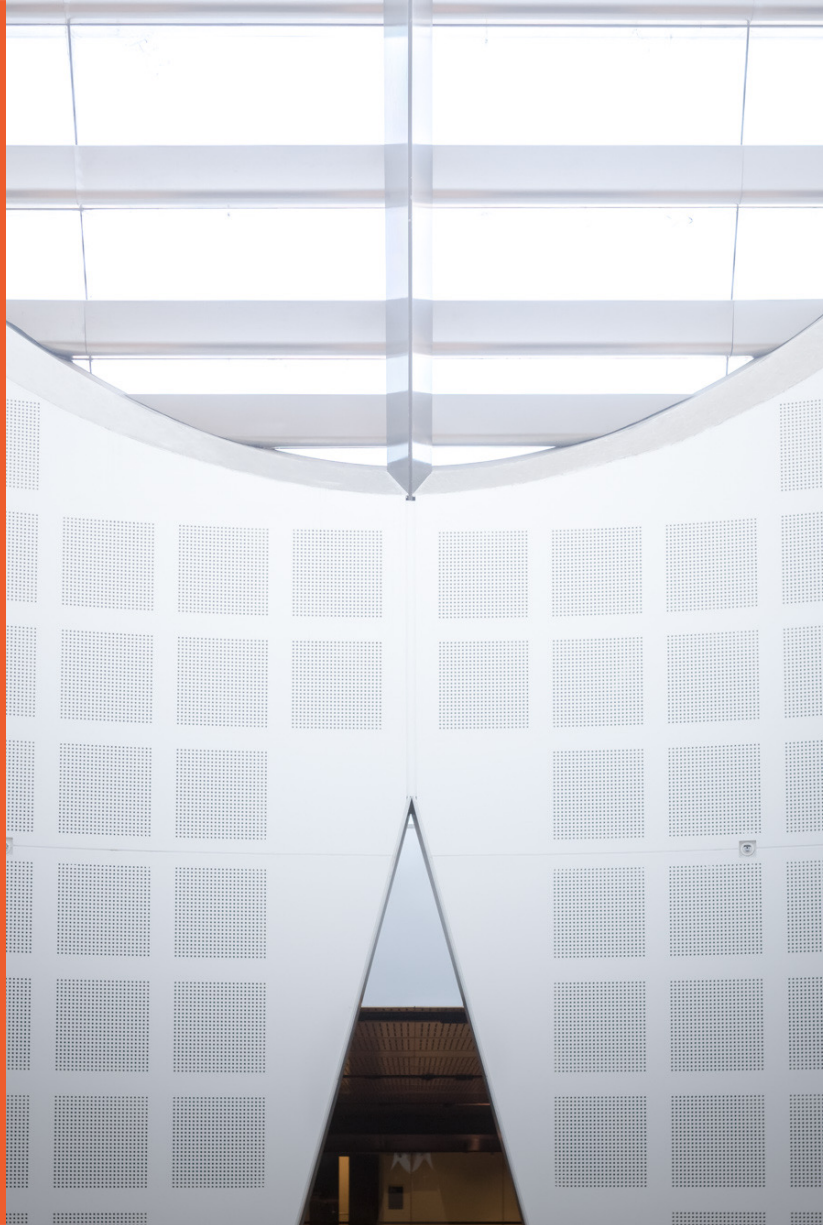
Deep Generation Models

Dr Chang Xu

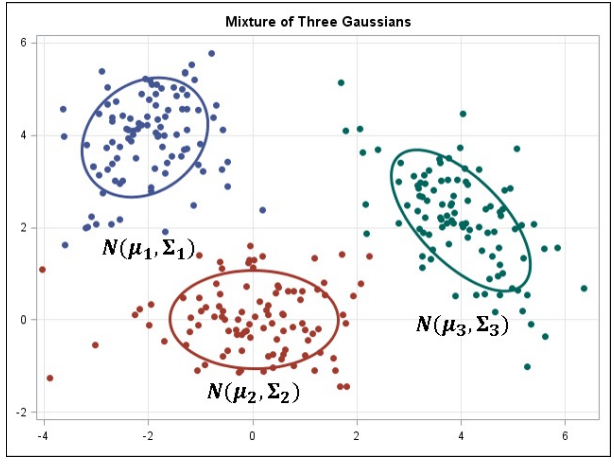
School of Computer Science

THE UNIVERSITY OF SYDNEY



Generative Modeling

- Density Estimation



- Sample Generation

Training Sample

Generated Sample

Image credit to [Ian Goodfellow, NIPS Tutorial on Generative Model 2016]

Why study generative model?

- Realistic generation tasks

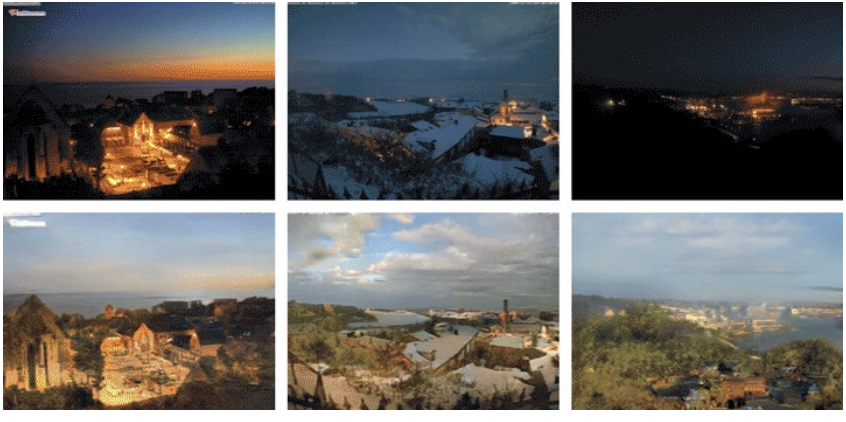


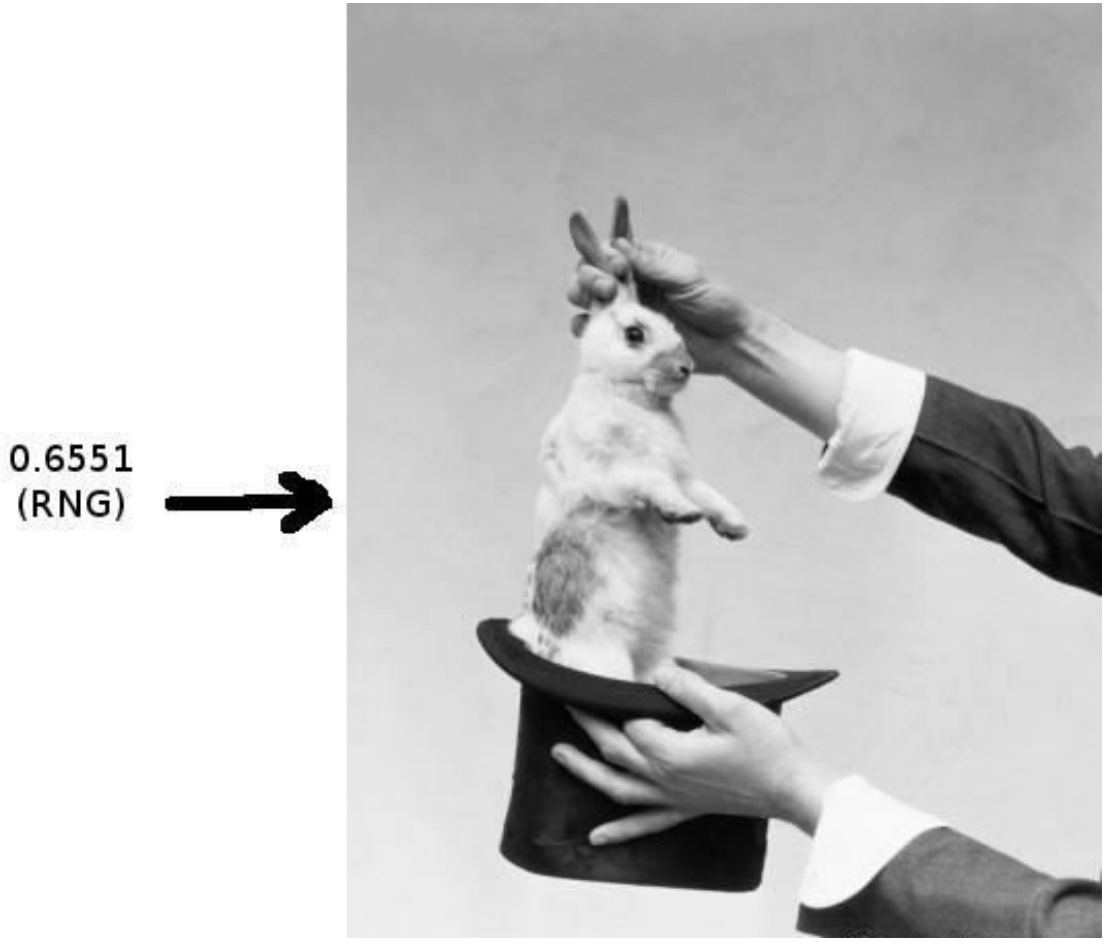
Image credit to [Jun-Yan Zhu et al. 2017]

