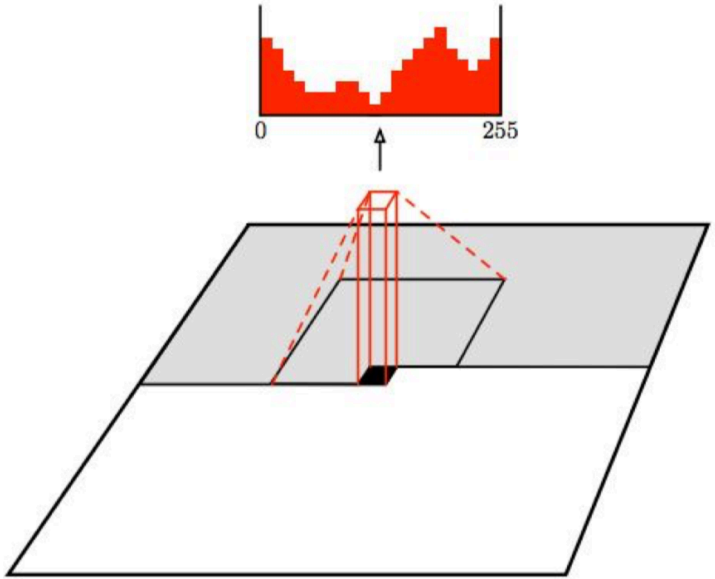
Training is faster than PixelRNN

Can parallelize convolutions since context region values are known from training images

Test time generation is still slower

Generation much proceed sequentially

Softmax loss at each pixel



Slides credit: Stanford cs231n

PixelCNN++ Results

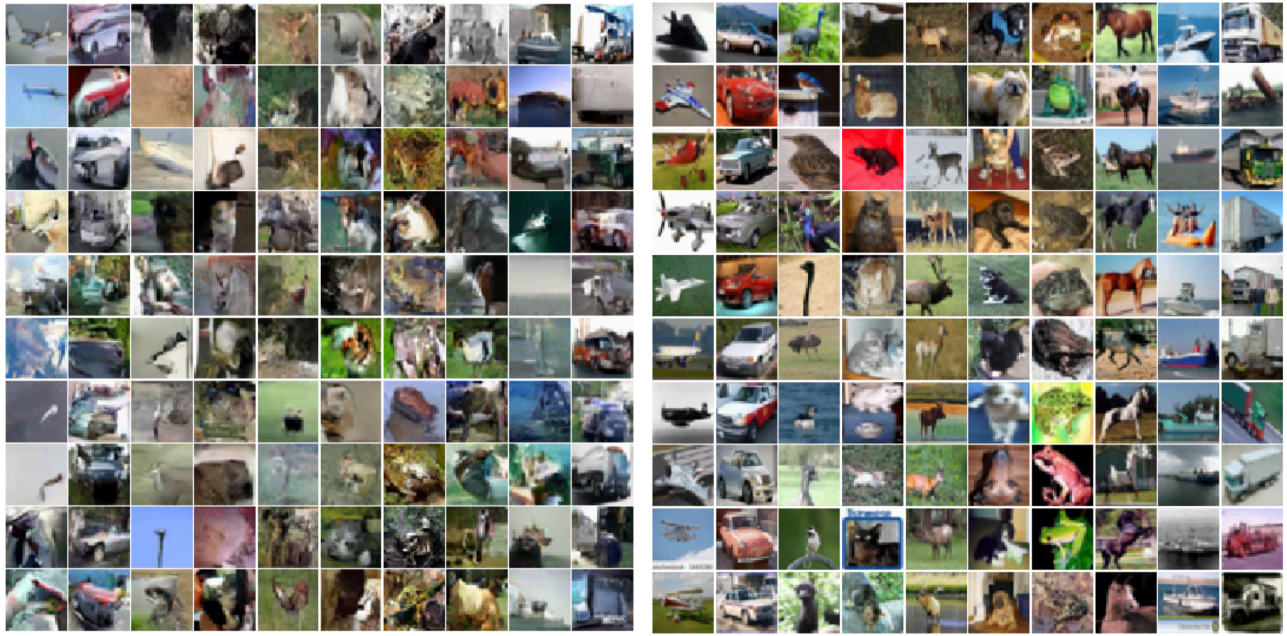


Figure 4: Class-conditional samples from our PixelCNN for CIFAR-10 (left) and real CIFAR-10 images for comparison (right).

"PIXELCNN++: IMPROVING THE PIXELCNN WITH DISCRETIZED LOGISTIC MIXTURE LIKELIHOOD AND OTHER MODIFICATIONS", Salimans et. al

PixelRNN and PixelCNN Summary

Pros:

- Can explicitly compute likelihood p(x)
- Explicit likelihood of training data gives good evaluation metric

Cons:

- Sequential generation => Slow

Much more work on this improving PixelCNN..:

- PixelCNN++
- Gated PixelCNN
- Conditional PixelCNN

Slides credit: Stanford cs231n

Variational Autoencoders(VAE)

So far…

PixelCNN define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i \,|\, x_1, \ldots, x_{i-1})$$

Slides credit: Stanford cs231n

So far…

PixelCNN define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i \,|\, x_1, \ldots, x_{i-1})$$

VAEs define a intractable density function with **latent variable z:**

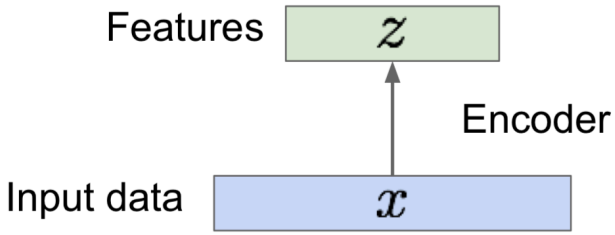
$$p_\theta(x) = \int p_\theta(z) p_\theta(x \,|\, z) dz$$

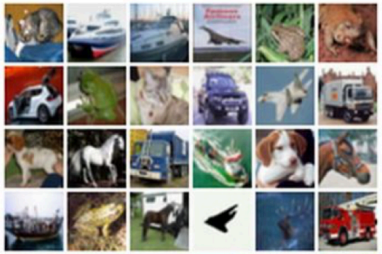
Cannot optimize directly, derive and optimize lower bound on likelihood instead.

Slides credit: Stanford cs231n

Background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



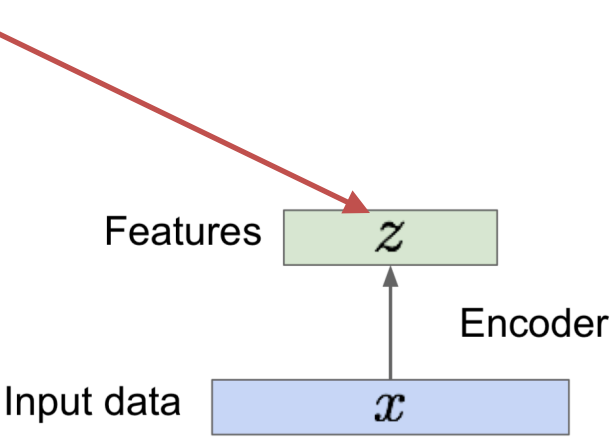


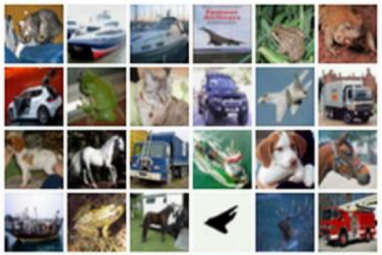
Slides credit: Stanford cs231n

Background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x** (dimensionality reduction)





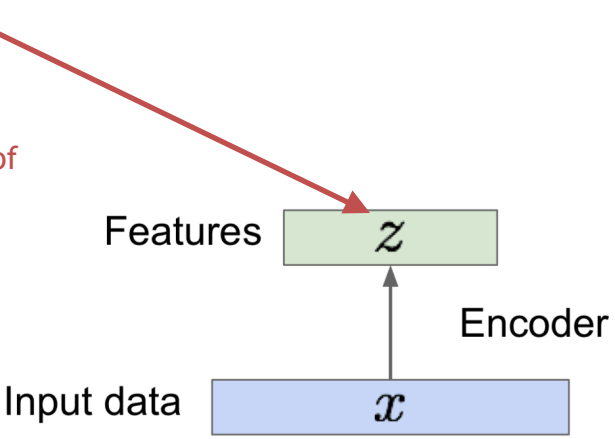
Slides credit: Stanford cs231n

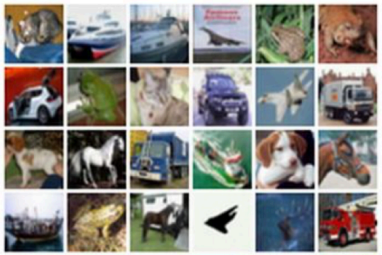
Background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x** (dimensionality reduction)

Want **z** to capture meaningful factors of variation in data => smaller than **x**



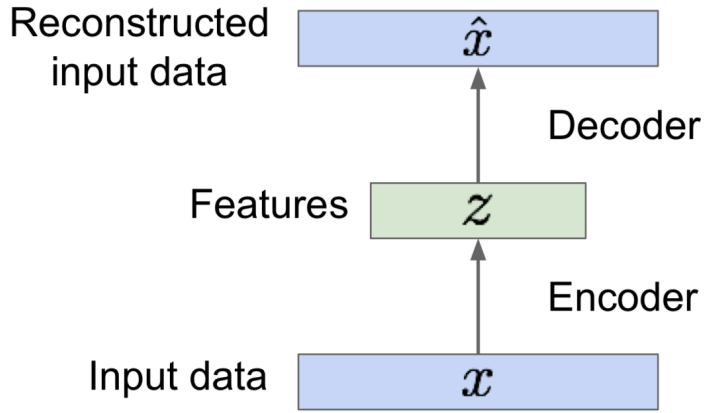


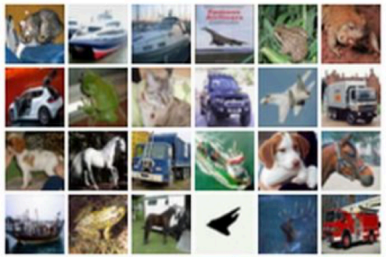
Slides credit: Stanford cs231n

Background first: Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data



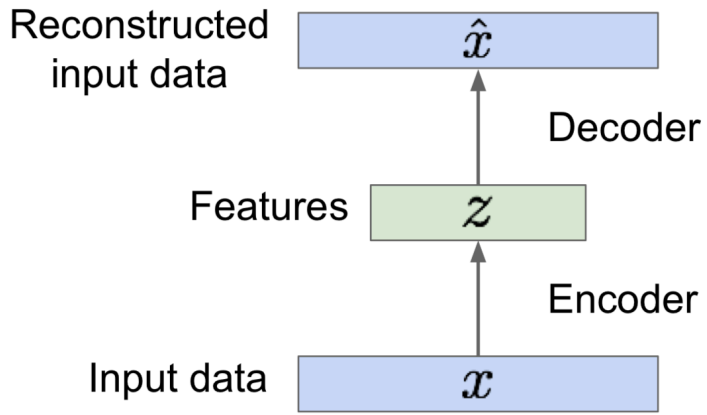


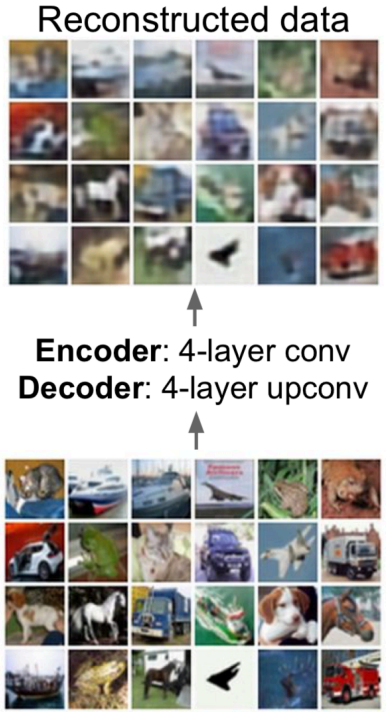
Slides credit: Stanford cs231n

Background first: Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data





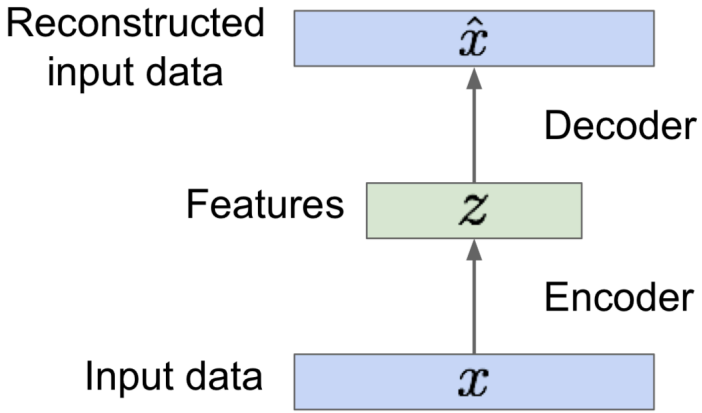
Slides credit: Stanford cs231n

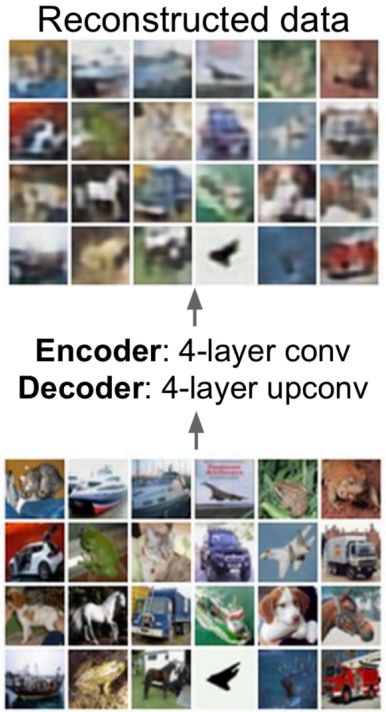
Background first: Autoencoders

Doesn't use labels!

L2 Loss function:

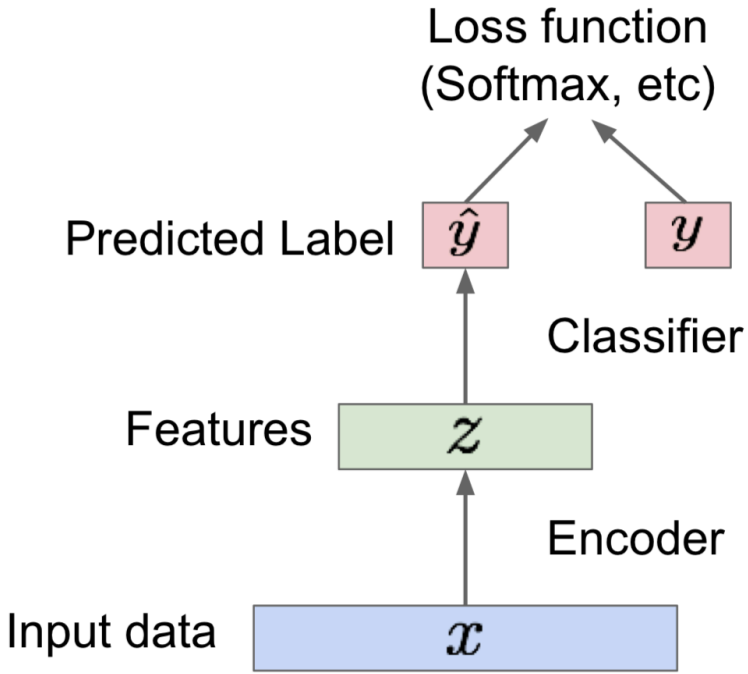
$$||x - \hat{x}||^2$$





Slides credit: Stanford cs231n

Background first: Autoencoders

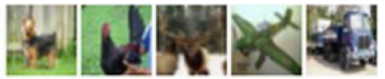


Encoder can be used to initialize a **supervised** model

Fine-tune encoder jointly with classifier

Train for final task (sometimes with small data)

bird plane
dog deer truck

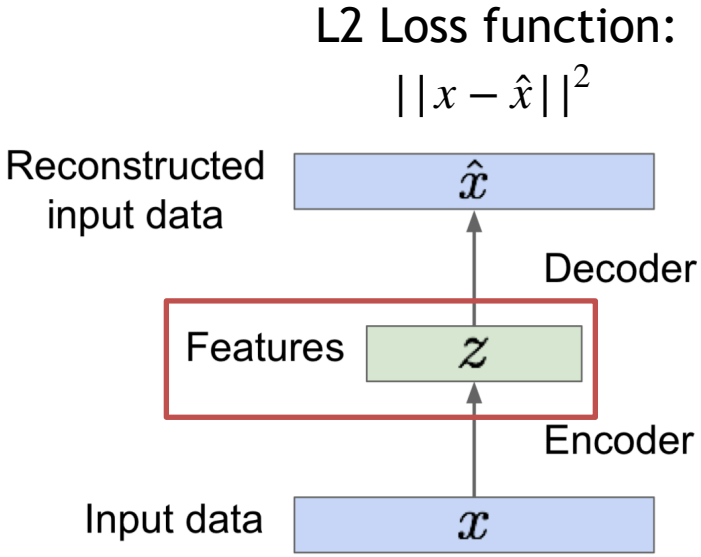


Slides credit: Stanford cs231n

Variational Autoencoder

Not possible to sample form an autoencoder

Force Z to be gaussian!

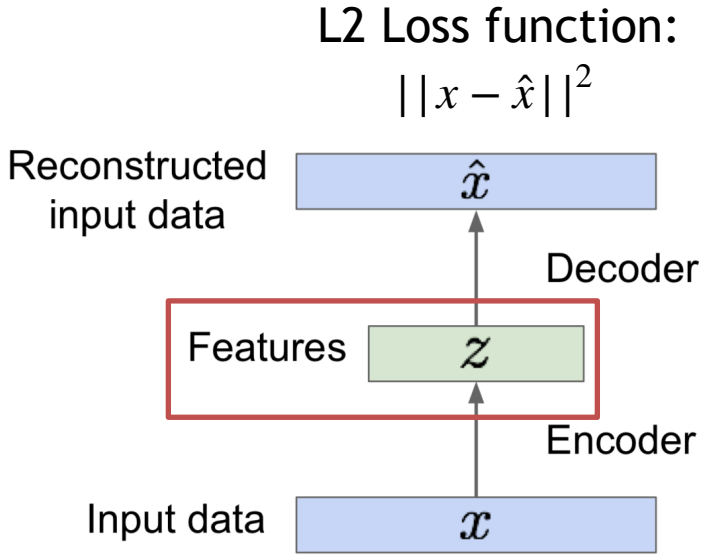


Variational Autoencoder

Not possible to sample form an autoencoder

Force Z to be gaussian!

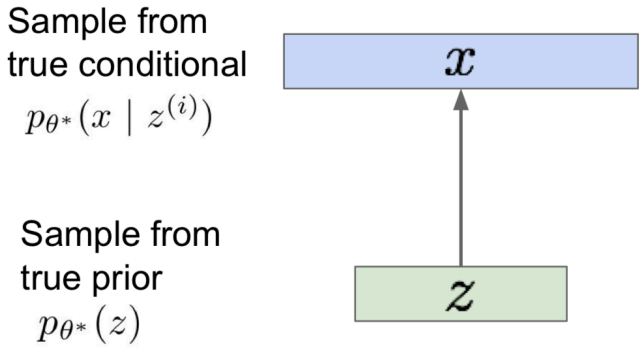
=> Possible to sample from



Variational Autoencoders(VAE)

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data is generated from underlying unobserved latent representation **z**



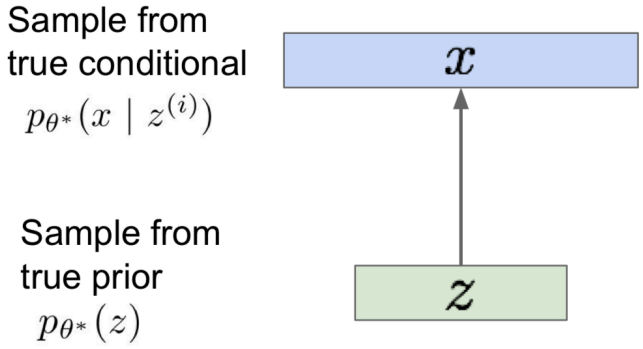
Slides credit: Stanford cs231n

Variational Autoencoders(VAE)

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data is generated from underlying unobserved latent representation **z**

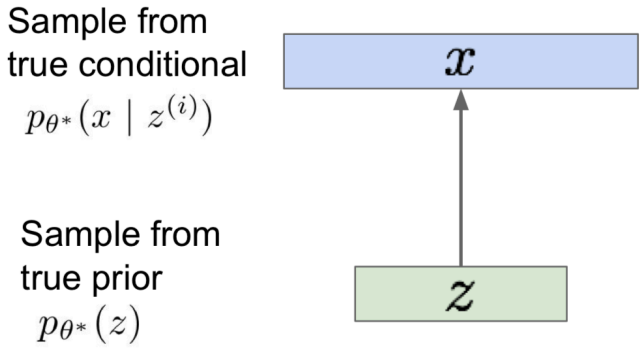
Intuition: **x** is an image, **z** is latent factors used to generate x: attributes, orientation, etc.



Slides credit: Stanford cs231n

Variational Autoencoders(VAE)

We want to estimate the true parameters, theta, of this generative model



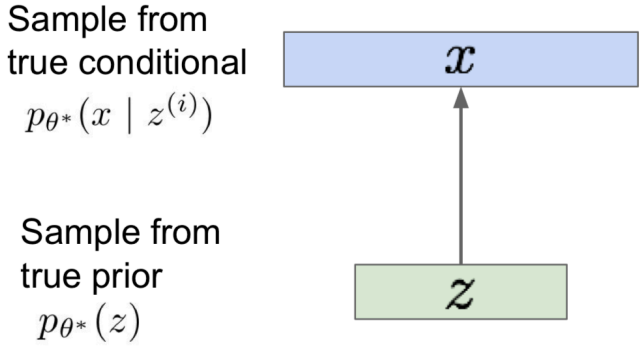
Slides credit: Stanford cs231n

Variational Autoencoders(VAE)

We want to estimate the true parameters, theta, of this generative model

How should we represent it?

Choose prior p(z) to be simple. e.g: Gaussian.



Slides credit: Stanford cs231n

Variational Autoencoders(VAE)

We want to estimate the true parameters, theta, of this generative model

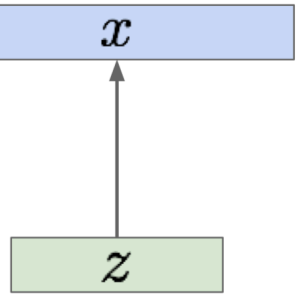
How should we represent it?

Choose prior p(z) to be simple. e.g: Gaussian.

Conditional p(x|z) is complex (generates image) => represent with neural network.

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior $p_{\theta^*}(z)$

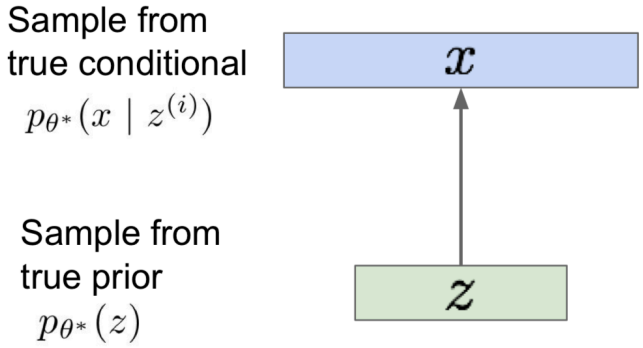


Slides credit: Stanford cs231n

Training VAEs

Maximize likelihood of training data:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x \mid z) \partial z$$

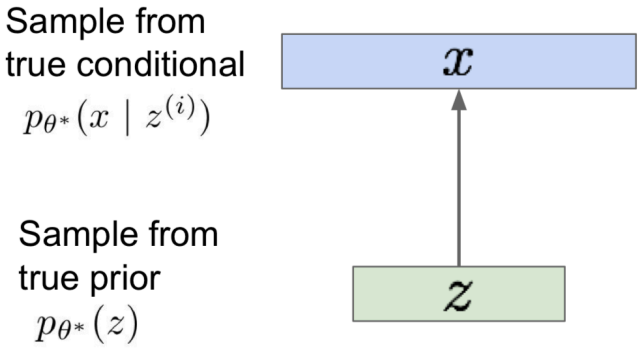


Training VAEs

Maximize likelihood of training data:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x \mid z) \partial z$$

Problem: Intractable! Can't compute over every possible **z.**



Training VAEs

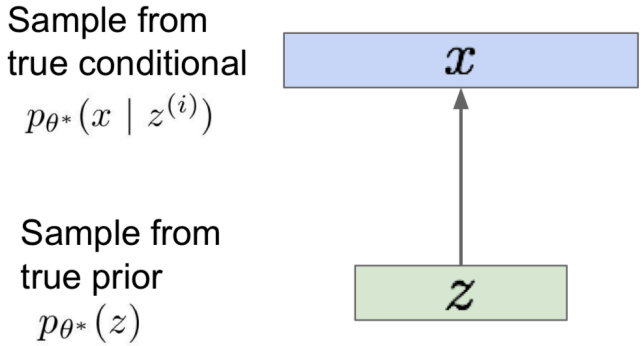
Maximize likelihood of training data:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x \mid z) \partial z$$

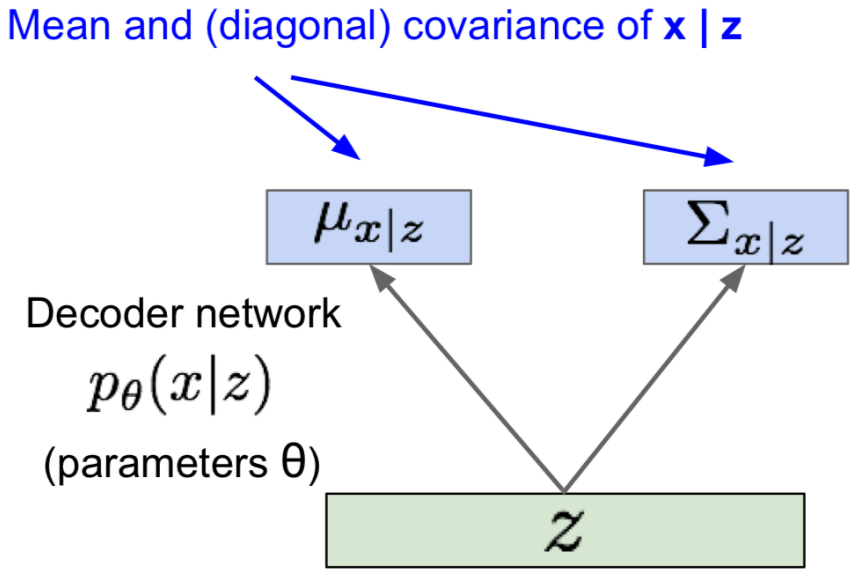
Problem: Intractable! Can't compute over every possible **z.**

Solution:
Define a decoder q(z|x) that approximates p(x|z)

This allows us to:
Derive a lower bound on the data likelihood that is tractable



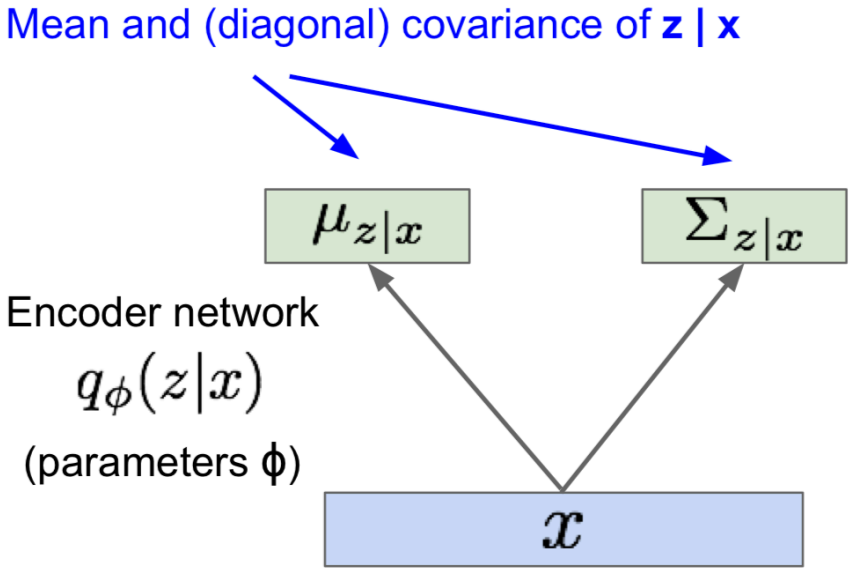
Variational Autoencoders(VAE)

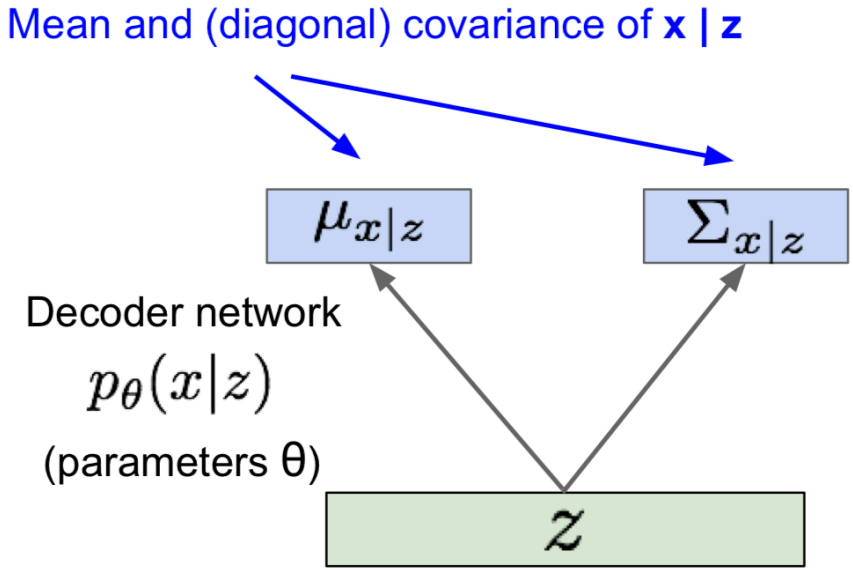


Slides credit: Stanford cs231n

Variational Autoencoders(VAE)

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic





Slides credit: Stanford cs231n

Variational Autoencoders(VAE). The math….

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})q_\phi(z \mid x^{(i)})}\right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Logarithms})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

See Stanford video for a good explanation!

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

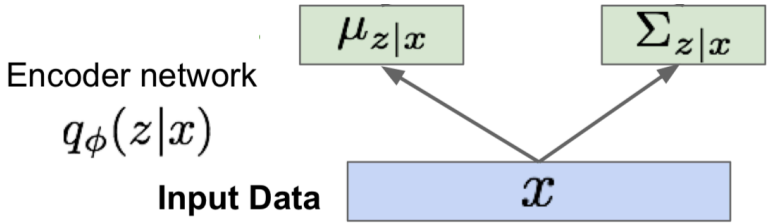
Input Data x

Slides credit: Stanford cs231n

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$



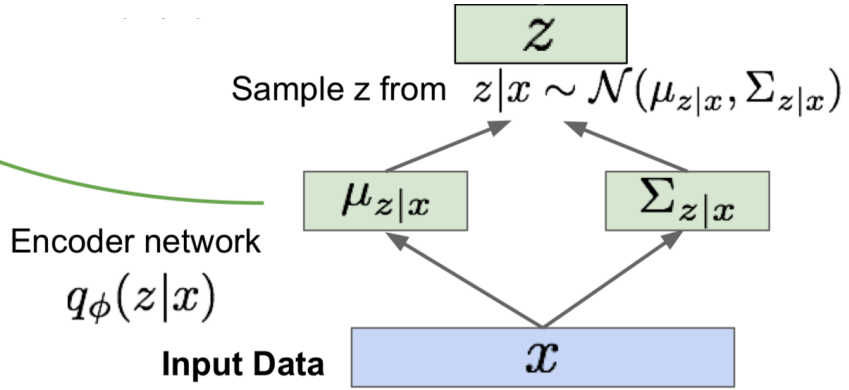
Slides credit: Stanford cs231n

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Make approximate posterior distribution close to prior