Image credit to [Phillip Isola et al. 2017]

- Simulate possible futures for planning (e.g. Stock Market Prediction)
- Training generative models can also enable inference of latent representations that can be useful as general features

Generative Adversarial Networks

Generative Adversarial Networks



Generative Adversarial Networks

- A counterfeiter-police game between two components: a generator **G** and a discriminator **D**
- **G**: counterfeiter, trying to fool police with fake currency
- **D**: police, trying to detect the counterfeit currency
- Competition drives both to improve, until counterfeits are *indistinguishable* from genuine currency

Generative Adversarial Networks

- A two-player game between two components: a generator **G** and a discriminator **D**

- **G:** try to fool the discriminator by generating real-looking images
- **D:** try to distinguish between real and fake images

Generative Adversarial Networks

- A two-player game between two components: a generator **G** and a discriminator **D**
- **G:** try to fool the discriminator by generating real-looking images
- **D:** try to distinguish between real and fake images

score on real data

real data

min generated data

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

D predicting that real data is genuine

D predicting that G's generated data is fake

$\min \mathbb{E}[\log(1-D(G(z)))$

$\Rightarrow \max G(z)$

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

Generative Adversarial Networks

- A two-player game between two components: a generator **G** and a discriminator **D**
- **G:** try to fool the discriminator by generating real-looking images
- **D:** try to distinguish between real and fake images

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

D predicting that real data is genuine

D predicting that G's generated data is fake

D's goal: maximize objective such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)

G's goal: minimize objective such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Generative Adversarial Networks

- A two-player game between two components: a generator **G** and a discriminator **D**
- **G:** try to fool the discriminator by generating real-looking images
- **D:** try to distinguish between real and fake images

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

D predicting that real data is genuine

D predicting that G's generated data is fake

Q: What can we use to represent D and G?

A: Neural networks!

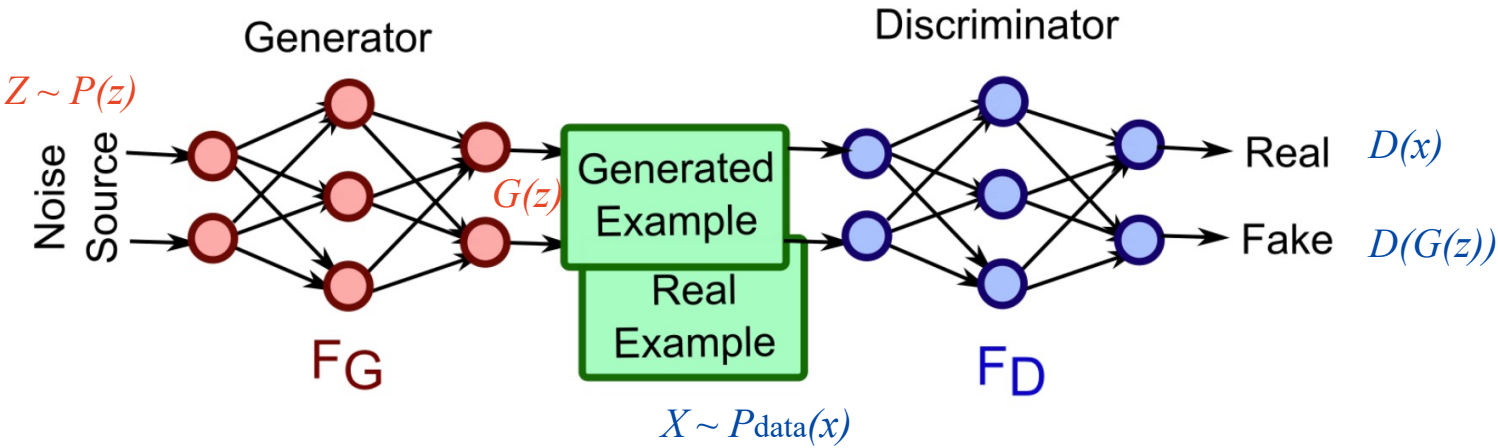
Generative Adversarial Networks

- A two-player game between two components: a generator **G** and a discriminator **D**

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

D predicting that real data is genuine

D predicting that G's generated data is fake



Generative Adversarial Networks

- A two-player game between two components: a generator **G** and a discriminator **D**

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- Alternate between:
  1. Gradient ascent on **D**

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

  2. Gradient descent on **G**

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

Algorithm

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

  - 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:

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

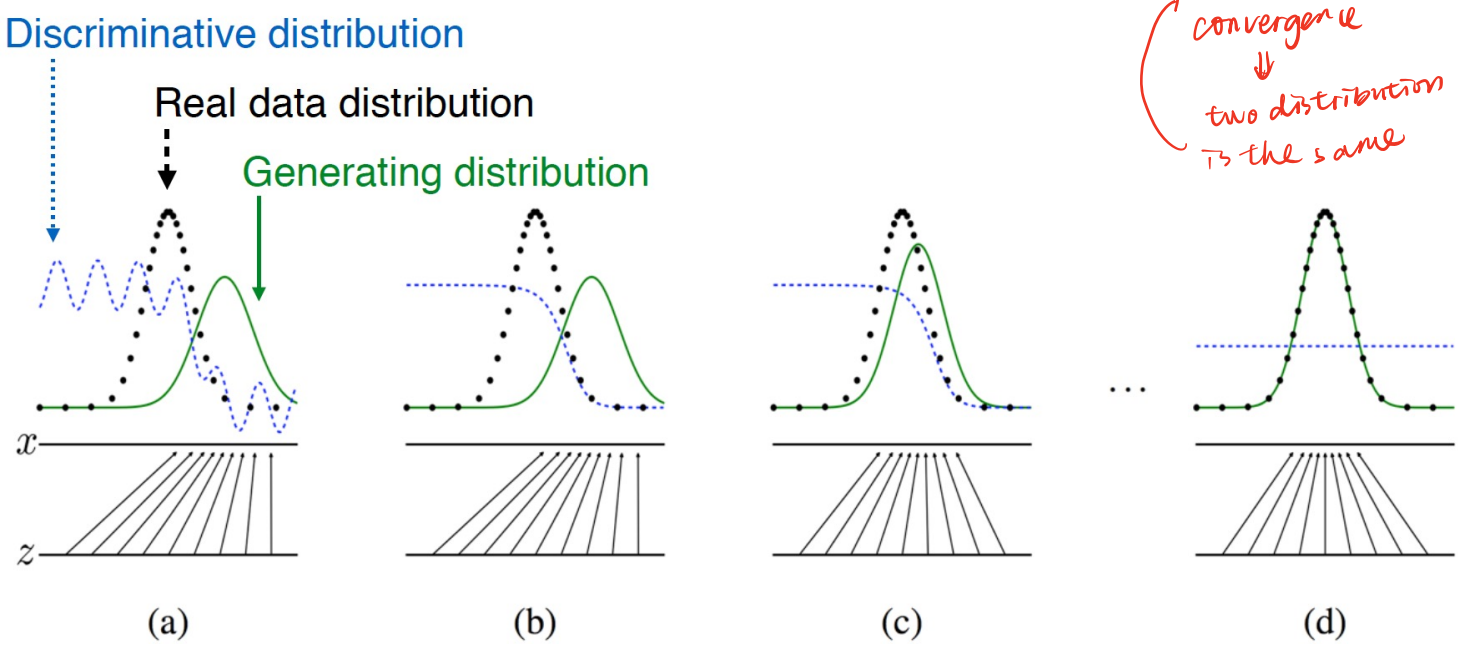
end for

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

A simple example

D output 1/2.