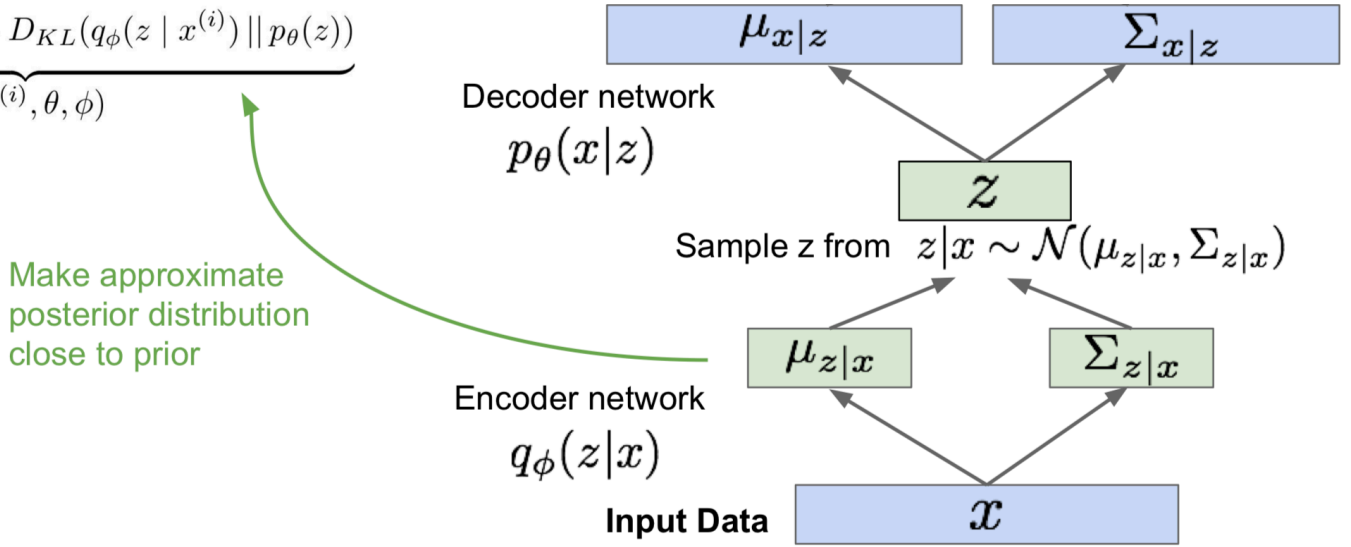


Slides credit: Stanford cs231n

Variational Autoencoders

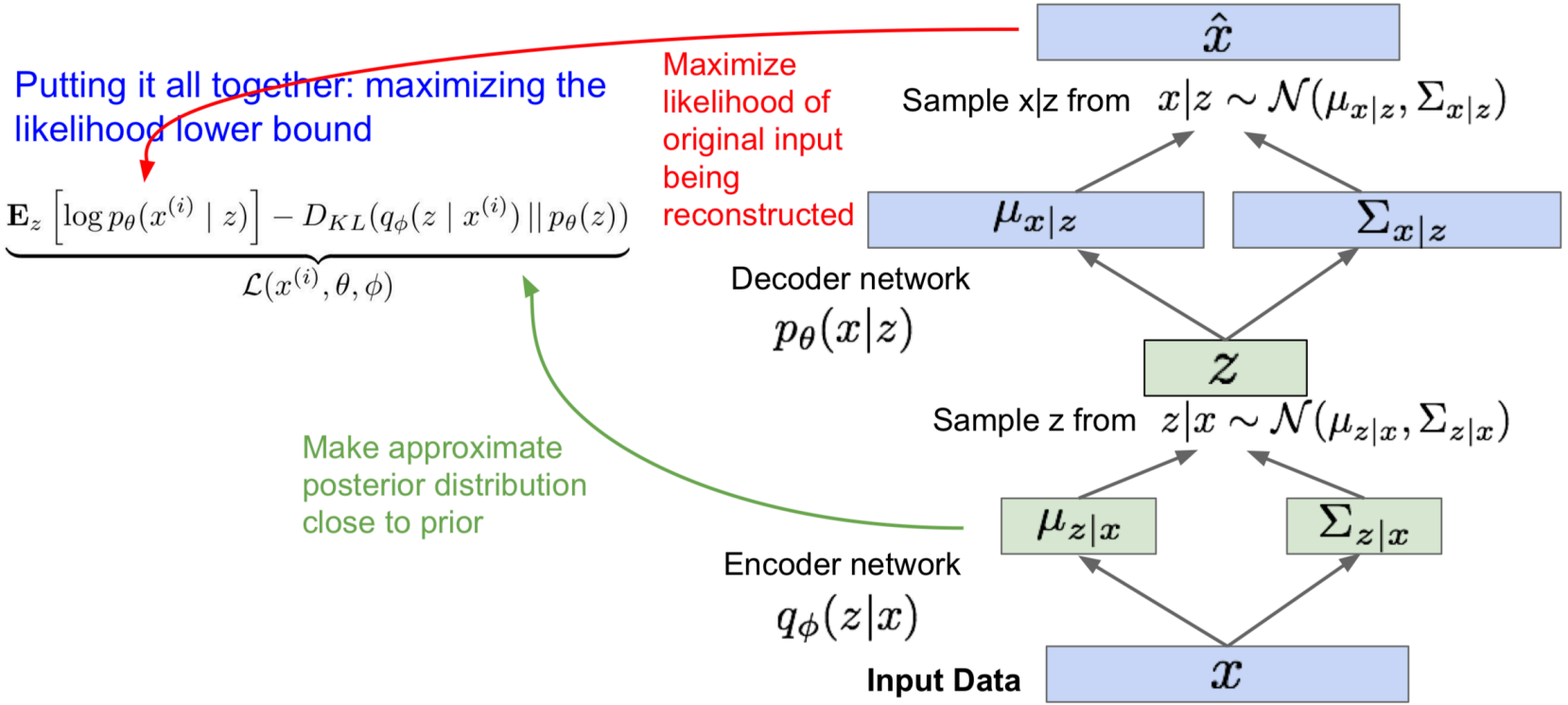
Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$



Slides credit: Stanford cs231n

Variational Autoencoders



Slides credit: Stanford cs231n

Variational Autoencoders

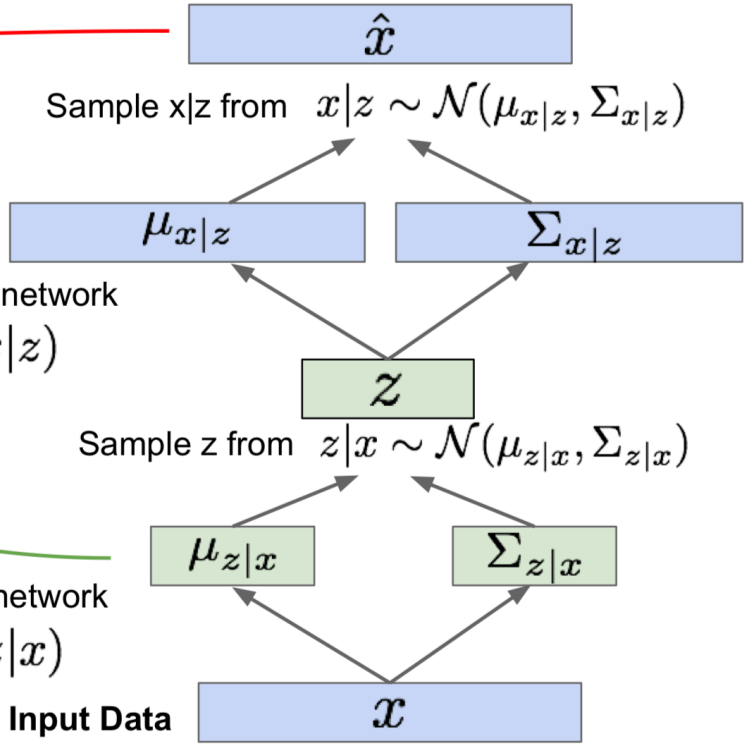
Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed

Make approximate posterior distribution close to prior

For every minibatch of input data: compute this forward pass, and then backprop!



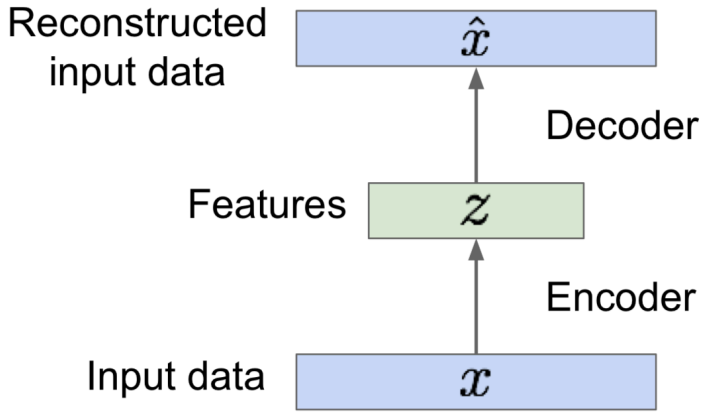
Slides credit: Stanford cs231n

VAE “Summary”

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

L2 Loss function:

$$||x - \hat{x}||^2$$



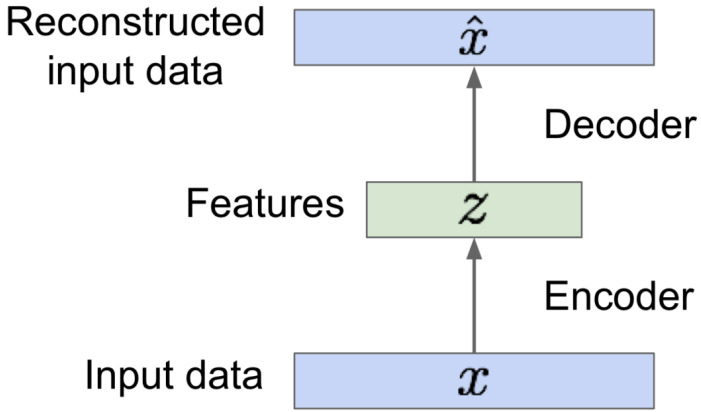
VAE “Summary”

L2 Loss

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

L2 Loss function:

$$||x - \hat{x}||^2$$



VAE “Summary”

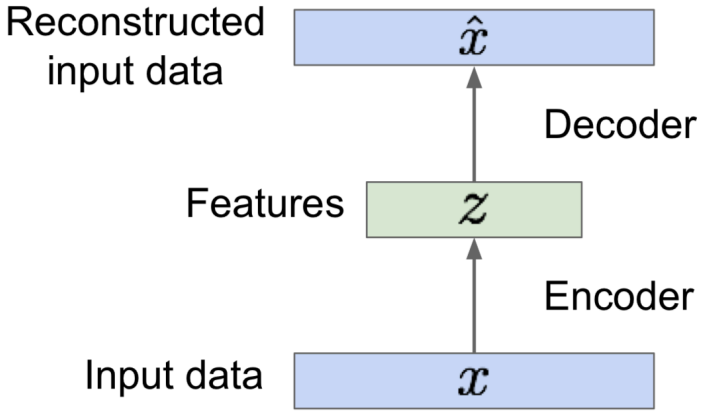
L2 Loss

“Force z to be gaussian Loss”

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

L2 Loss function:

$$||x - \hat{x}||^2$$



VAE “Summary”

L2 Loss

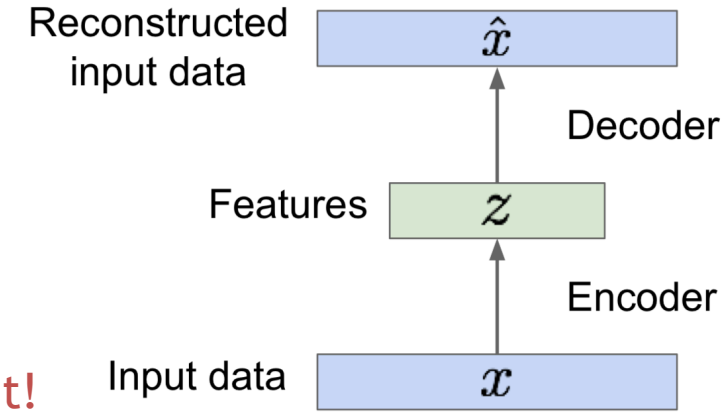
“Force z to be gaussian Loss”

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

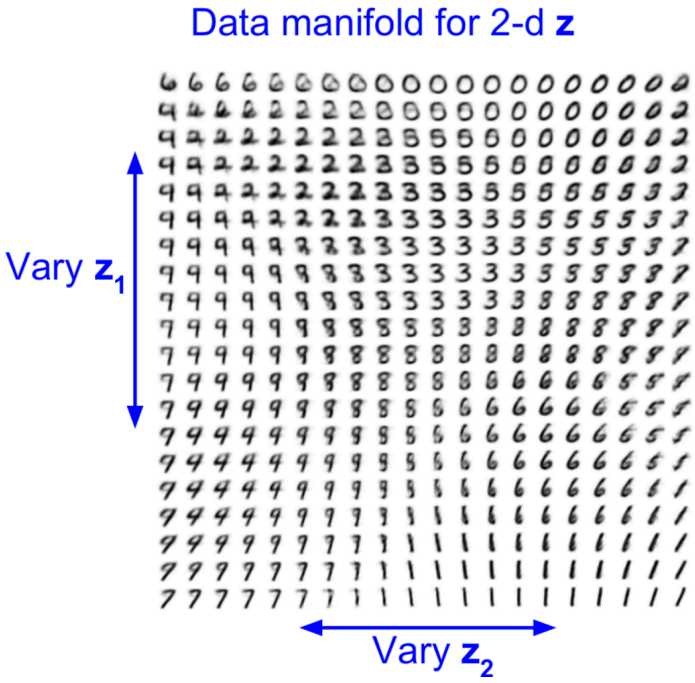
If z is gaussian, we are able to sample from it!

L2 Loss function:

$$||x - \hat{x}||^2$$



VAE Sampling Results

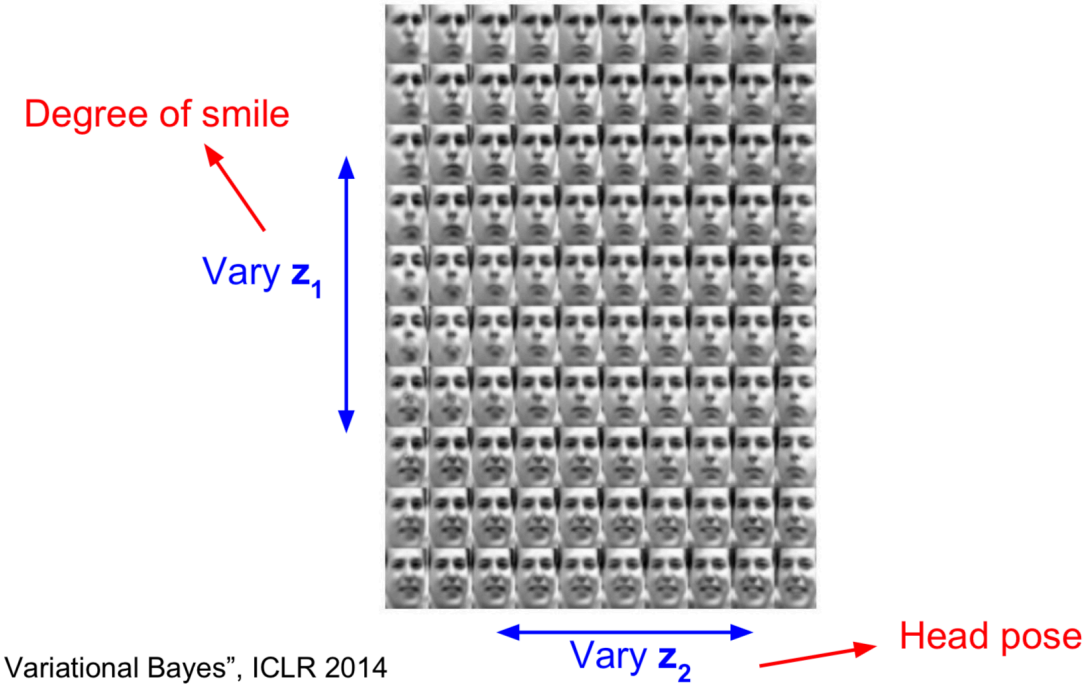


Slides credit: Stanford cs231n

VAE Sampling Results

Diagonal prior on **z**
=> independent
latent variables

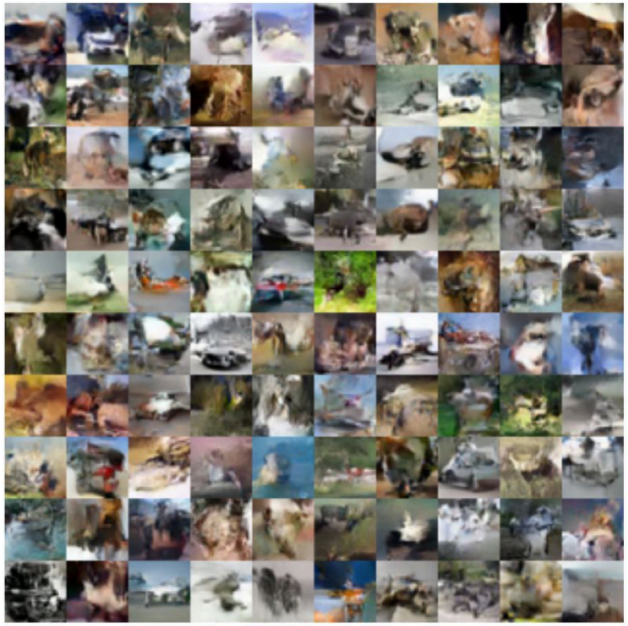
Different
dimensions of **z**
encode
interpretable factors
of variation



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Slides credit: Stanford cs231n

VAE Sampling Results



32x32 CIFAR-10

Labeled Faces in the Wild

Slides credit: Stanford cs231n

VAE Summary

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a variational lower bound

Pros:

- Principled approach to generative models
- Allows inference of q(z|x), can be useful for feature representations for other tasks

Cons:

- Maximizes lower bound of likelihood; not useful as an evaluation metric
- Samples are often blurry

Generative Adversarial Networks (GANS)