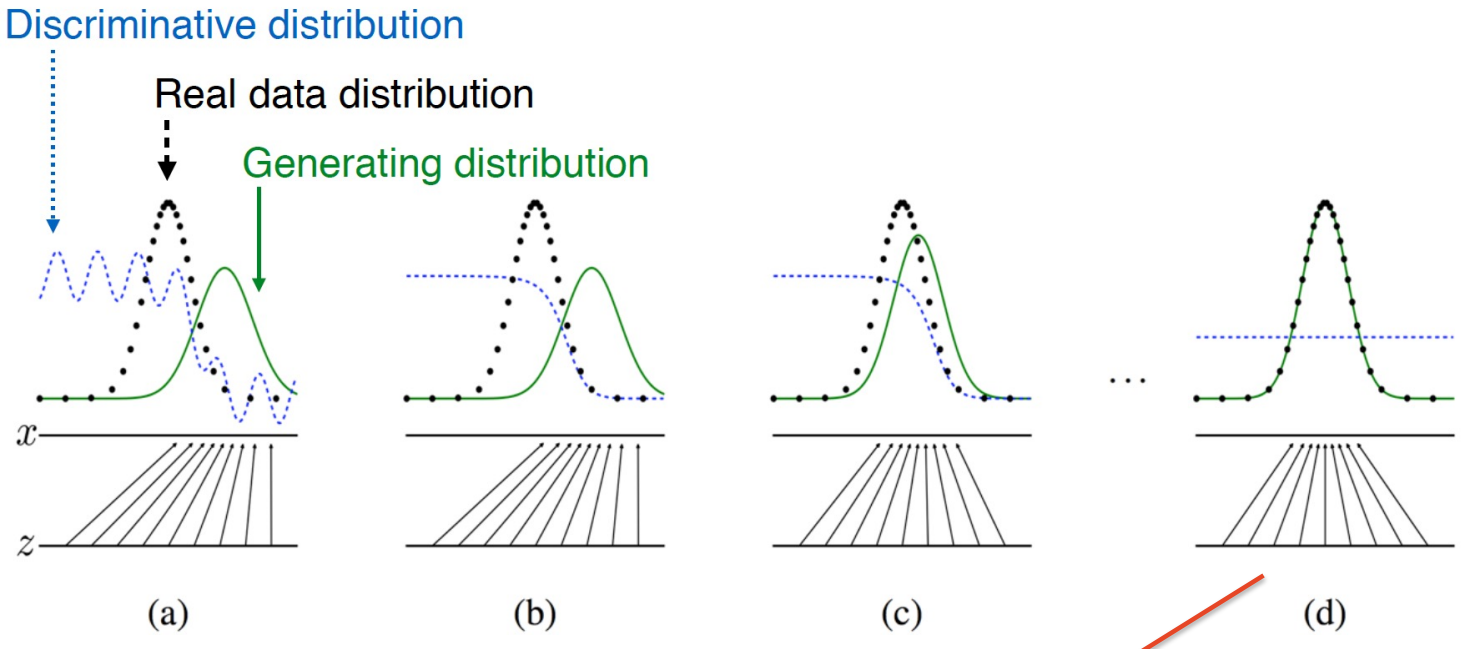
convergence ⇓ two distribution is the same



Generative adversarial nets are trained by simultaneously updating the discriminative distribution (blue line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution p_g (green line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows show how the mapping x = G(z) imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g.

Image credit to [Ian Goodfellow et al., Generative Adversarial Nets, NIPS 2016]

A simple example

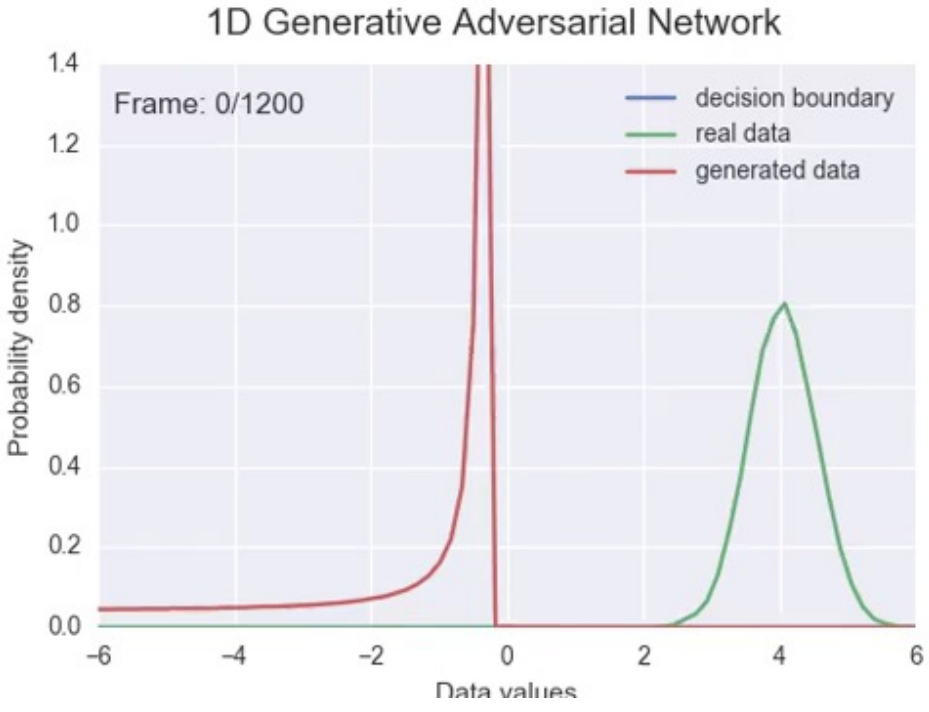


After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{data}$. The discriminator is unable to differentiate between the two distributions, i.e. **D(x) = 1/2** .

Image credit to [Ian Goodfellow et al., Generative Adversarial Nets, NIPS 2016]

A simple example

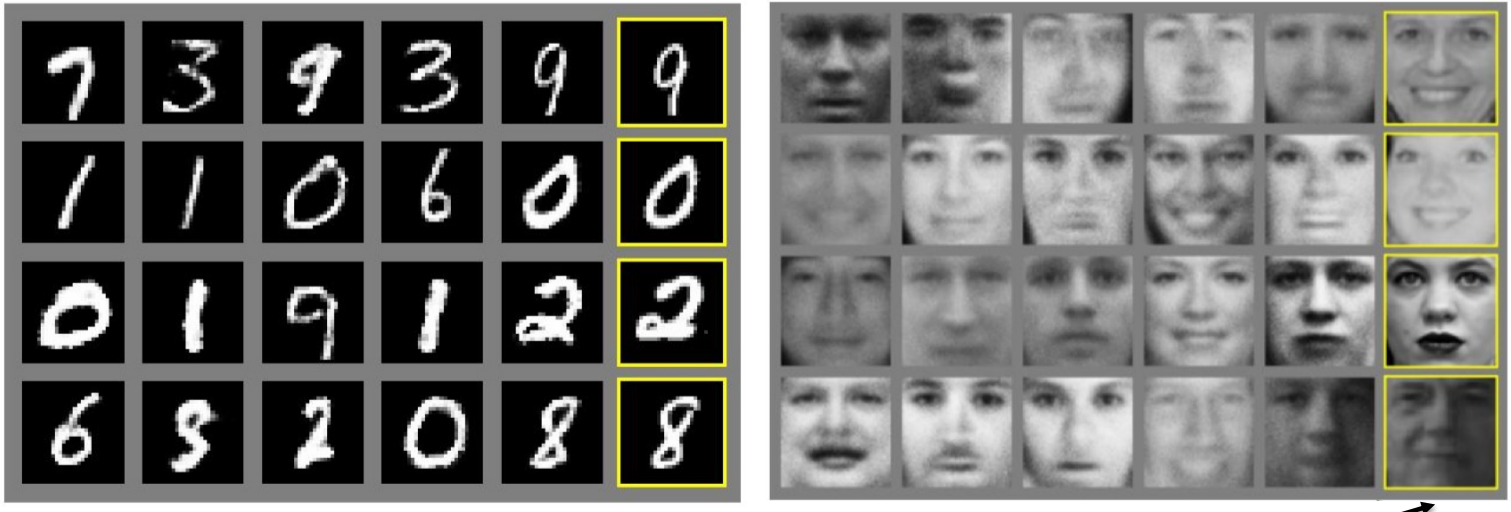
When the data distribution is a 1-D Gaussian distribution



http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/

Generative Adversarial Networks

- Generated samples



Nearest neighbor from training set

Image credit to [Ian Goodfellow et al., Generative Adversarial Nets, NIPS 2016]

Generative Adversarial Networks

- Generated samples (CIFAR-10)

Nearest neighbor from training set

Image credit to [Ian Goodfellow et al., Generative Adversarial Nets, NIPS 2016]

Generative Adversarial Networks

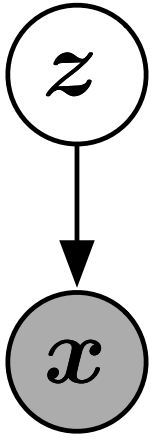
- $\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}^{(G)})$

  ↓ random noise

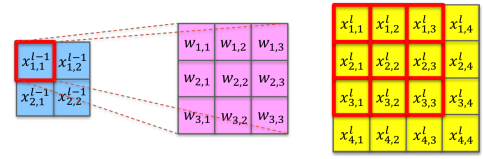
- It is only required that, G is differentiable ↓ end-to-end

- So, having training data from real data distribution p_r what we want is a generative model that can draw samples from generator's distribution p_g, where $p_r \approx p_g$

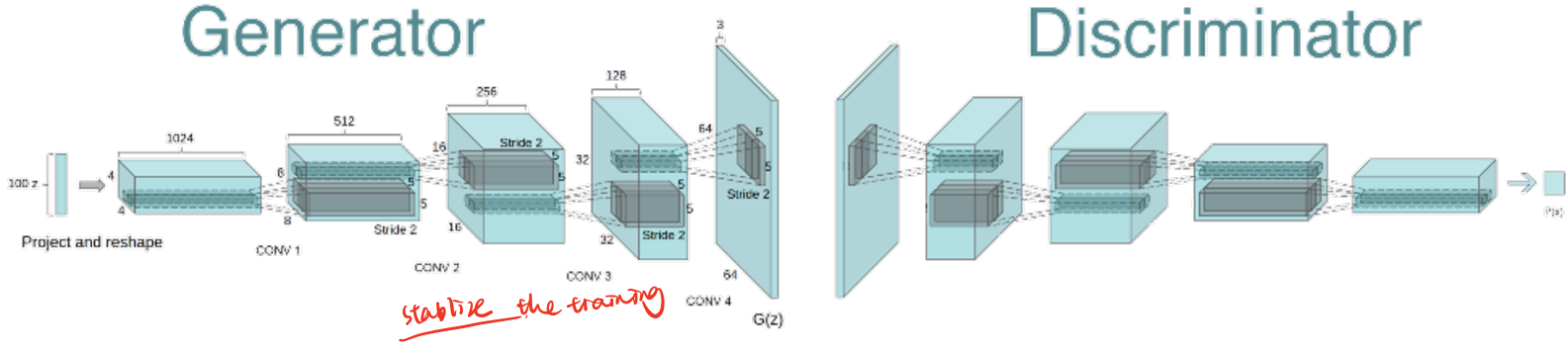
- Don't need to write a formula for p_g just learn to draw sample directly.



Deep Convolutional GANs (DCGANs)



Transposed Convolution



- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

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

Deep Convolutional GANs (DCGANs)

Samples from the model look much better!

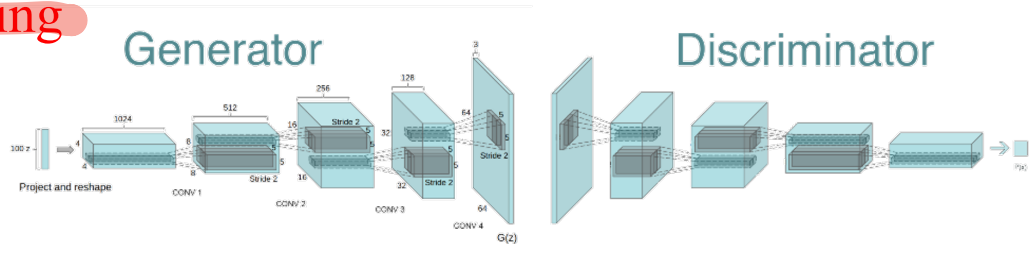
Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Wasserstein GAN

Difficulty of Generative Adversarial Networks

- The gradient vanishing

- The full objective



$$J^{(D)} = -E_{x\sim P_r}[\log D(x)] - E_{x\sim P_g}[\log(1 - D(x))]$$

$$J^{(G)} = E_{x\sim P_g}[\log(1 - D(x))]$$

- At the very early phase of training, D is very easy to be confident in detecting G, so D will output almost always 0

In GAN, better discriminator leads to worse vanishing gradient in its generator!

0 → gradient ≈ 0 → worse for generator

Proof Sketch: [Martin Arjovsky et al., Wasserstein GAN, 2017]

- Minimizing generator yields minimizing the JS divergence when the discriminator is optimal:

$$J^{(D)} = -E_{x\sim P_r}[\log D(x)] - E_{x\sim P_g}[\log(1 - D(x))]$$

- The optimal ***D*** for any P_r and P_g is always:

$$-P_r(x)\log D(x) - P_g(x)\log[1 - D(x)]$$

$$-\frac{P_r(x)}{D(x)} + \frac{P_g(x)}{1 - D(x)} = 0$$

$$D^*(x) = \frac{P_r(x)}{P_r(x) + P_g(x)}$$

- The generative loss (by adding a term independent of P_g) is:

$$J^{(G)} = E_{x\sim P_g}[\log(1 - D(x))]$$

Plug D^* into $J^{(G)}$:

Jensen–Shannon divergence

$$J^{(D^*)} = 2JS(P_r||P_g) - 2\log 2$$

So, when ***D*** is ***optimal***, minimizing the loss is equal to minimizing the *JS divergence*.

JS divergence:

$$\mathrm{JS}(p \parallel q) = \frac{1}{2}\mathrm{KL}(p \parallel m) + \frac{1}{2}\mathrm{KL}(q \parallel m)$$

where

average distribution of $p(x)$ and $q(x)$.

$$m(x) = \frac{1}{2}\left(p(x) + q(x)\right)$$

$$\mathrm{KL}(p \parallel q) = -\sum_{x} p(x) \log\left(\frac{q(x)}{p(x)}\right)$$

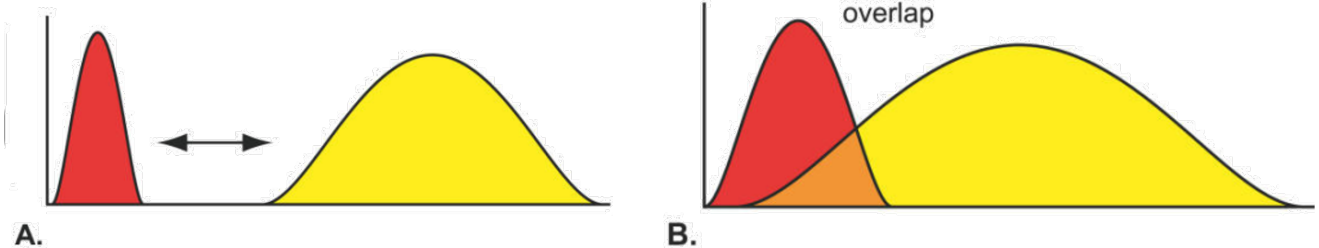
Proof Sketch:

constant
↓
no update.

$$J^{(D^*)} = 2JS(P_r||P_g) - 2\log 2$$

- If the supports (underlying low-dimension manifolds) of P_r and P_g (almost) have no overlap, then $JS(P_r||P_g) = \log 2$, and thus the gradient of $J^{(G)}$ w.r.t. P_g vanishes.

- The probability that the support of P_r and P_g have almost zero overlap is 1.



[Martin Arjovsky et al., Wasserstein GAN, 2017]

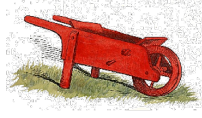
Preliminaries: distance measures for distributions

- JS

$$JS(P||Q) = \frac{1}{2}KL(P||\frac{P+Q}{2}) + \frac{1}{2}KL(Q||\frac{P+Q}{2})$$

- Wasserstein

$$W(P||Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$



the set of all joint γ distributions whose marginals are P and Q, respectively.

γ indicates a plan to transport "mass" from x to y, when performing P into Q.

The Wasserstein (or Earth-Mover) distance is then the "cost" of the **optimal** transport plan

Image credit to: sbl.inria.fr

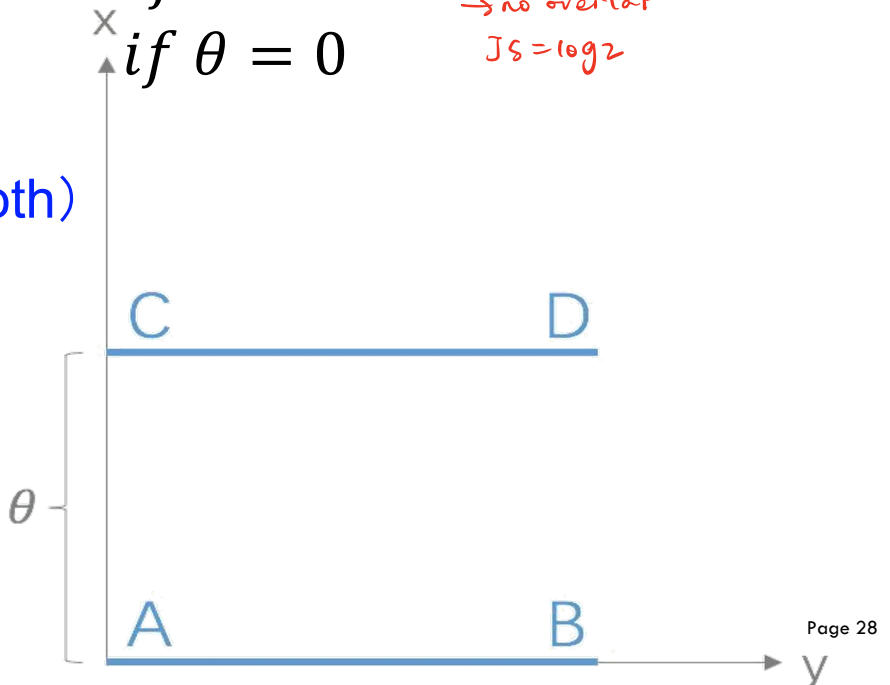
Examples

$$KL(P_1||P_2) = \begin{cases} +\infty & if\ \theta \neq 0 \\ 0 & if\ \theta = 0 \end{cases}$$

$$JS(P_1||P_2) = \begin{cases} \log 2 & if\ \theta \neq 0 \\ 0 & if\ \theta = 0 \end{cases}$$

if there is a significant gap
→ no overlap
JS = log2

$$W(P_1, P_2) = |\theta| \text{ (Smooth)}$$



Wasserstein GANs

- The Earth-Mover (EM) distance or Wasserstein-1:

$$W(P||Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$

have nicer properties when optimized than JS

- However, the infimum is highly intractable.