Taxonomy of Generative Models

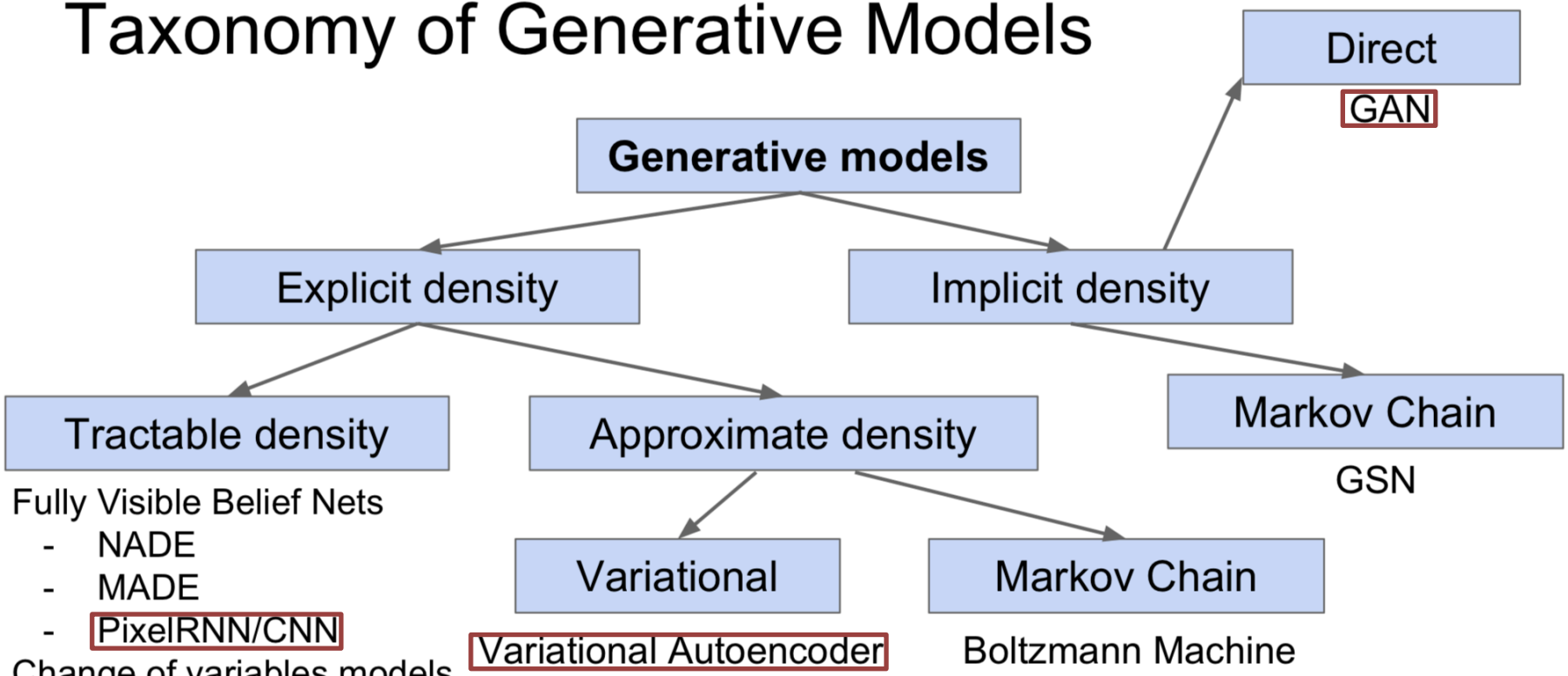


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Generative Adversarial Networks

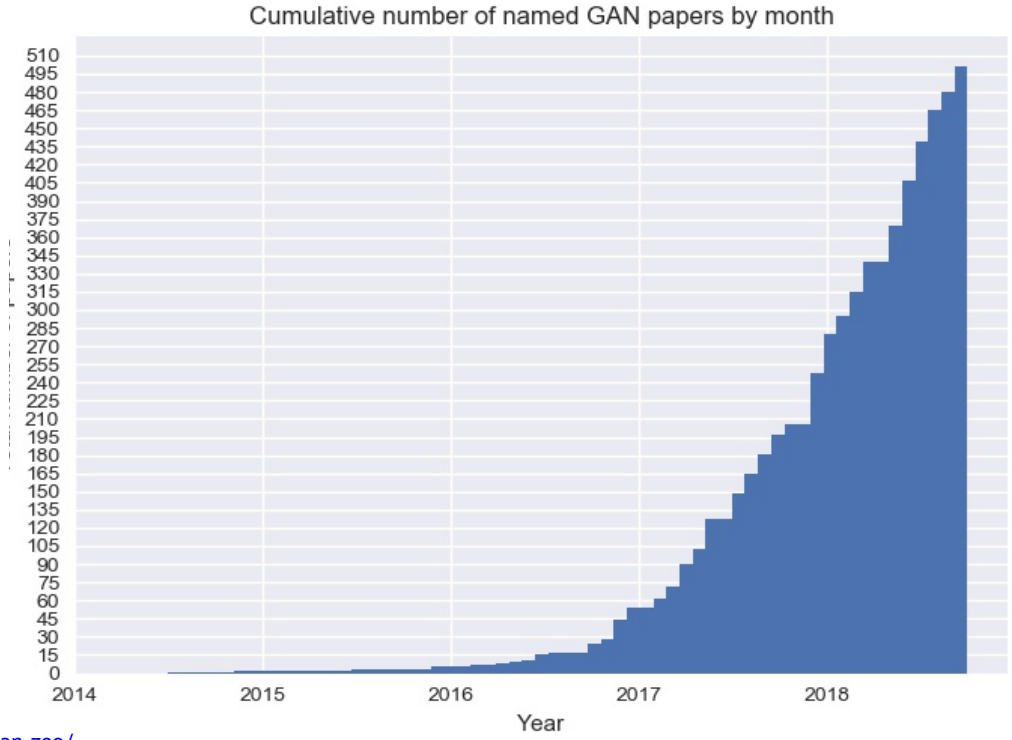
Generative Adversarial Nets

Ian J. Goodfellow[*], **Jean Pouget-Abadie**[†], **Mehdi Mirza**, **Bing Xu**, **David Warde-Farley**, **Sherjil Ozair**[‡], **Aaron Courville**, **Yoshua Bengio**[§]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

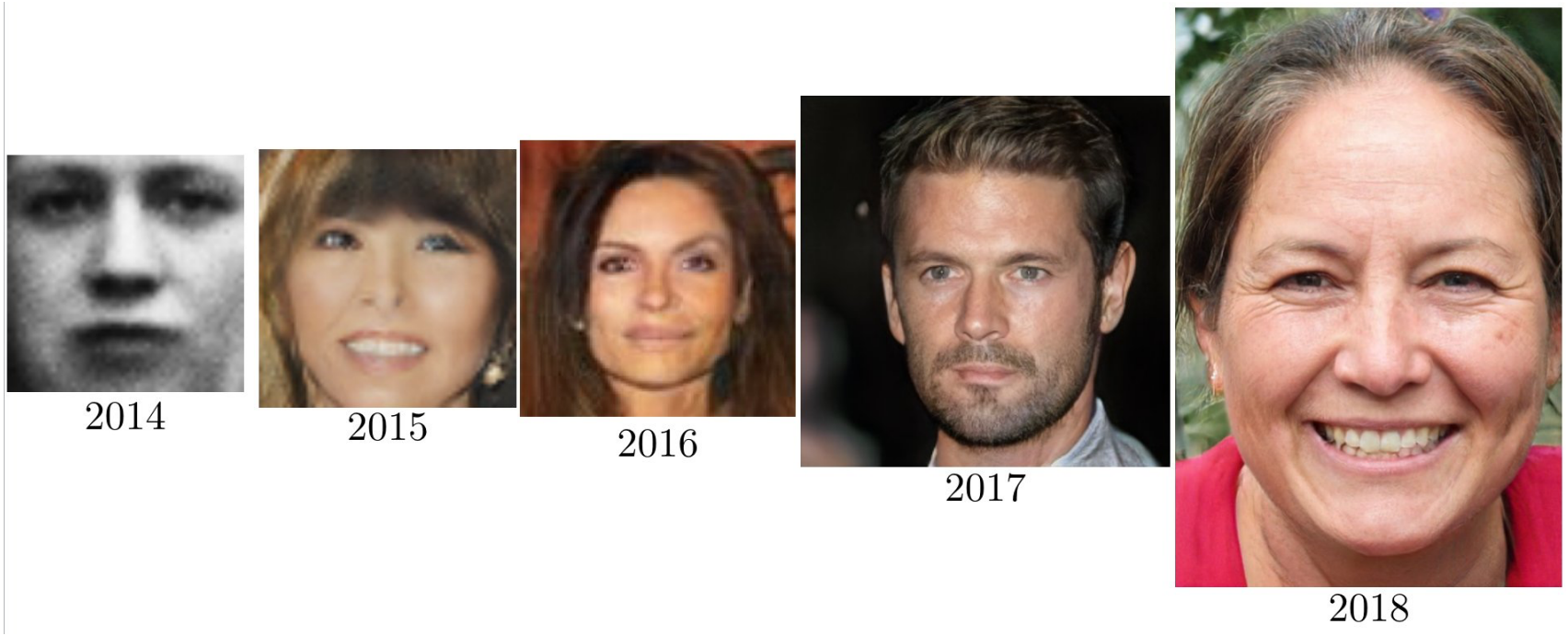
We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

Generative Adversarial Networks



https://github.com/hindupuravinash/the-gan-zoo/

“4.5 years of GAN progress”



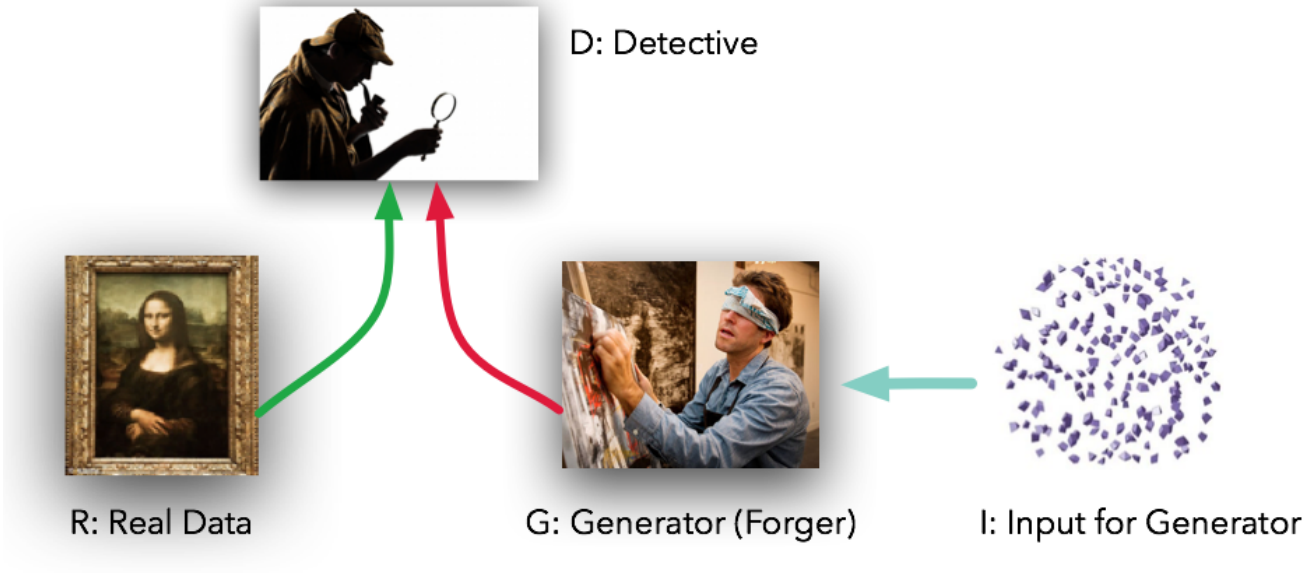
https://twitter.com/goodfellow_ian/status/1084973596236144640

Generative Adversarial Networks

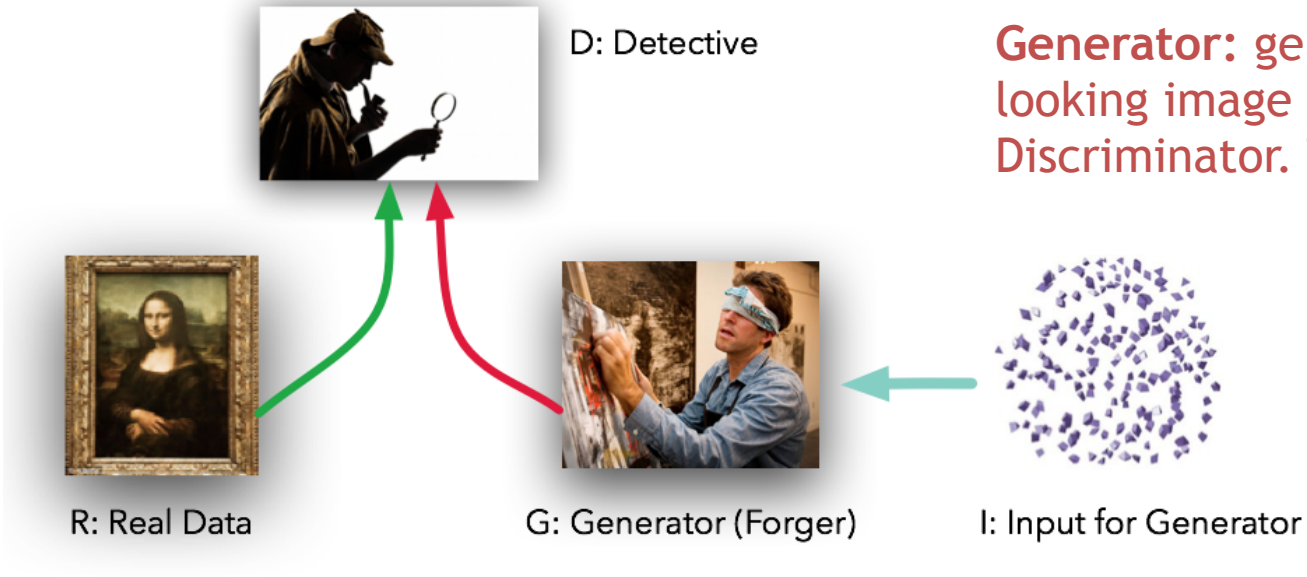
Instead of defining the probability distribution explicitly..

Lets take inspiration from game theory and optimize it as a two-player game

Generative Adversarial Networks



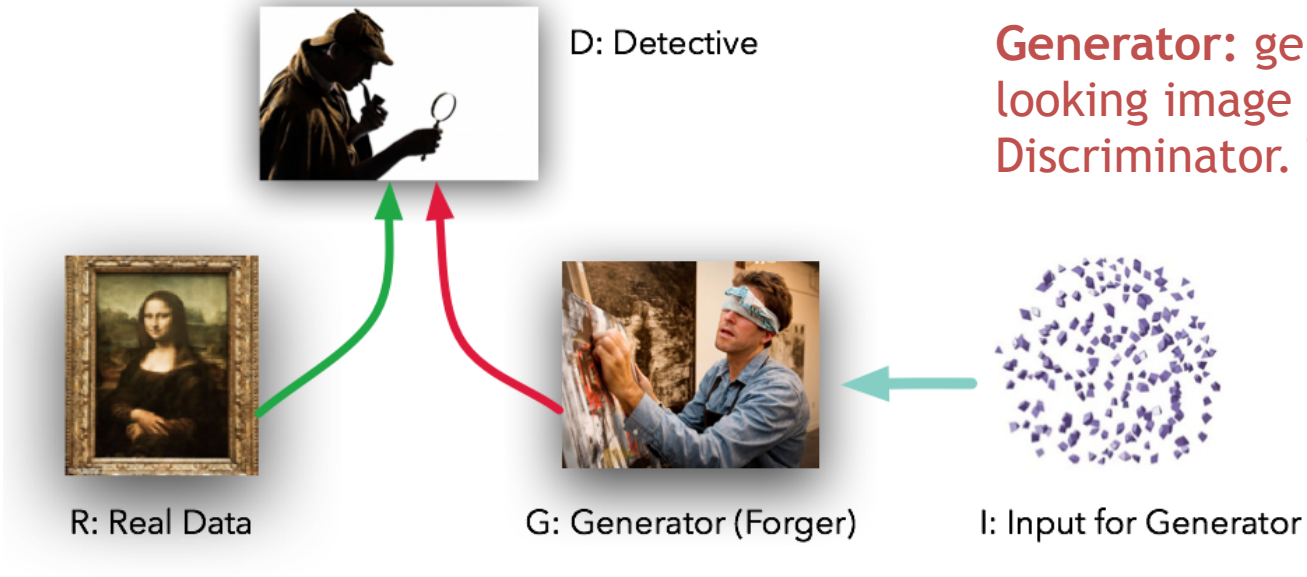
Generative Adversarial Networks



Generator: generate a realistic looking image and fool the Discriminator. The "Art forger"

Generative Adversarial Networks

Discriminator: try to detect counterfeit images. The "Detective"