- Wasserstein distance has a duality form

$$W(P_r, P_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_g}[f(x)]$$

$$= \frac{1}{K} \sup_{||f||_L \leq K} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_g}[f(x)]$$

where supremum is over all the K-Lipschitz functions

$$\|f(x) - f(y)\| \leqslant L\|x - y\|$$

Wasserstein GANs

- Wasserstein distance has a duality form

$$W(P_r, P_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_g}[f(x)]$$
$$= \frac{1}{K} \sup_{||f||_L \leq K} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_g}[f(x)]$$

where supremum is over all the K-Lipschitz functions

- Consider a w-parameterized family of functions $\{f_w\}_{w \in W}$ that are all K -Lipschitz

$$W(P_r, P_g) = \max_{w \in W} \mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{x \sim P_g}[f_w(x)]$$

For example, $W = [-c, c]^l$. To satisfy this requirement, WGAN enforces the weights of D lie within a compact space [-c, c] by applying weight clipping

Wasserstein GANs

- The loss for **discriminator/critic** f_w

$$\mathbb{E}_{x\sim P_r}[f_w(x)] - \mathbb{E}_{x\sim P_g}[f_w(x)]$$

- f_w's goal: maximize Wasserstein distance between real data distribution and generative distribution

- The loss for **generator**

$$-\mathbb{E}_{x\sim P_g}[f_w(x)] = -\mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z))]$$

- g_θ's goal: minimize Wasserstein distance between real data distribution and generative distribution

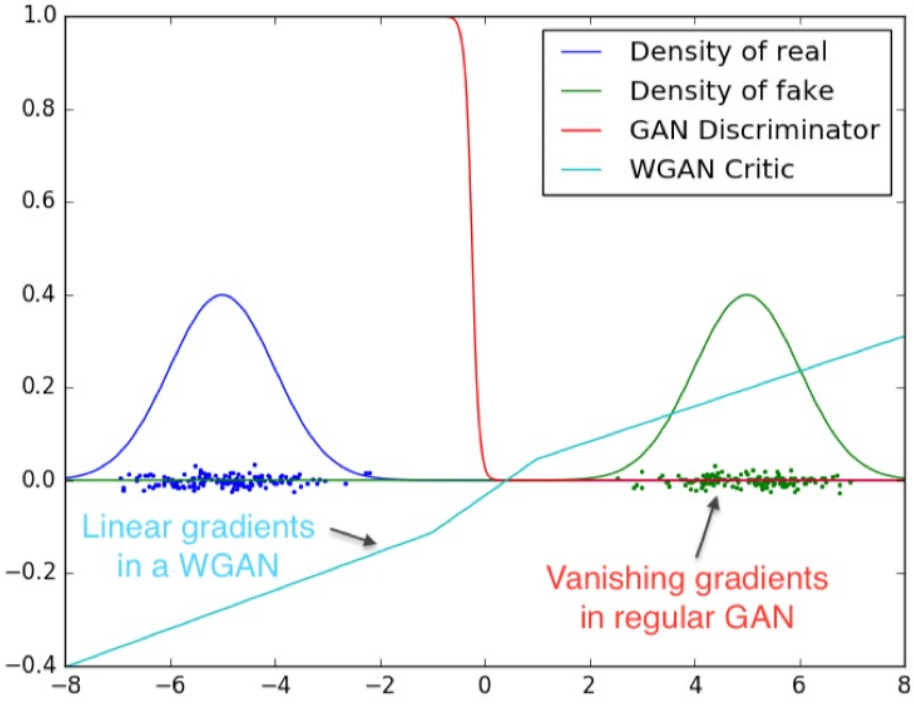
Algorithm

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size.
n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$ → compute gradient of f_w
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ → gradient ascent on f_w
    $w \leftarrow \text{clip}(w, -c, c)$ → weight clipping to satisfy that functions $\{f_w\}_{w \in W}$ are all K - Lipschitz
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$ → compute gradient of g_θ
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$ → gradient decent on g_θ
end while

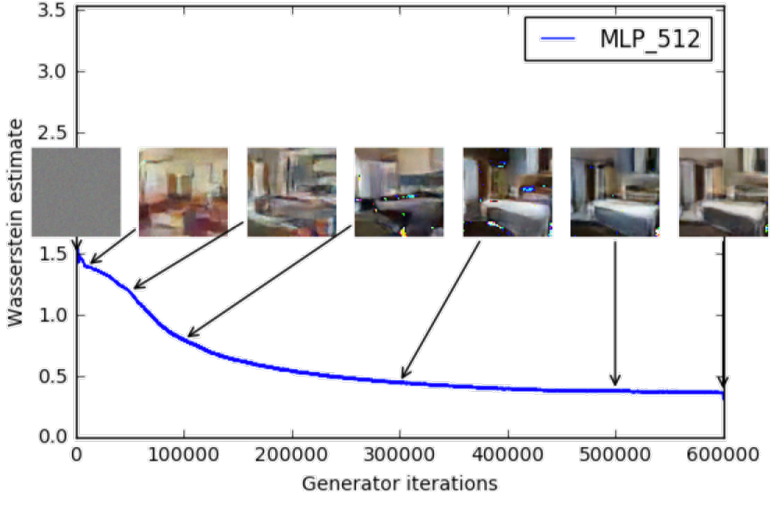
Algorithm



Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. WGAN critic provides very clean gradients on all parts of the space.

Image credit to [Martin Arjovsky et al., Wasserstein GAN, 2017]

Meaningful loss metric



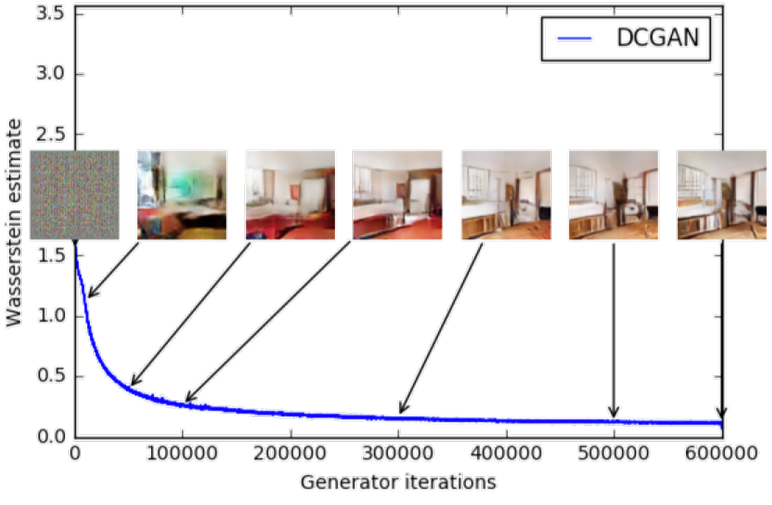
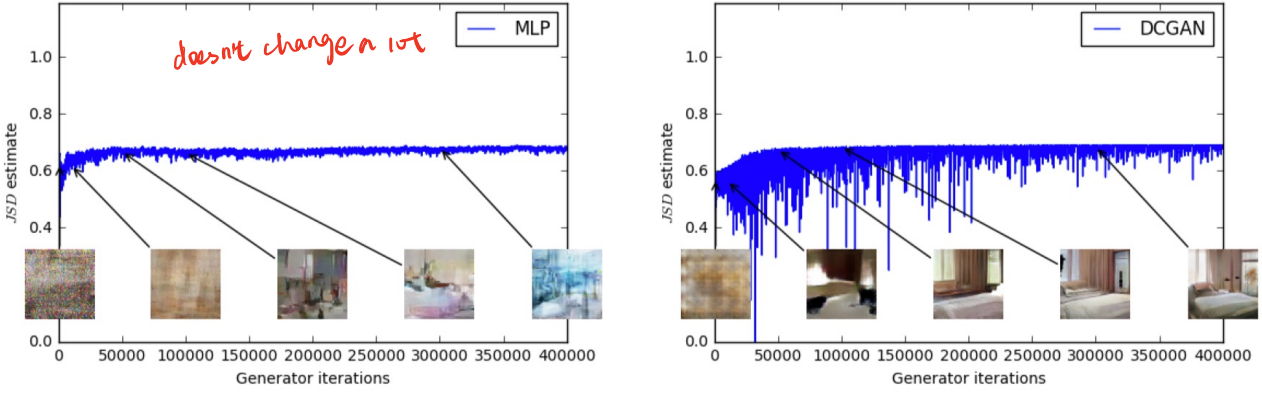


Image credit to [Martin Arjovsky et al., Wasserstein GAN, 2017]

Meaningful loss metric

- Vanilla GANs



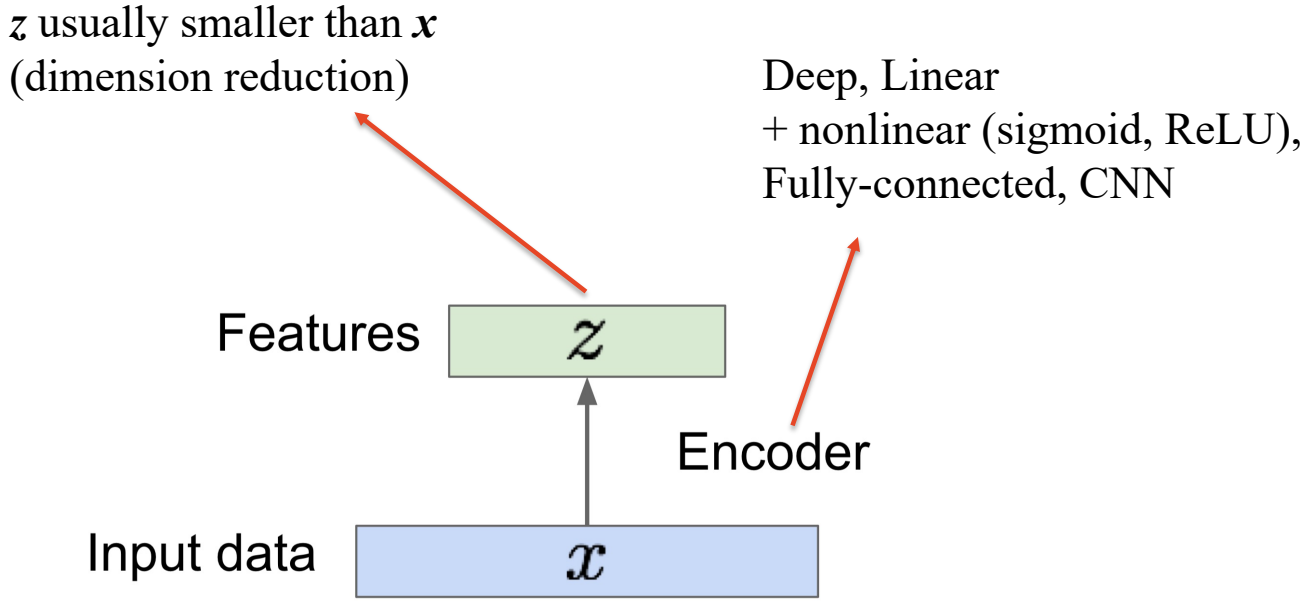
JS estimates for an MLP generator (upper left) and a DCGAN generator (upper right) trained with the standard GAN procedure. Both had a DCGAN discriminator. Both curves have increasing error. Samples get better for the DCGAN but the JS estimate increases or stays constant, pointing towards no significant correlation between sample quality and loss.

Image credit to [Martin Arjovsky et al., Wasserstein GAN, 2017]

Variational Auto-Encoder

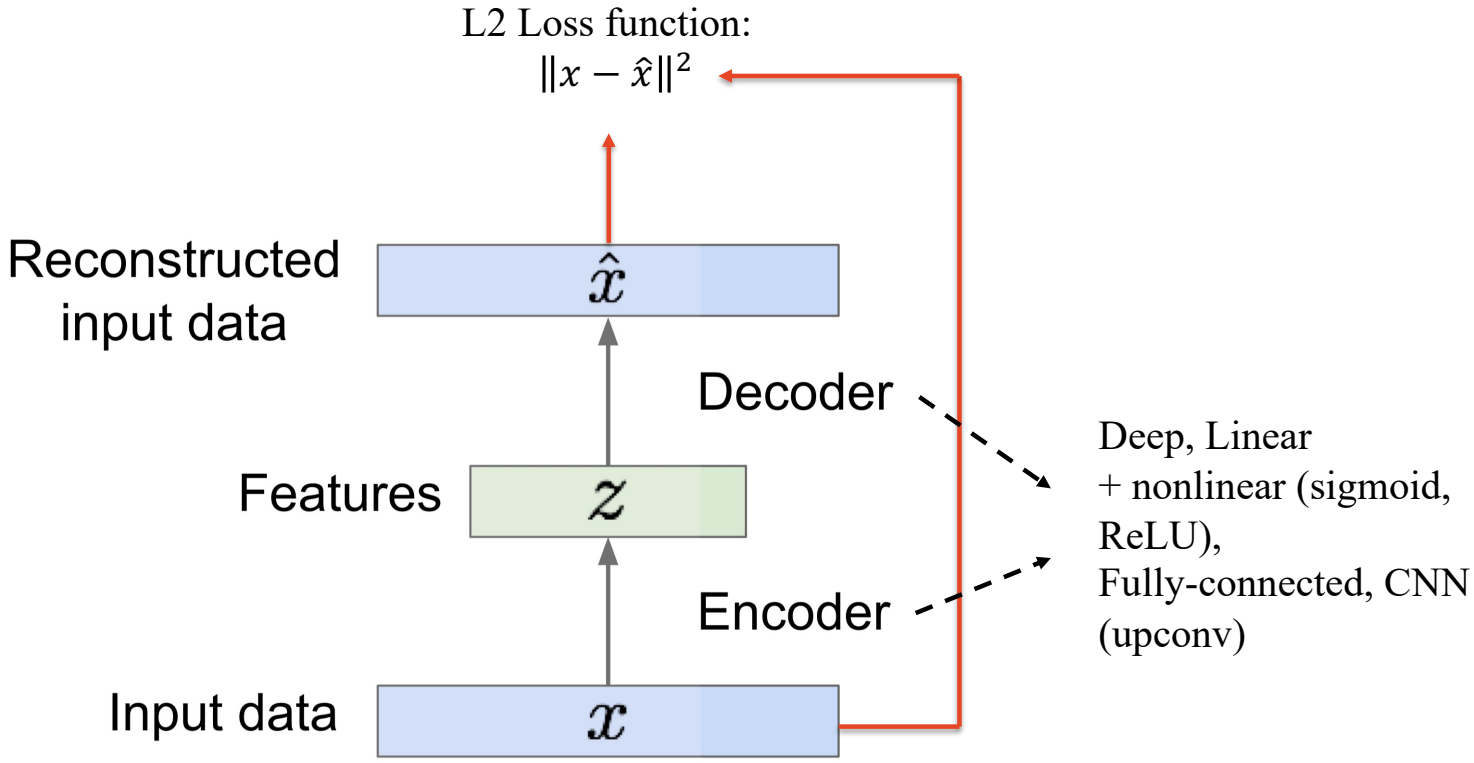
Recap: Autoencoder

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

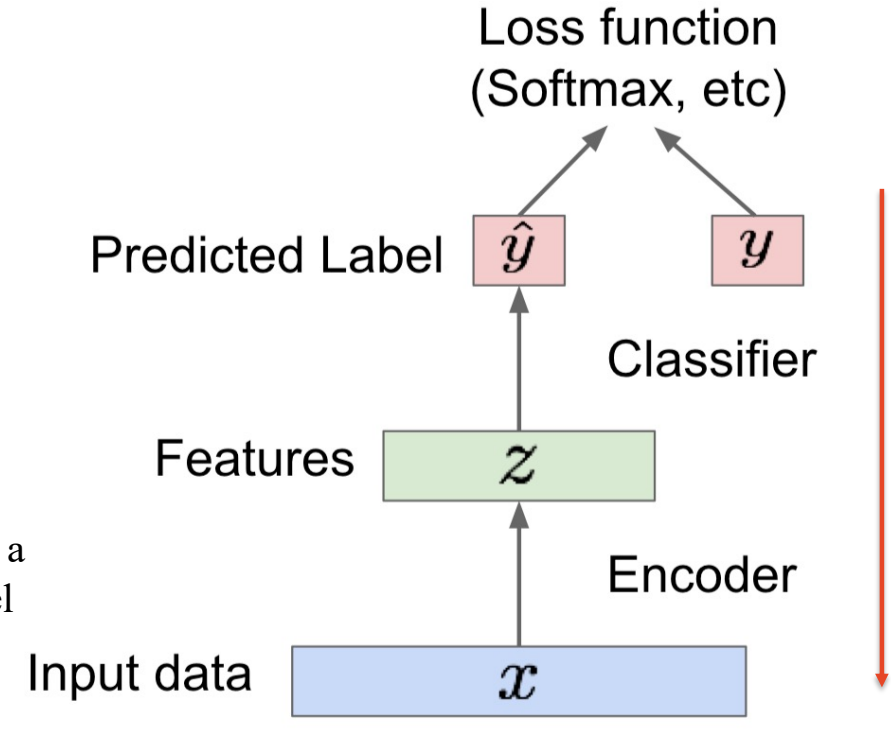


Recap: Autoencoder

- Train such that features can be used to reconstruct original data
  "Autoencoding" – encoding itself