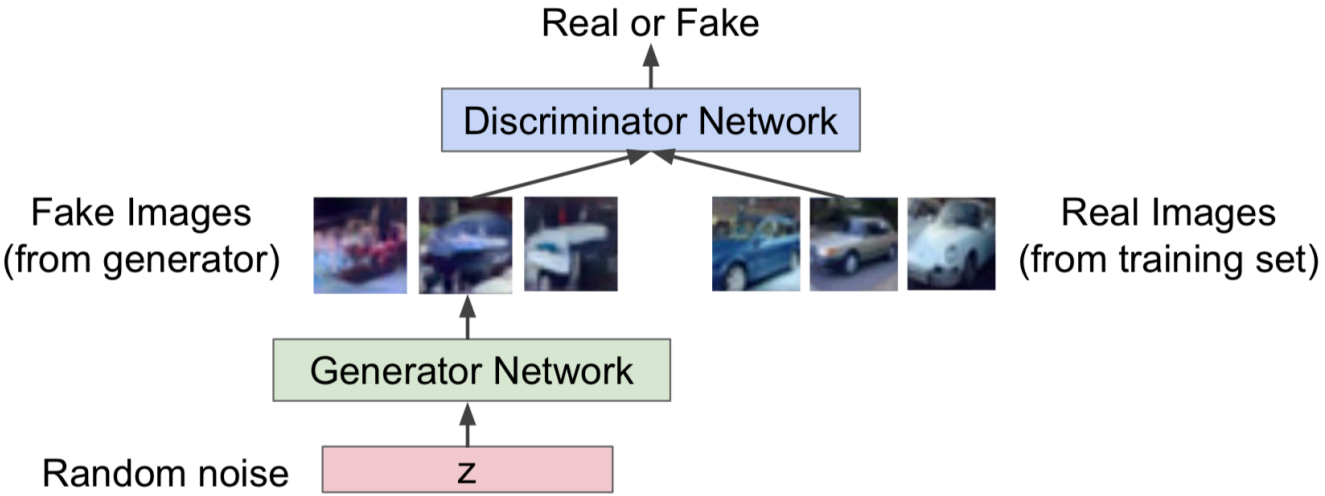


Generator: generate a realistic looking image and fool the Discriminator. The "Art forger"

Training GANs

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images



Training GANs

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**

Objective Function:

$$\min_{\theta_g} \max_{\theta_d} [E_{x \sim p_{data}} \log D_{\theta_d}(x) + E_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

Training GANs

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 the range (0,1) of real image

Objective Function:

$$\min_{\theta_g} \max_{\theta_d} [E_{x \sim p_{data}} \log D_{\theta_d}(x) + E_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

Discriminator output for real data

Training GANs

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 the range (0,1) of real image

Objective Function:

$$\min_{\theta_g} \max_{\theta_d} [E_{x \sim p_{data}} \log D_{\theta_d}(x) + E_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

Discriminator output for real data

Discriminator output for generated data

Training GANs

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 the range (0,1) of real image

Objective Function:

$$\min_{\theta_g} \max_{\theta_d} [E_{x \sim p_{data}} \log D_{\theta_d}(x) + E_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

Discriminator output for real data

Discriminator output for generated data

- **Discriminator(D)** wants to maximize objective function such that D(x) is close to 1 (real) and D(G(z)) is close to 0(fake)
- **Generator(G)** wants to minimize objective function such that D(G(Z)) is close to 1(real)

Training GANs

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Slides credit: Stanford cs231n

Training GANs

Note! Due to poor gradient signals for the generator, we use a different gradient step

We optimize the generator for **gradient ascent:**

$$\max_{\theta_g} E_{z\sim p(z)}[\log(D_{\theta_d}(G_{\theta_g}(z)))]$$

Details in original paper, Goodfellow et. al. (NIPS 2014)

Training GANs

Often k=1

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:

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

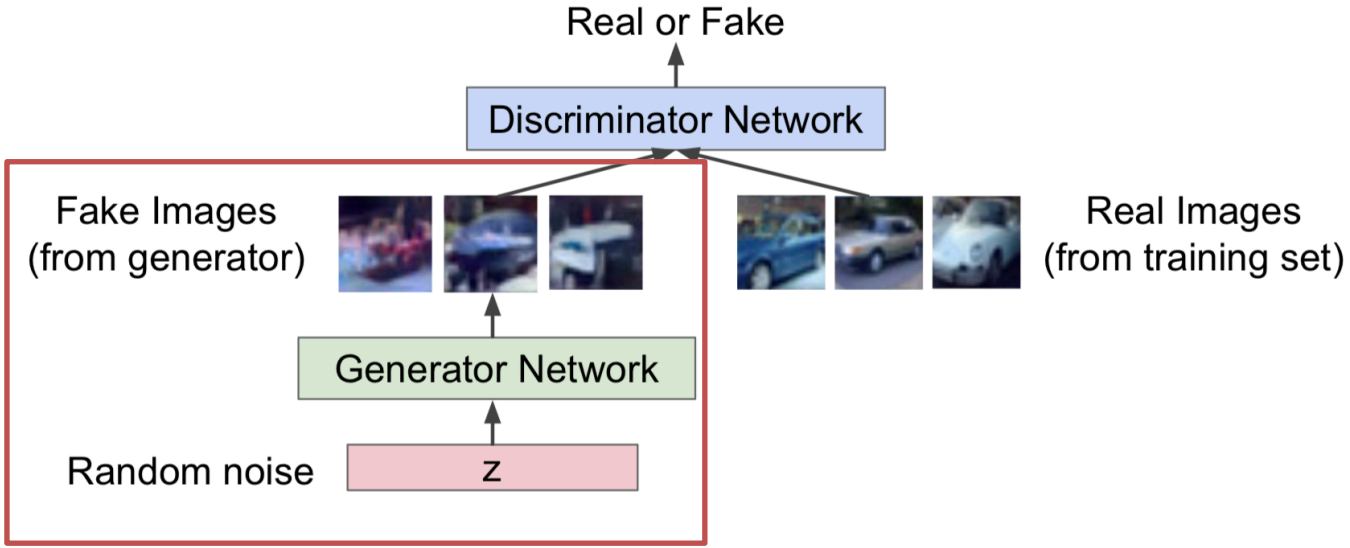
  end for

  - 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:

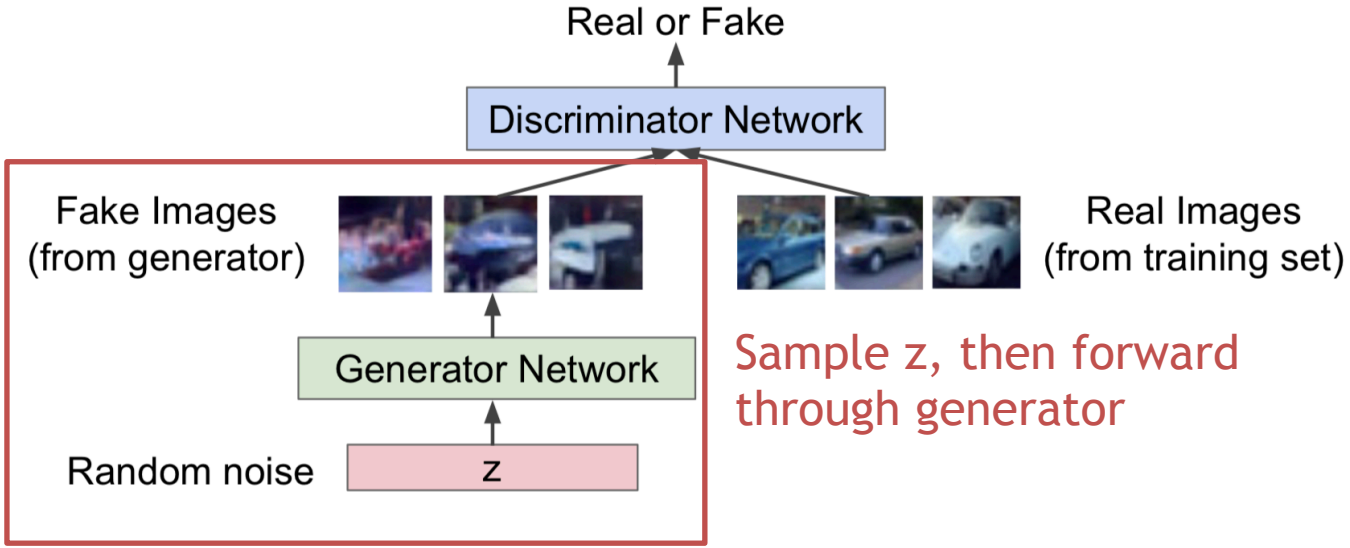
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

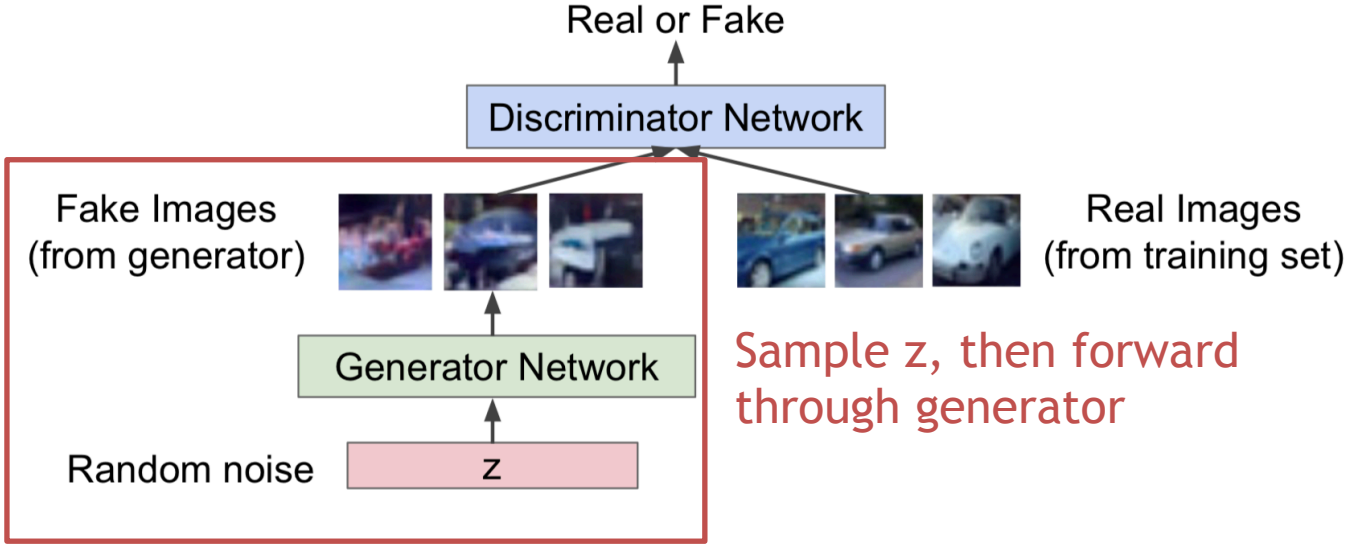
Generating new samples



Generating new samples

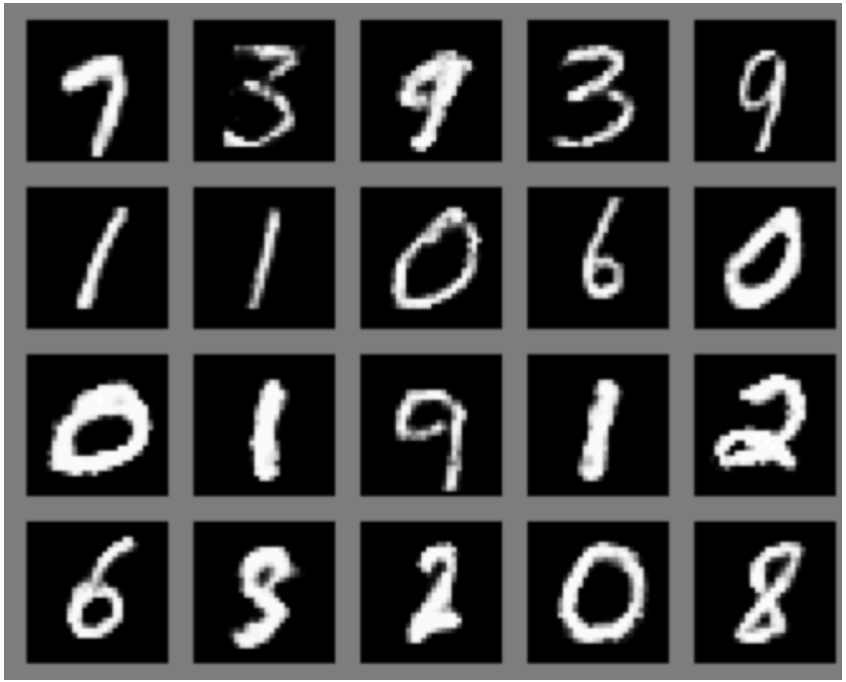


Generating new samples

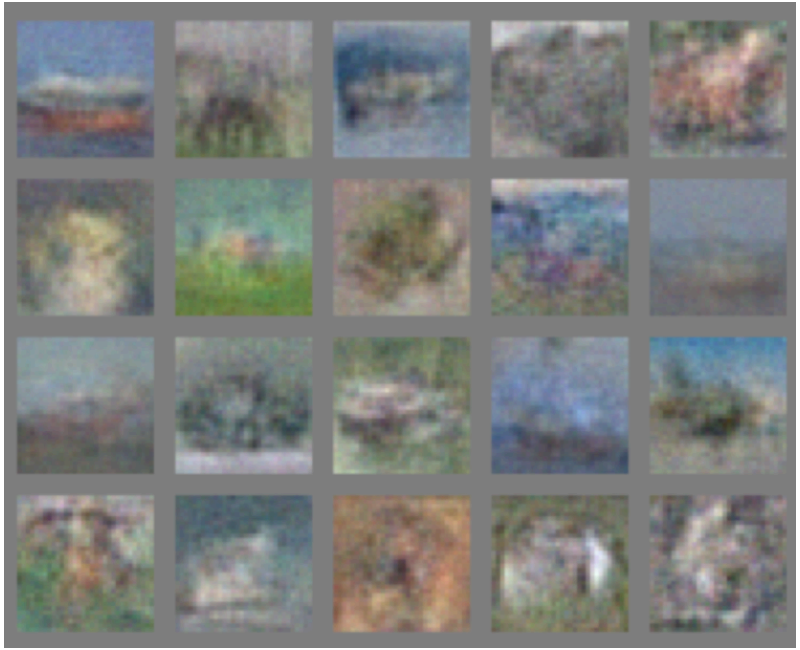


Often, the Discriminator is not useful for any task except training

GANs results: MNIST



GANs results: CIFAR-10



GANs results: TFD

Original GAN problems #1

The original GAN architecture was notoriously hard to train

It required careful hyperparameter initialization and careful training

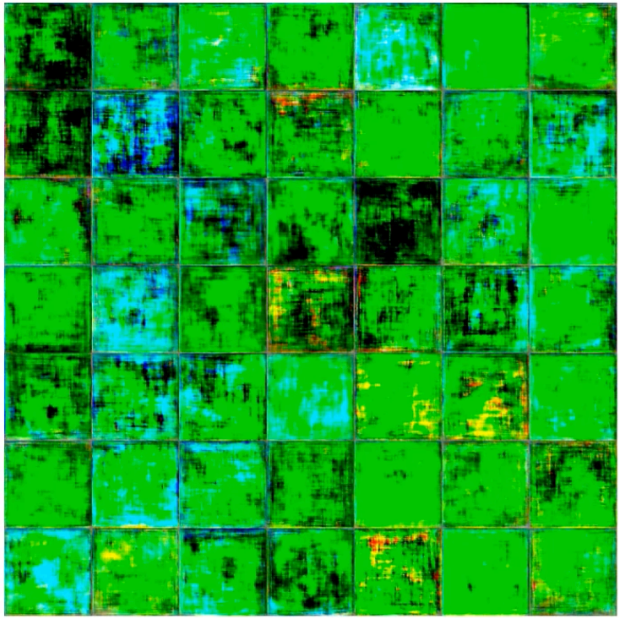
Training for higher resolution was **practically impossible**

Original GAN problems #1

Real images: Left
Generated images: Right

Mode Collapse

Start of training



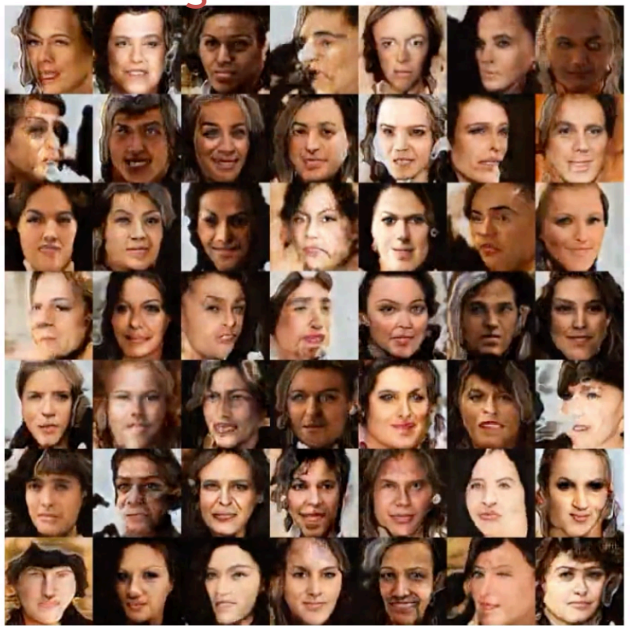
Slide Source: David Luebke

Original GAN problems #1

Real images: Left
Generated images: Right

Mode Collapse

"Middle" of training



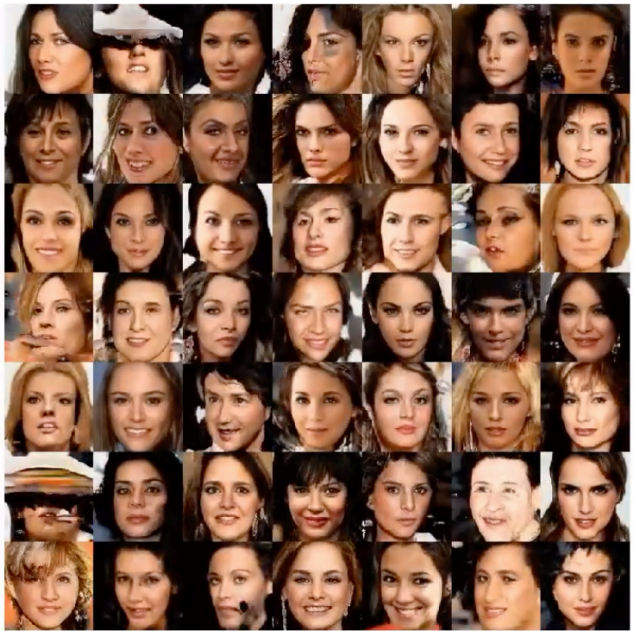
Slide Source: David Luebke

Original GAN problems #1

Real images: Left
Generated images: Right

Mode Collapse

Starting to loose variation



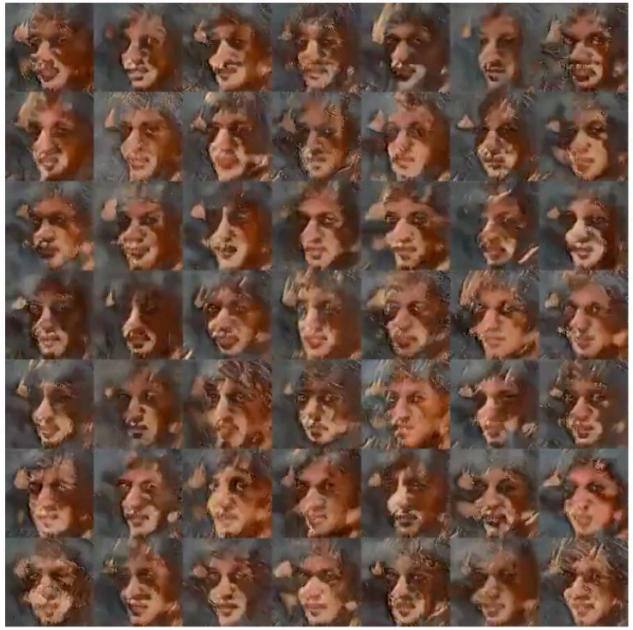
Slide Source: David Luebke

Original GAN problems #1

Real images: Left
Generated images: Right

Mode Collapse

....



Slide Source: David Luebke

Original GAN problems #1

Mode Collapse

What happens?

A lot of research on this…

“Simple” explanation:

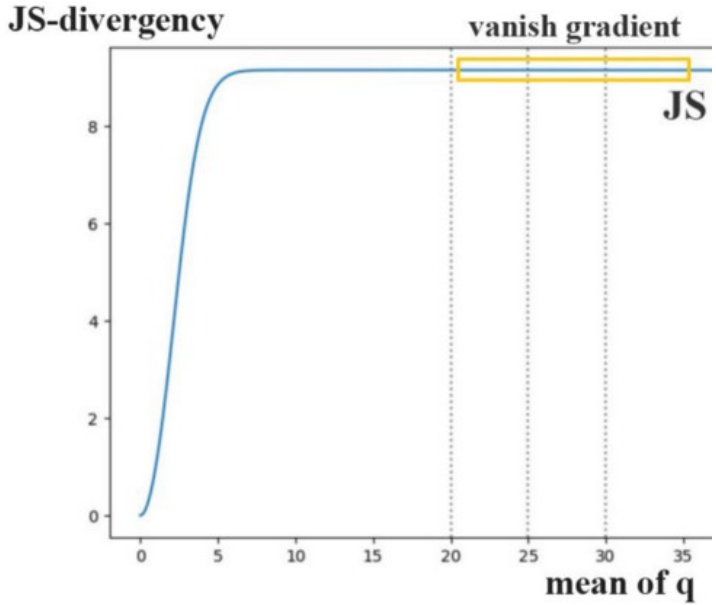
Discriminator starts to mislead the generator, and collapses over time

Original GAN problems #2

Vanishing Gradients

The original GAN loss minimized the Jenson-Shannon divergence.

But the JS-divergence struggles with vanishing gradients



Original GAN problems #2

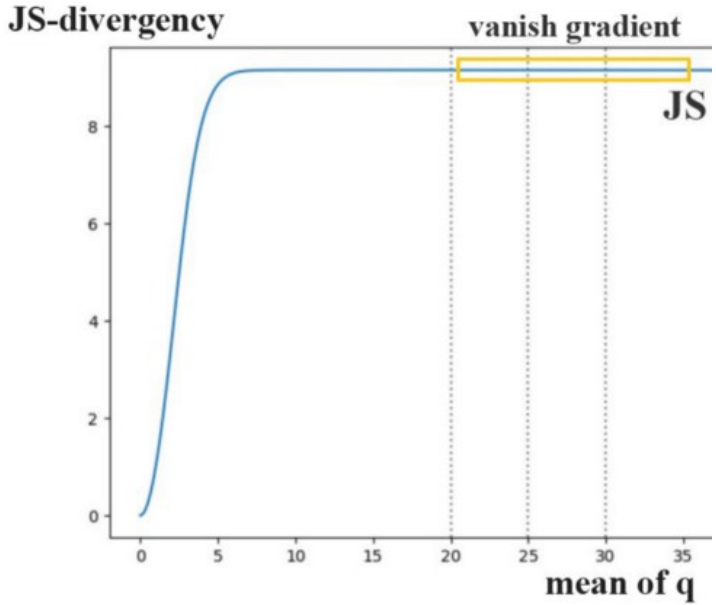
Vanishing Gradients

The original GAN loss minimized the Jenson-Shannon divergence.

But the JS-divergence struggles with vanishing gradients

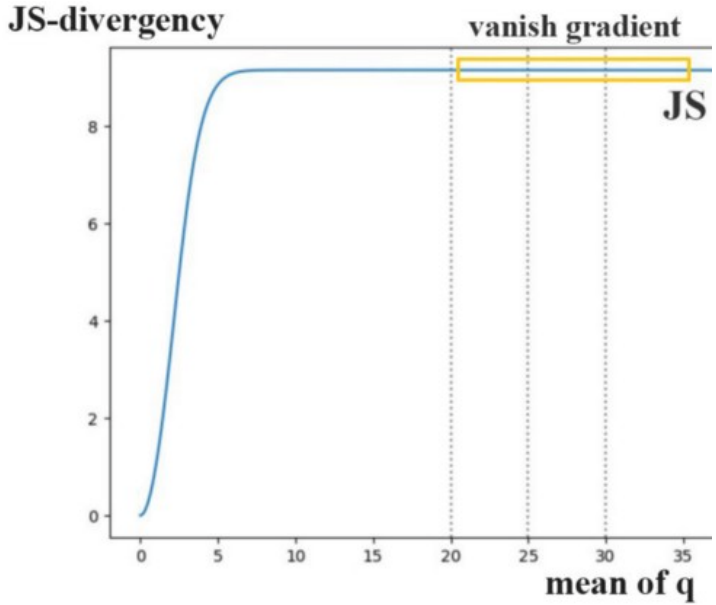
What happens?

- Generator gradient is close to 0
- Discriminator continues to improve



Original GAN problems #2

Vanishing Gradients



When does it happen?

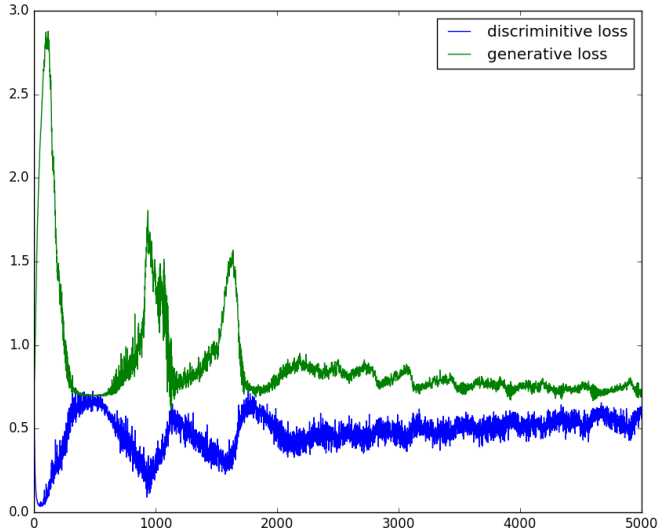
- If the discriminator is much “better” than the generator.
- When the generator is generating samples that does not resemble the original images.

Original GAN problems #3

How do we measure performance?

The GAN loss function only measures how good the discriminator is vs the generator.

Has **no meaning** for a human…

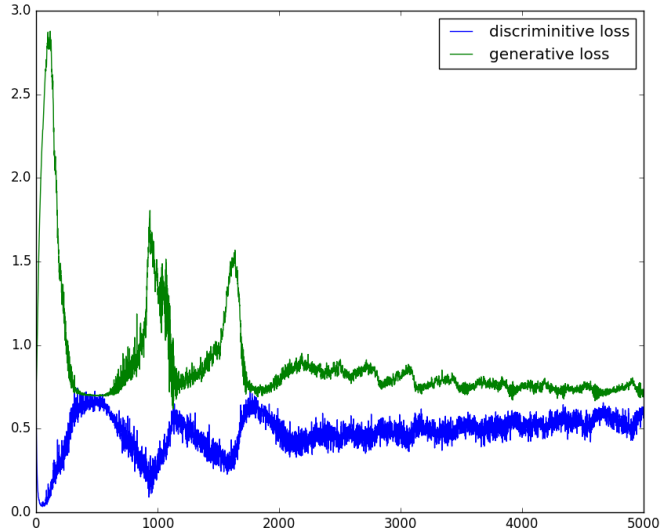


Original GAN problems #3

How do we measure performance?

The GAN loss function only measures how good the discriminator is vs the generator.

Has **no meaning** for a human…



I'll get back to this later

Wasserstein Distance

A replacement to the original GAN loss function

Use Earth-mover distance instead

“Wasserstein GAN”, Arjovsky et. al.

“Improved Training of Wasserstein GANs”, Gulrajani et. al.

Wasserstein Distance

Pros:

- Removes the problem of **vanishing gradients.**
- **Mode Collapse** is a less severe problem.
- Overall training is more stable.
- Wasserstein loss often relates to overall image quality

Cons:

- Unable to converge to global minima, but OK in practice

Wasserstein Distance

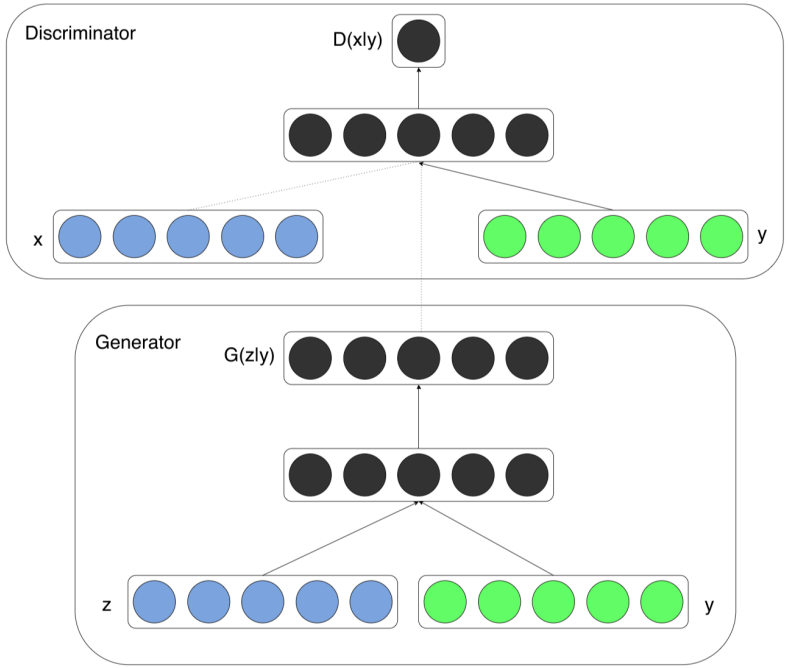
For those who's gonna try it in their project:

Use the loss function from
"Improved training of Wasserstein GANs"

Known as "Wasserstein with Gradient Penalty" or WGAN-GP

Conditional GAN

Original GAN has no control over modes of data being generated



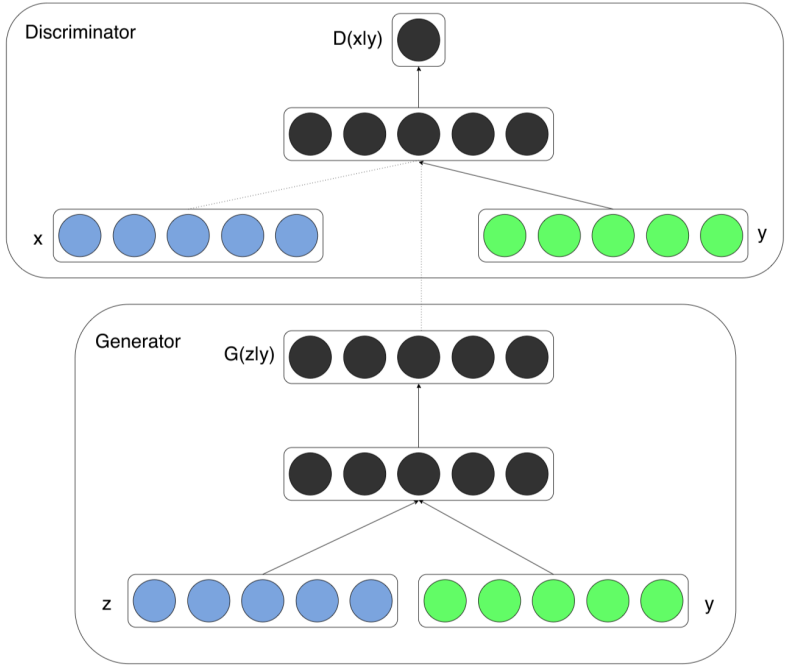
"Conditional Generative Adversarial Nets", Mirza et al.

Conditional GAN

Original GAN has no control over modes of data being generated

Can introduce **conditional information**:

- **y:** conditional information
- Can be image label, semantic maps, pose information etc.



"Conditional Generative Adversarial Nets", Mirza et al.

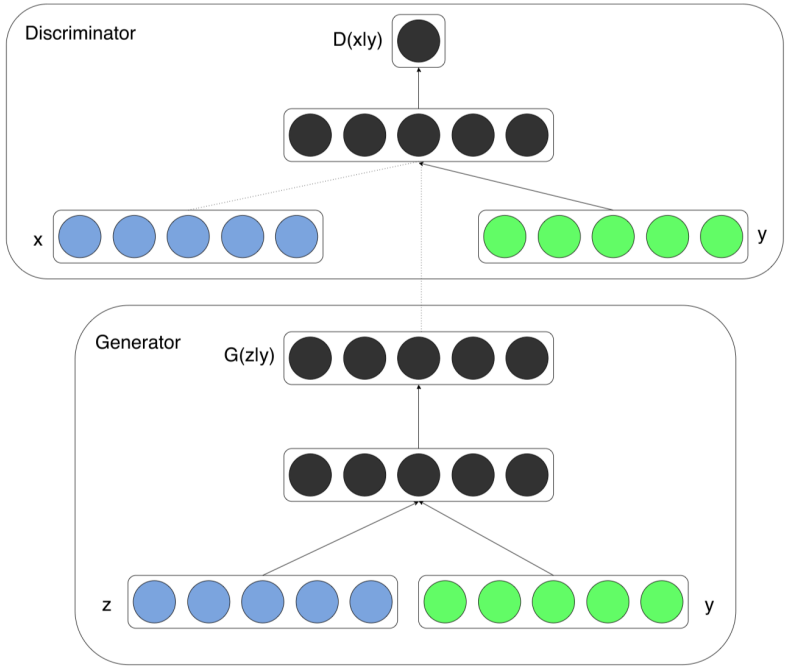
Conditional GAN

Pros:

- More control over generated data
- Generally helps convergence

Cons:

- Requires labeled data.



"Conditional Generative Adversarial Nets", Mirza et al.