Recap: Autoencoder



Throw away decoder

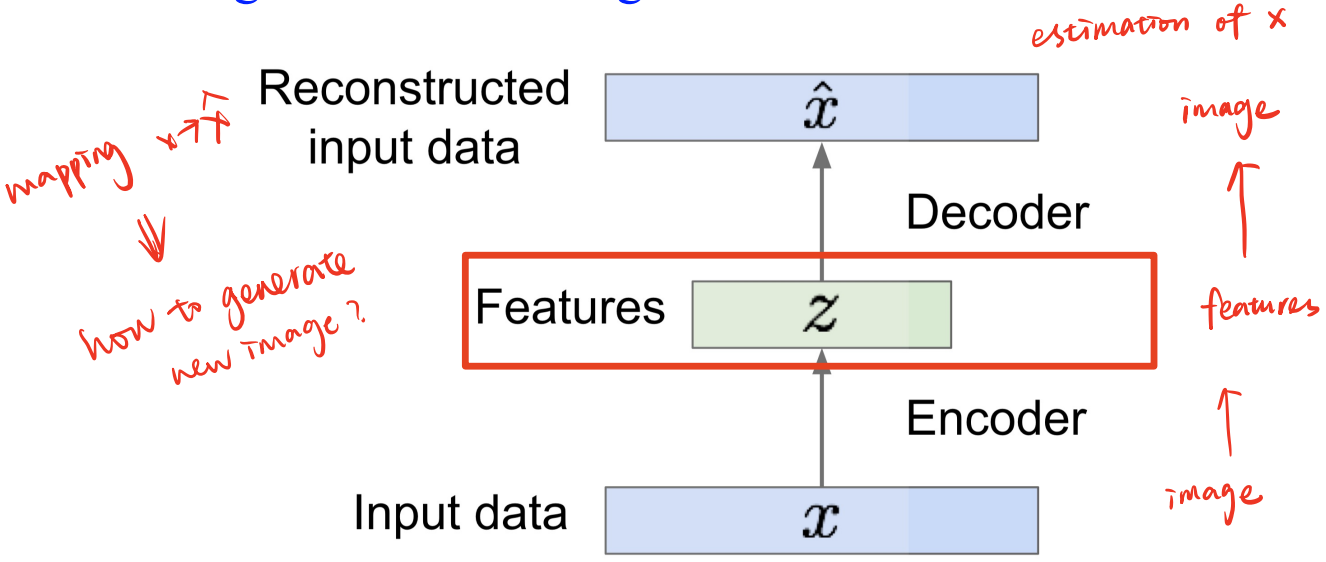
Encoder can be used to initialize a supervised model

Fine-tune encoder jointly with classifier

Recap: Autoencoder

Autoencoders can reconstruct data, and can learn features to initialize a supervised model. Features capture factors of variation in training data.

Can we generate new images from an autoencoder?

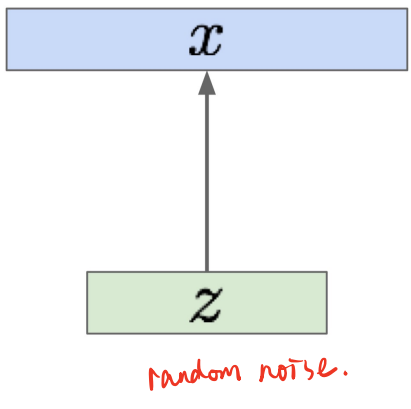


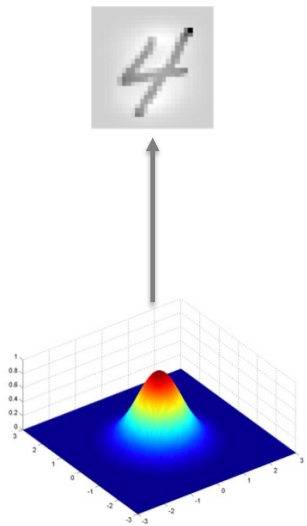
Variational Autoencoders

- Probabilistic spin on autoencoders - will let us sample from the model to generate data!
- Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation z

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

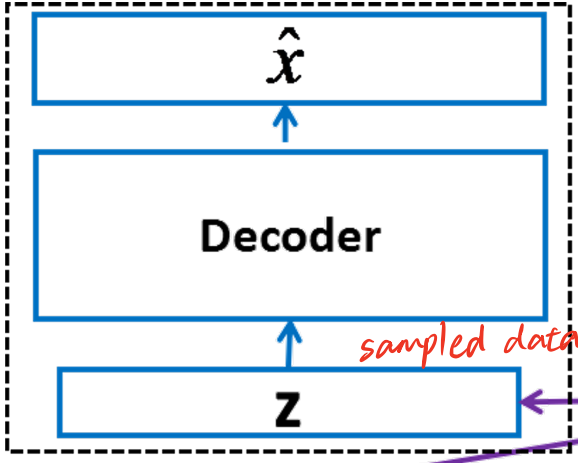
Sample from true prior $p_{\theta*}(z)$





Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose.

Variational Autoencoders



4 Decoder

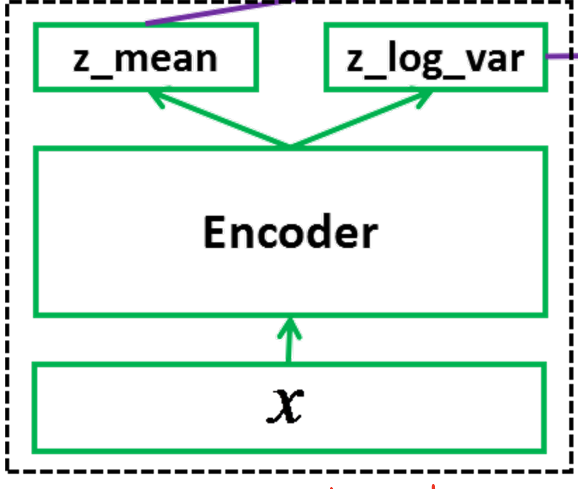
standard autoencoder → extract feature
VAE → distribution z mean var

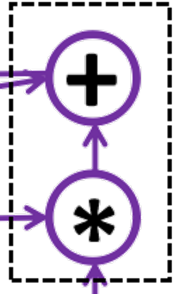
Reparemerization

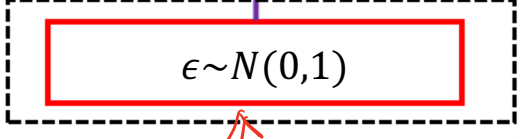
sampled data

3

$z = \sigma \cdot \epsilon + \mu$







σ, μ is irrelevant w.r.to μ (sampling process).

2

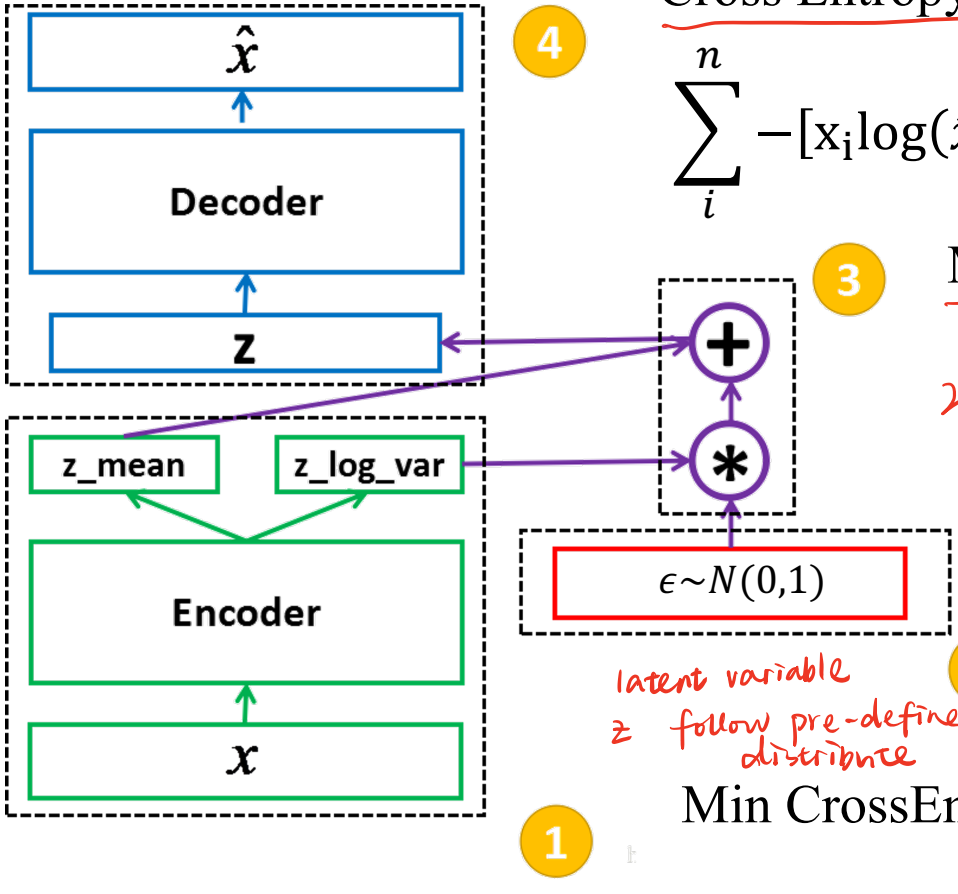
how to back propogate sampling?

Sampling

real world image

1 Encoder

Variational Autoencoders



Cross Entropy

$$\sum_{i}^{n} -[x_i \log(\hat{x}_i) + (1 - x_i)\log(1 - \hat{x}_i)]$$

MSE $\sum_{i}^{n} (x_i - \hat{x}_i)^2$

2 distributions. (pre-defined & estimated).

KL Divergence

$$-0.5(1 + \log \sigma^2 - \mu^2 - \sigma^2)$$

latent variable z follow pre-defined distribute

Min CrossEntropy + KLDivergence

Variational Autoencoders

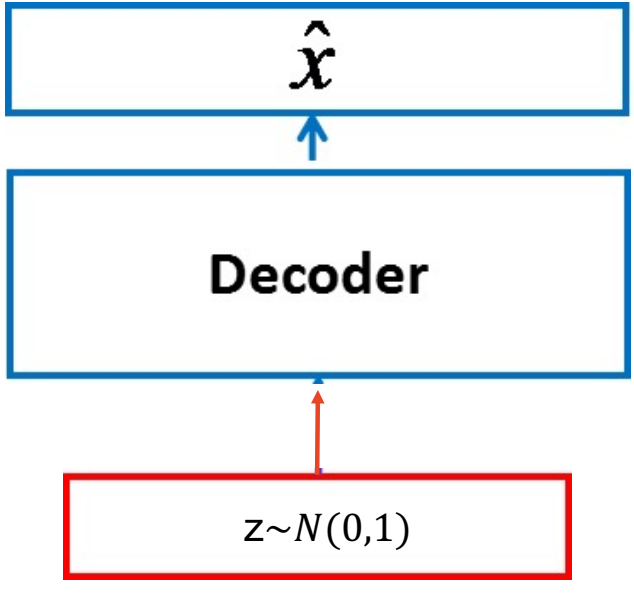
learn distribution

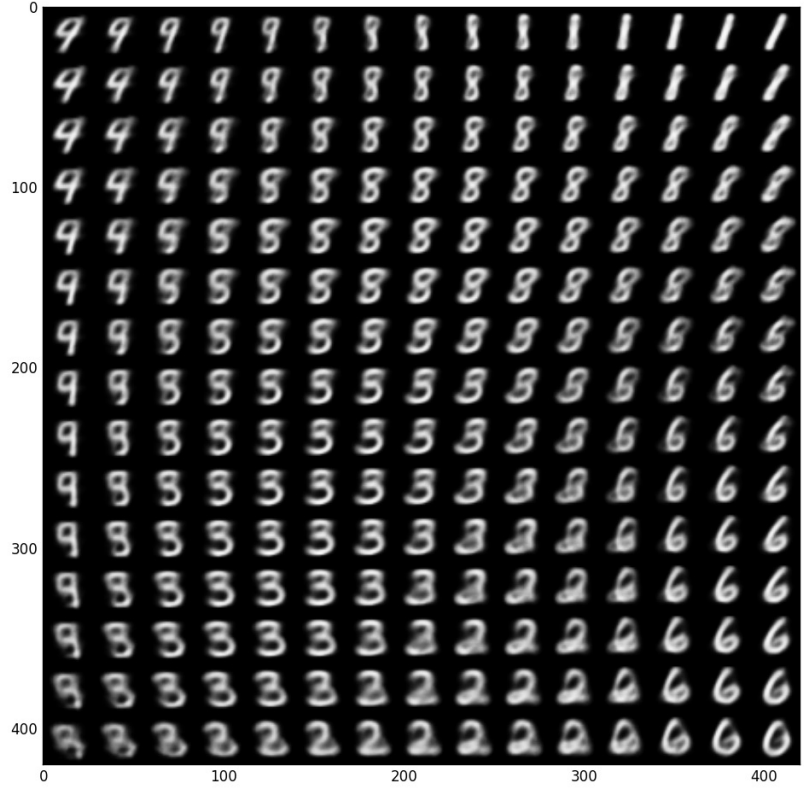
map distribution to image

Generating Data:

Use decoder network.

Sample z from prior.





Variational Autoencoders

Generating Data:

32x32 CIFAR-10

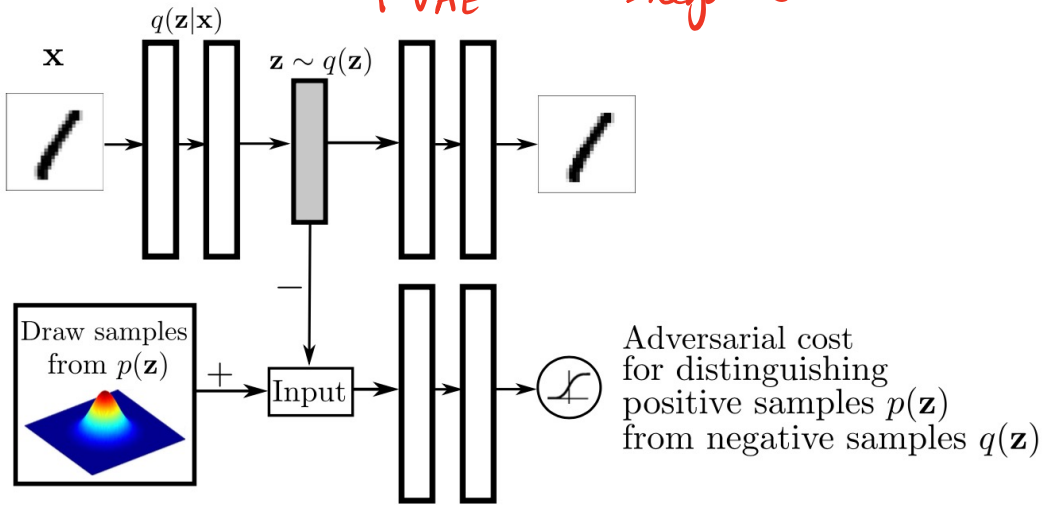
Labeled Faces in the Wild

Image credit to [(L) Drick Kingma et al. 2016; (R) Anders Larsen et al. 2017].

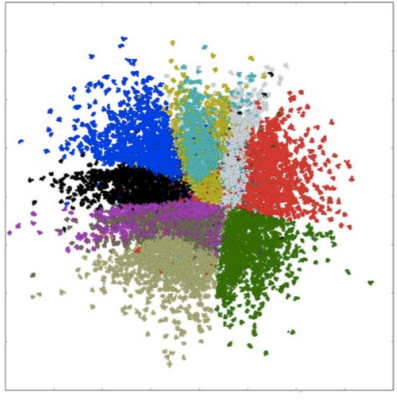
VAE + GAN

sampled z from pre-defined → positive
{ VAE → negative

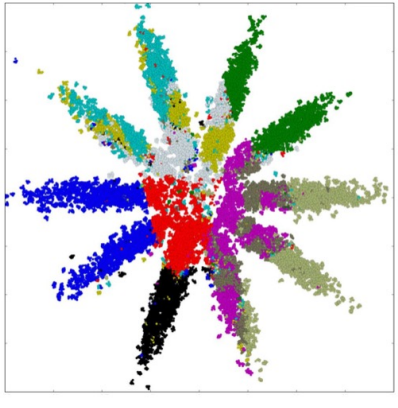
$$q(\mathbf{z}) = \int_{\mathbf{x}} q(\mathbf{z}|\mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x}$$



Code Space of MNIST:



Gaussian Prior



Mixture of Gaussians

Makhzani et al., 2015