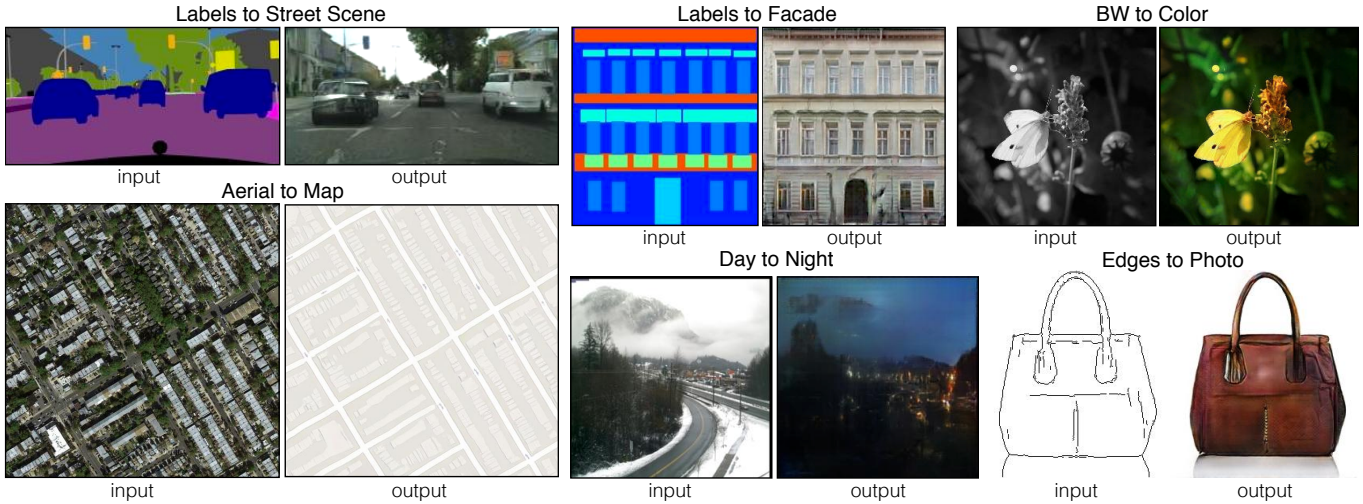
Pix2Pix

Image-to-Image Translation with Conditional Adversarial Networks

Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros

Berkeley AI Research (BAIR) Laboratory, UC Berkeley

{isola,junyanz,tinghuiz,efros}@eecs.berkeley.edu



NTNU

Pix2Pix Loss Function

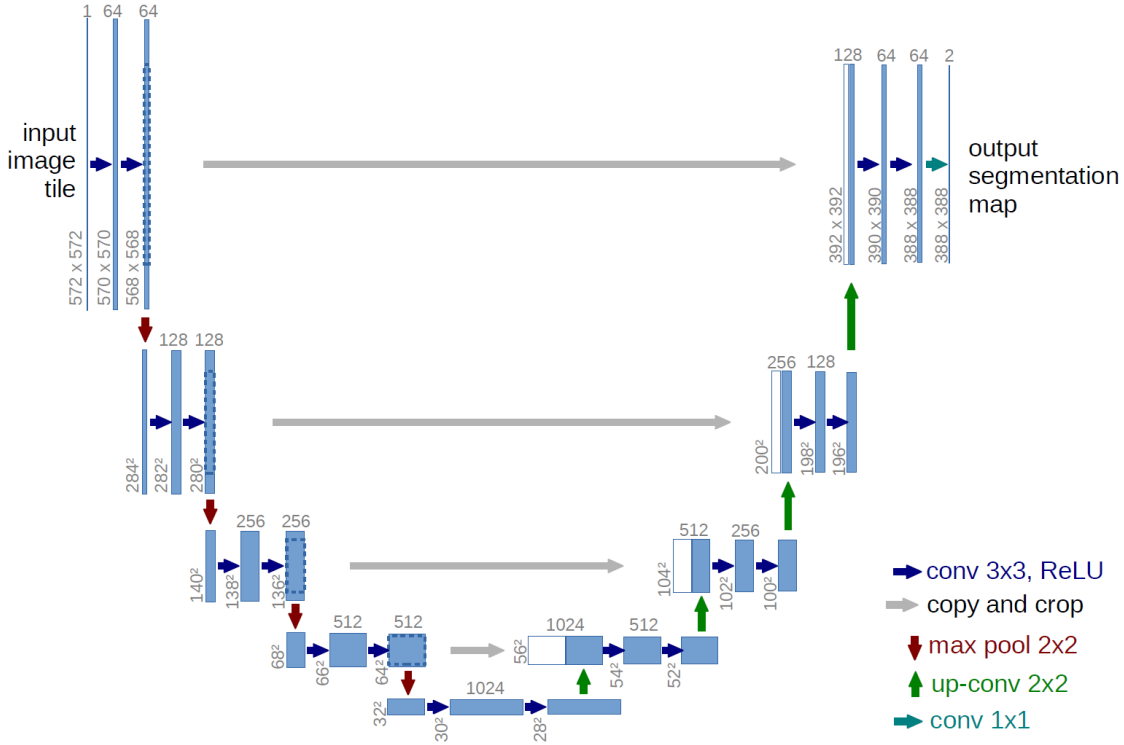
Standard GAN loss + L1 Loss

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G). \qquad (4)$$

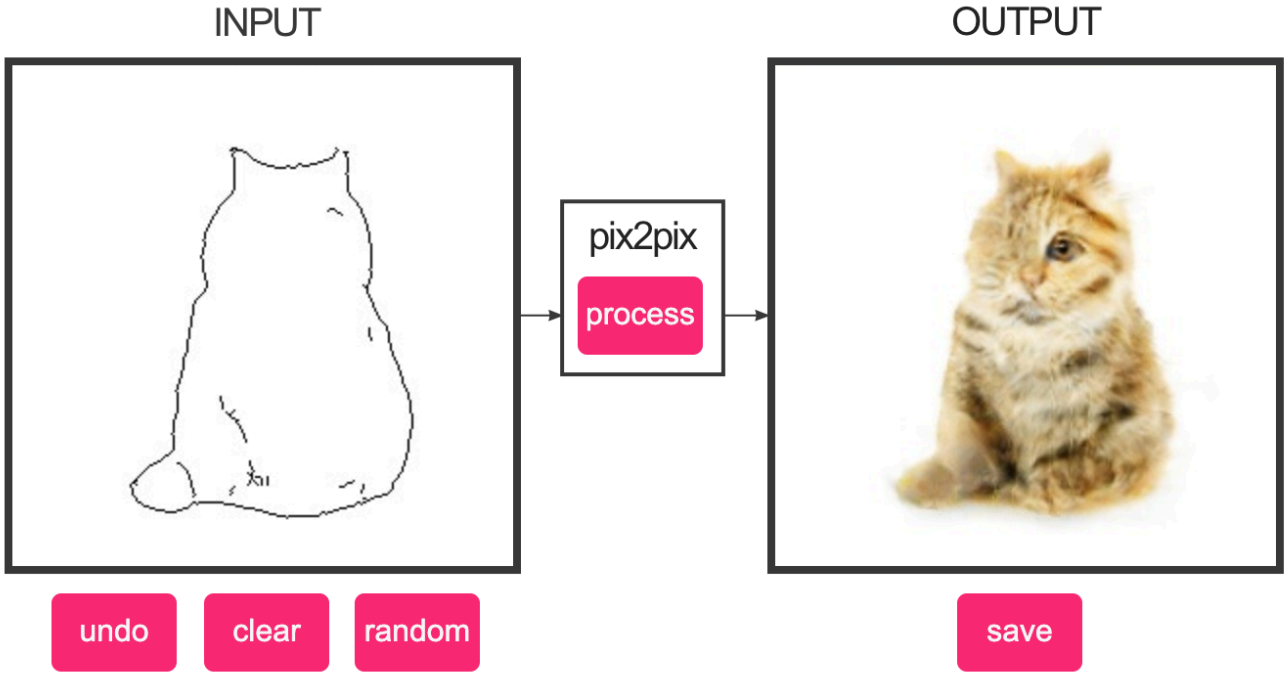
Works good for Pix2Pix

Can cause blurry images.

Pix2Pix Generator Architecture

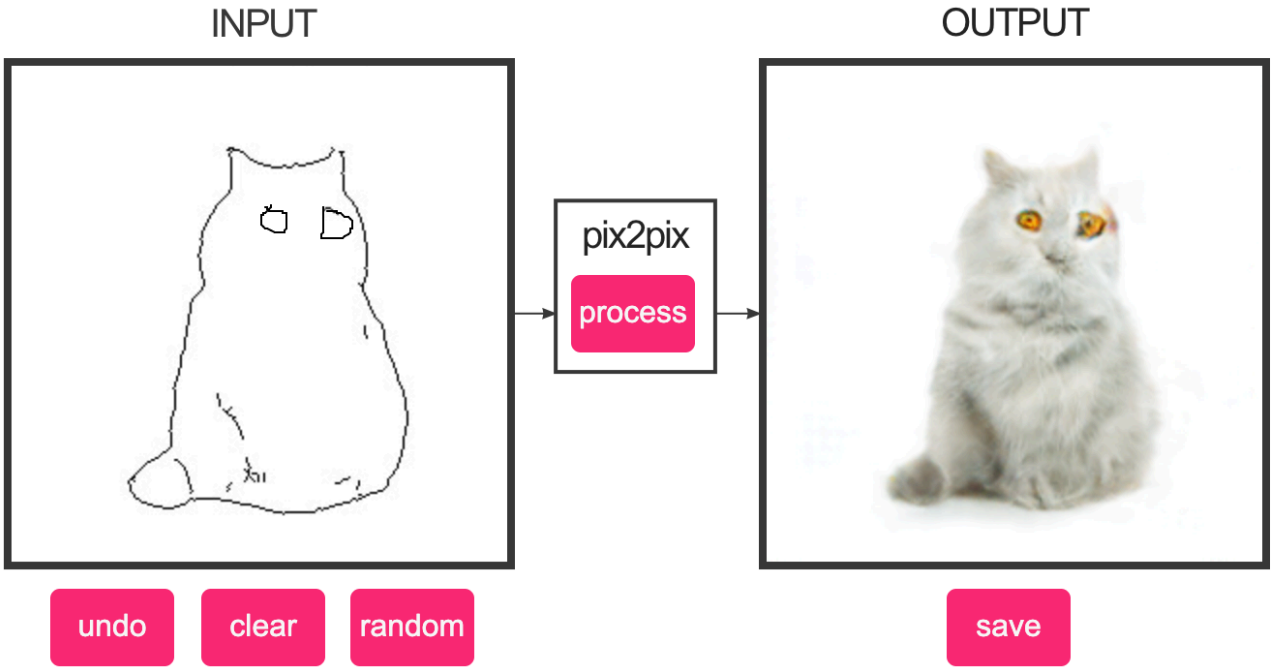


Pix2Pix: Interactive Playground



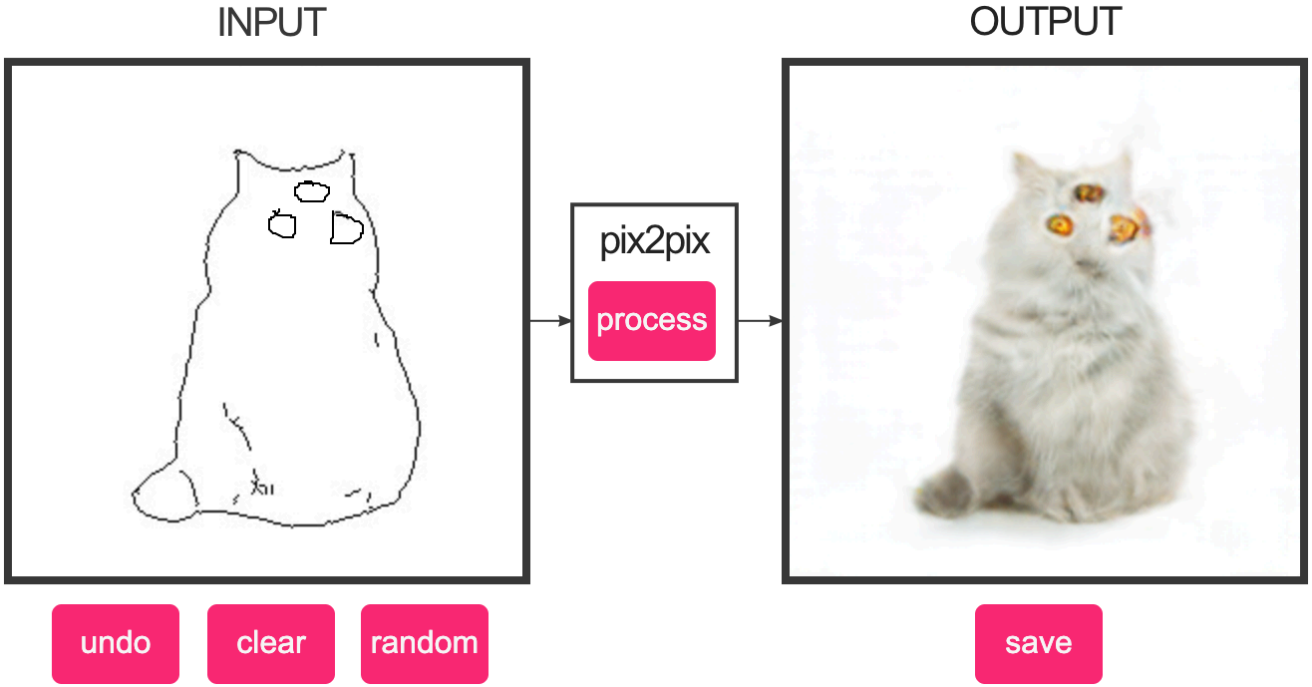
https://affinelayer.com/pixsrv/

Pix2Pix: Interactive Playground



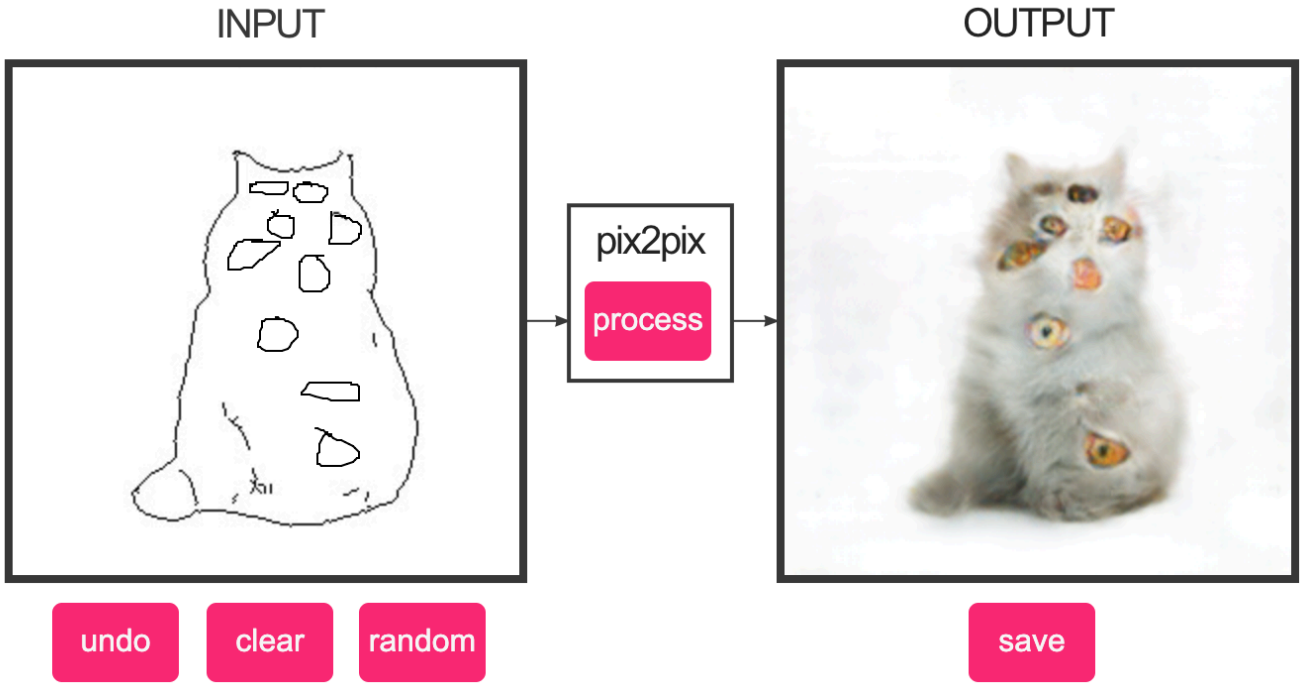
https://affinelayer.com/pixsrv/

Pix2Pix: Interactive Playground



https://affinelayer.com/pixsrv/

Pix2Pix: Interactive Playground



https://affinelayer.com/pixsrv/

Progressive Growing of GANs for Improved Quality, Stability, and Variation

Tero Karras, Samuli Laine, Timo Aila, Jaakko Lehtinen

Progressive Growing GANs

https://github.com/tkarras/progressive_growing_of_gans

ProGAN

Generating HD images was close to impossible before this paper

ProGAN Intuition

Curriculum learning is easier

- A human needs to learn addition before multiplication
- A GAN needs to learn basic shapes before complex textures

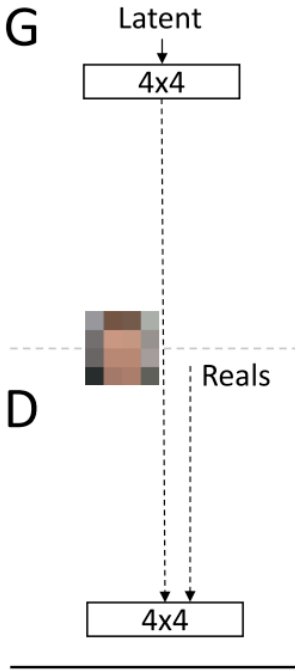
And it works in practice!

CelebA-HQ

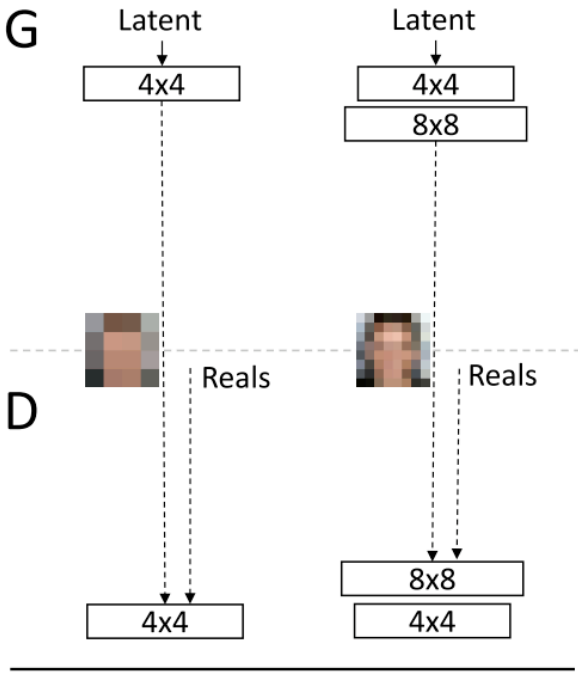
1024 × 1024

Progressive growing

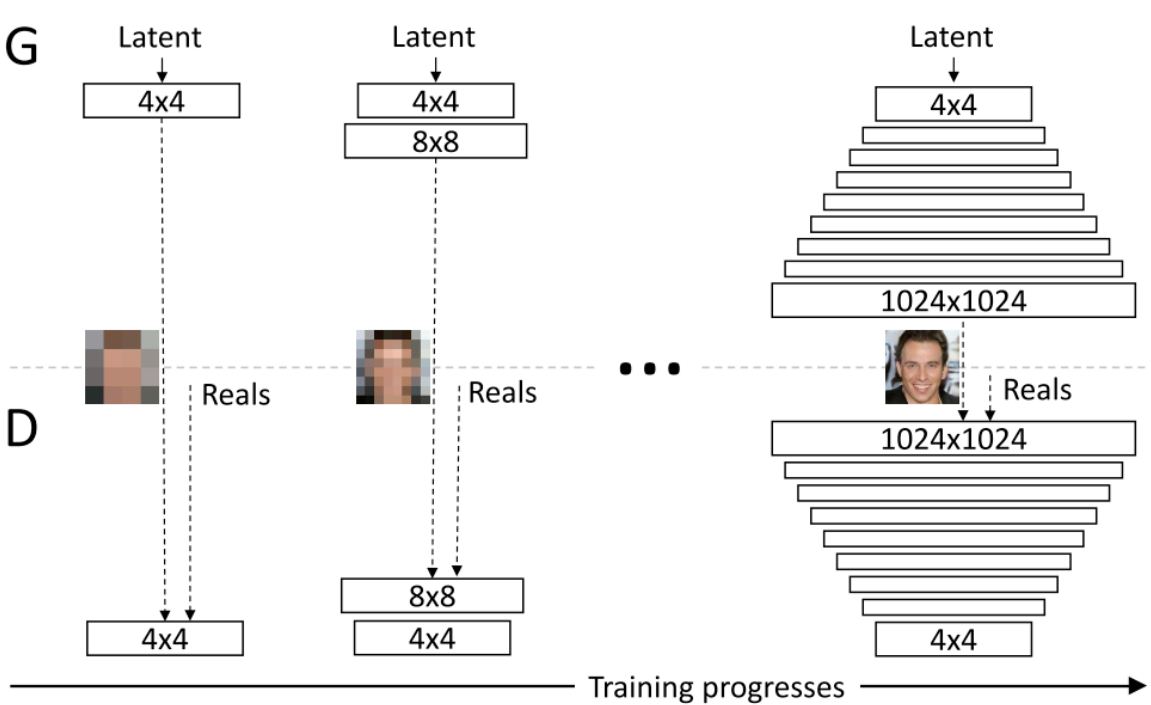
ProGAN



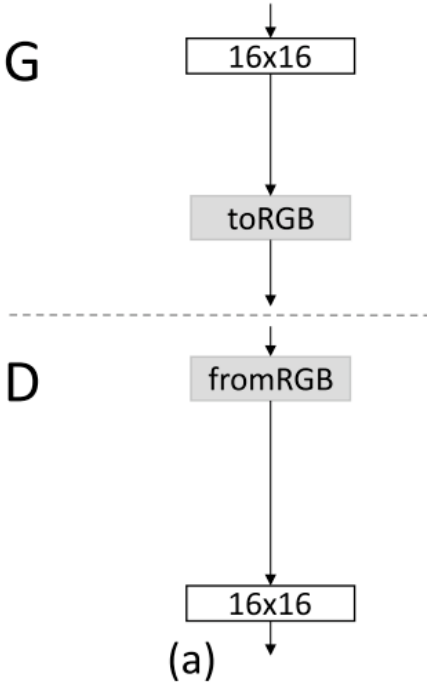
ProGAN



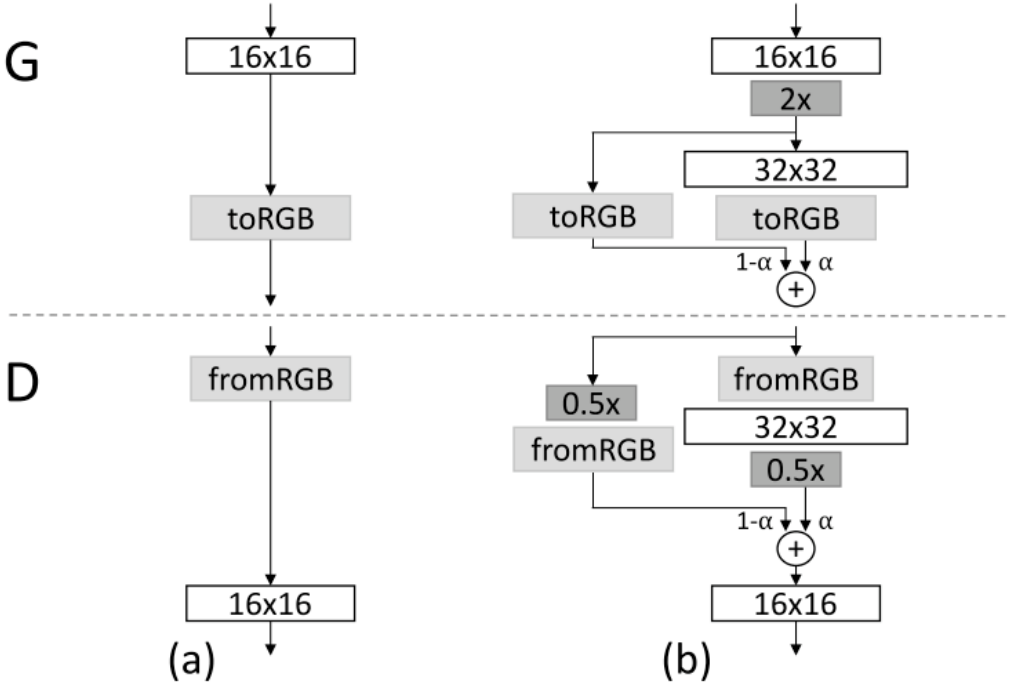
ProGAN


G
Latent
4x4
Latent
4x4
8x8
Latent
4x4
1024x1024
Reals
Reals
Reals
D
4x4
8x8
4x4
1024x1024
4x4
Training progresses

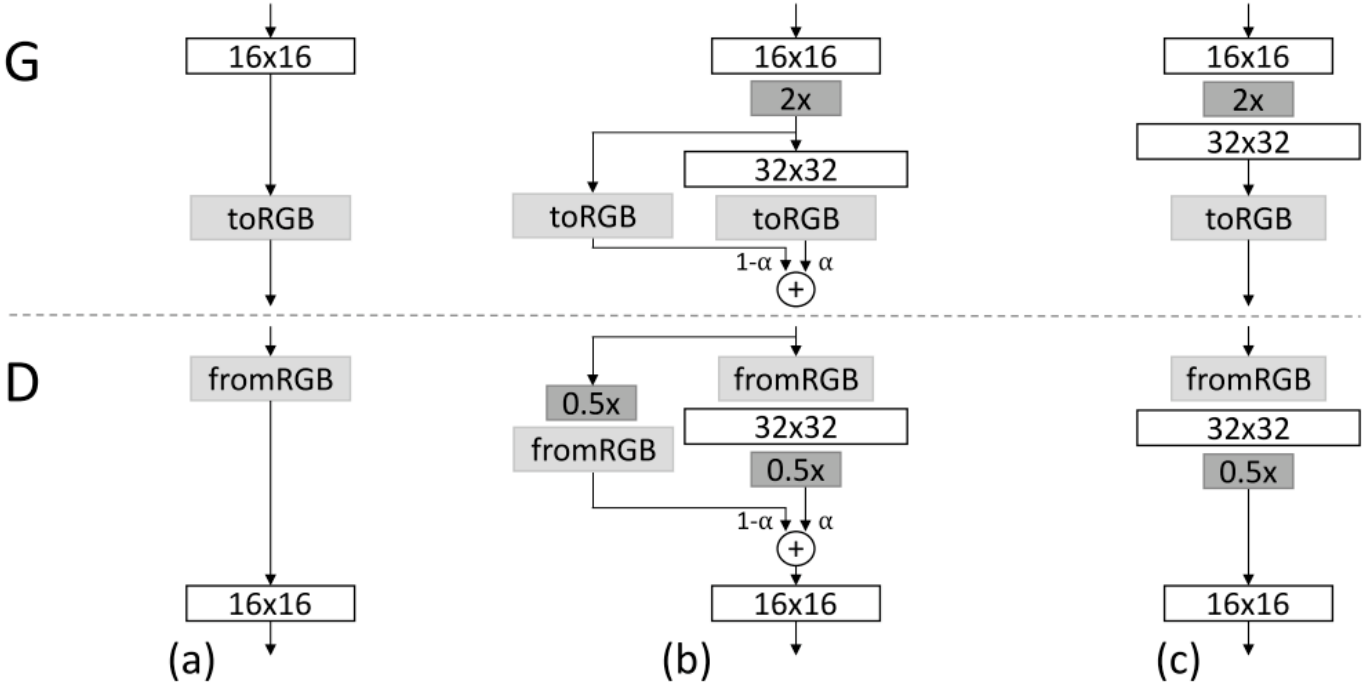
ProGAN Transition Phase



ProGAN Transition Phase



ProGAN Transition Phase



ProGAN

Pros:

- 2-6 time faster training time
- Improved image quality
- Stable training

Cons:

- No control over generated attributes such as smile, skin color, etc. (Fixed in StyleGAN)
- 1024x1024 images requires large GPU VRAM

Large Scale GAN Training for High Fidelity Natural Image Synthesis

Andrew Brock, Jeff Donahue, Karen Simonyan

BigGAN

BigGAN

“Improving the state-of-the-art by 300%”

BigGAN sidetrack: How can we measure the image quality?

Two standard performance metrics:

- Inception Score
- Freichet Inception Distance

Both tries to evaluate the generated images quality vs the

BigGAN sidetrack: Inception Score(IS)

The IS metric measures the image quality by checking:

1. A CNN should be highly confident that there is a **single object** in the image.
2. Over all the generated images, there should be a **large variance** in number of classes generated

This is measured with an InceptionV3 network:

1. p(y|x) should be very confident for a single class and close to 0 for rest.
2. For all the images, p(y|x) should vary a lot. In other words, p(y) for all y should be high

BigGAN sidetrack: Inception Score(IS)

The IS metric measures the image quality by checking:

1. A CNN should be highly confident that there is a **single object** in the image.
2. Over all the generated images, there should be a **large variance** in number of classes generated

This is measured with an InceptionV3 network:

1. p(y|x) should be very confident for a single class and close to 0 for rest.
2. For all the images, p(y|x) should vary a lot. In other words, p(y) for all y should be high

Formal definition:

$$\boldsymbol{IS}(G) = exp(E_{x \sim p_g}[KL(p(y \mid \boldsymbol{x}) \mid\mid p(y))])$$

BigGAN

Batch	Ch.	Param (M)	Shared	Hier.	Ortho.	Itr $\times 10^3$	FID	IS
256	64	81.5	SA-GAN Baseline			1000	18.65	52.52
512	64	81.5	✗	✗	✗	1000	15.30	58.77($\pm$1.18)
1024	64	81.5	✗	✗	✗	1000	14.88	63.03($\pm$1.42)
2048	64	81.5	✗	✗	✗	732	12.39	76.85($\pm$3.83)
2048	96	173.5	✗	✗	✗	295($\pm$18)	9.54($\pm$0.62)	92.98($\pm$4.27)
2048	96	160.6	✓	✗	✗	185($\pm$11)	9.18($\pm$0.13)	94.94($\pm$1.32)
2048	96	158.3	✓	✓	✗	152($\pm$7)	8.73($\pm$0.45)	98.76($\pm$2.84)
2048	96	158.3	✓	✓	✓	165($\pm$13)	8.51($\pm$0.32)	99.31($\pm$2.10)
2048	64	71.3	✓	✓	✓	371($\pm$7)	10.48($\pm$0.10)	86.90($\pm$0.61)

BigGAN

Increasing batch size: 46% better IS score

Batch	Ch.	Param (M)	Shared	Hier.	Ortho.	Itr $\times 10^3$	FID	IS
256	64	81.5	SA-GAN Baseline			1000	18.65	52.52
512	64	81.5	✗	✗	✗	1000	15.30	58.77($\pm$1.18)
1024	64	81.5	✗	✗	✗	1000	14.88	63.03($\pm$1.42)
2048	64	81.5	✗	✗	✗	732	12.39	76.85($\pm$3.83)
2048	96	173.5	✗	✗	✗	295($\pm$18)	9.54($\pm$0.62)	92.98($\pm$4.27)
2048	96	160.6	✓	✗	✗	185($\pm$11)	9.18($\pm$0.13)	94.94($\pm$1.32)
2048	96	158.3	✓	✓	✗	152($\pm$7)	8.73($\pm$0.45)	98.76($\pm$2.84)
2048	96	158.3	✓	✓	✓	165($\pm$13)	8.51($\pm$0.32)	99.31($\pm$2.10)
2048	64	71.3	✓	✓	✓	371($\pm$7)	10.48($\pm$0.10)	86.90($\pm$0.61)

BigGAN

Increasing number of filters in each layer:
Additionally 21% better IS score

Batch	Ch.	Param (M)	Shared	Hier.	Ortho.	Itr $\times 10^3$	FID	IS
256	64	81.5	SA-GAN Baseline			1000	18.65	52.52
512	64	81.5	✗	✗	✗	1000	15.30	58.77($\pm$1.18)
1024	64	81.5	✗	✗	✗	1000	14.88	63.03($\pm$1.42)
2048	64	81.5	✗	✗	✗	732	12.39	76.85($\pm$3.83)
2048	96	173.5	✗	✗	✗	295($\pm$18)	9.54($\pm$0.62)	92.98($\pm$4.27)
2048	96	160.6	✓	✗	✗	185($\pm$11)	9.18($\pm$0.13)	94.94($\pm$1.32)
2048	96	158.3	✓	✓	✗	152($\pm$7)	8.73($\pm$0.45)	98.76($\pm$2.84)
2048	96	158.3	✓	✓	✓	165($\pm$13)	8.51($\pm$0.32)	99.31($\pm$2.10)
2048	64	71.3	✓	✓	✓	371($\pm$7)	10.48($\pm$0.10)	86.90($\pm$0.61)

Latent Vector Interpolation

Conditional information change

Latent vector is constant

Latent vector ~ decides pose

http://theo.io/blog/2018/11/13/biggan-interpolations/

Latent Vector Interpolation

Conditional information change

Latent vector is constant

Latent vector ~ decides pose

http://theo.io/blog/2018/11/13/biggan-interpolations/

BigGAN: Training time & TPU usage

- Q: How many TPUv3s?

BigGAN: Training time & TPU usage

- Q: How many TPUv3s?
  - 512

BigGAN: Training time & TPU usage

- Q: How many TPUv3s?
  - 512
- Q: How much will it cost to train for us?

BigGAN: Training time & TPU usage

- Q: How many TPUv3s?
  - 512
- Q: How much will it cost to train for us?
  - 8$ per TPU per hour
  - 48 Hours of training
  - ~196,000 USD

GAN Summary

- A **discriminator** and **generator** trained jointly to improve each other in a **two-player game**
- Is able to generate high quality images with a high diversity
- High resolution and lack of diversity is still an ongoing research questions

Where does GANs not work?

- GANs works great for images
- Suffers when the dataset has a high variance and limited number of images (such as CIFAR-10)
- Is **not** a good option for text or speech synthesis!
- Videos is still an active research question

Video-to-Video Synthesis

Ting-Chun Wang[1], Ming-Yu Liu[1], Jun-Yan Zhu[2], Guilin Liu[1],
Andrew Tao[1], Jan Kautz[1], Bryan Catanzaro[1]

[1]NVIDIA Corporation [2]MIT

Additional resources

Paper road-map:

- “Generative Adversarial Networks” Goodfellow et. al., NIPS 2014
- "Improved Techniques for Training GANs”, Saliman et. al.
- “Wasserstein GAN”, Arjovsky et. al.
- “Improved Training of Wasserstein GANs”, Gulrajani et. al.
- “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, Karras et. al.
- “Semantic Image Synthesis with Spatially-Adaptive Normalization”, Park et. al.

Videos:

- GAN Tutorial by Goodfellow
- Progressive Growing GAN video
- Video-to-Video Synthesis
- GTC 2019 Presentation on ProGANs
- CS231n Lecture

Additional cool stuff:

- BigGAN interpolations
- Pix2Pix playground
- BigGAN playground
- Two Minute Papers