

# CS231n

## Lecture 13. Generative Models

Tobig's 14기 서아라

# Supervised Learning

## Supervised Learning

**Data:**  $(x, y)$

$x$  is data,  $y$  is label

**Goal:** Learn a *function* to map  $x \rightarrow y$

**Examples:** Classification, regression, object detection, semantic segmentation, image captioning, etc.



→ Cat

Classification



DOG, DOG, CAT

Object Detection



GRASS, CAT,  
TREE, SKY

Semantic Segmentation



A cat sitting on a suitcase on the floor

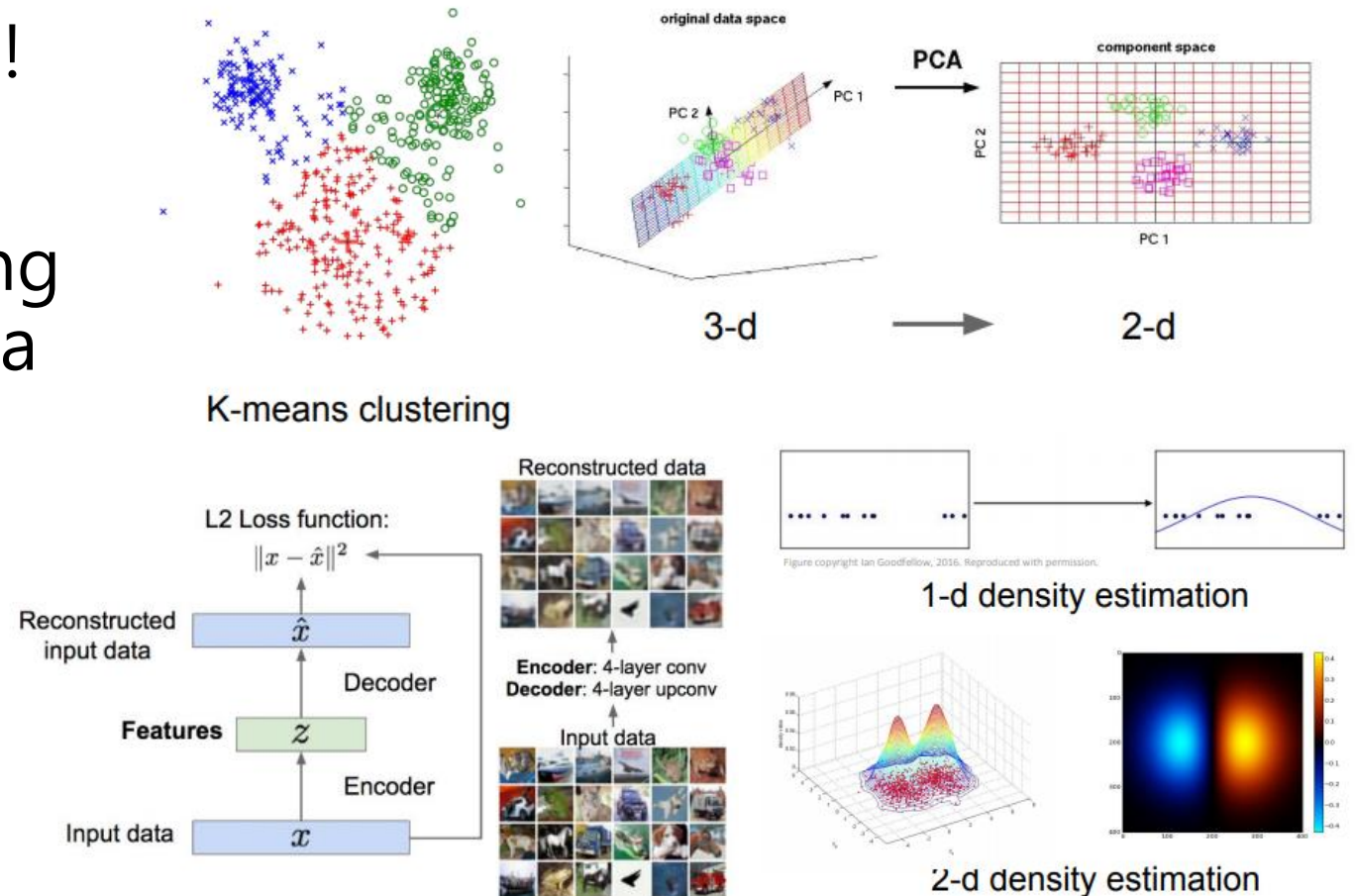
Image captioning

# Unsupervised Learning?

**Data:** x Just data, no labels!

**Goal:** Learn some underlying hidden structure of the data

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



# Generative Models

- Realistic samples for artwork, super-resolution, colorization, etc.



- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representations that can be useful as general features



# Taxonomy of Generative Models

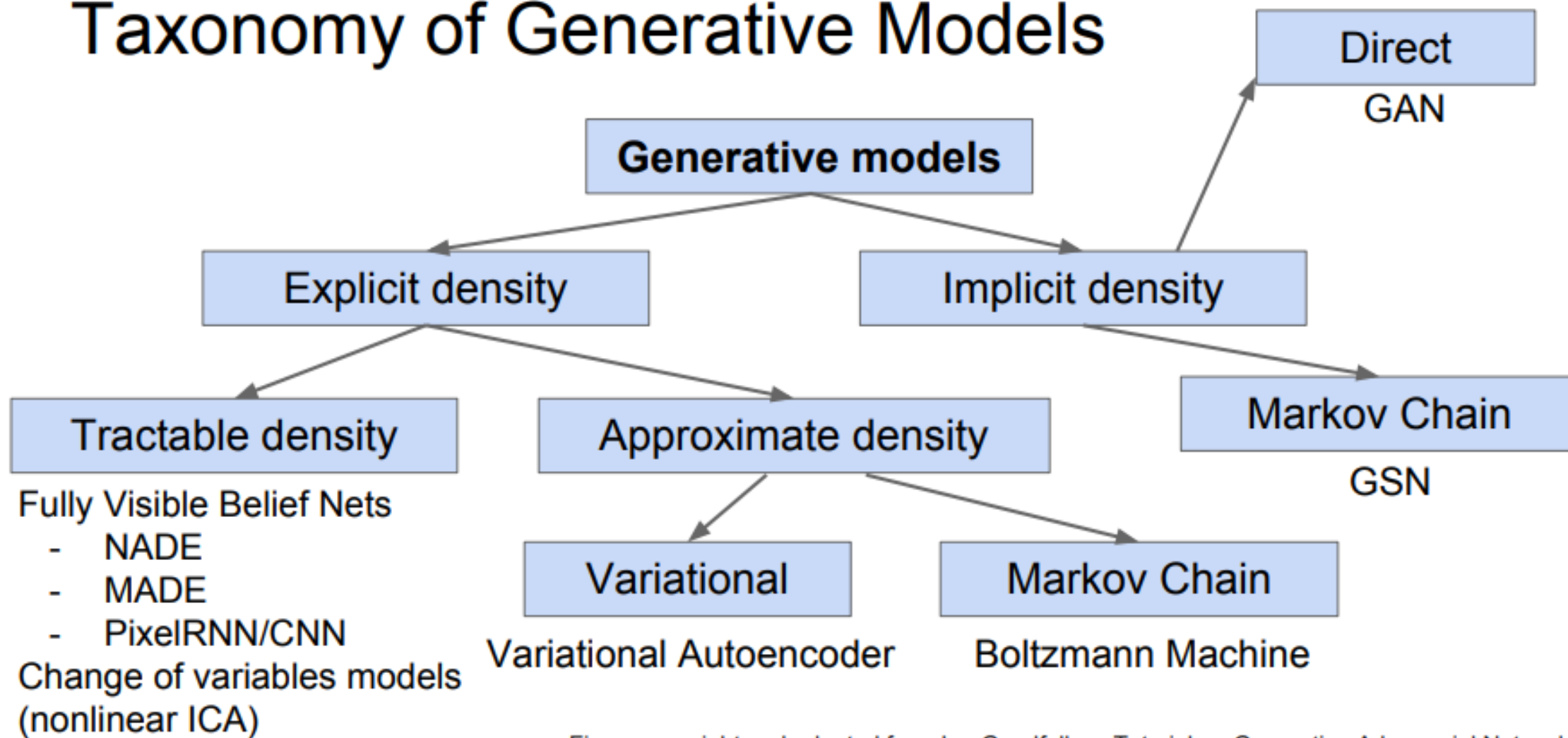


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# PixelRNN

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

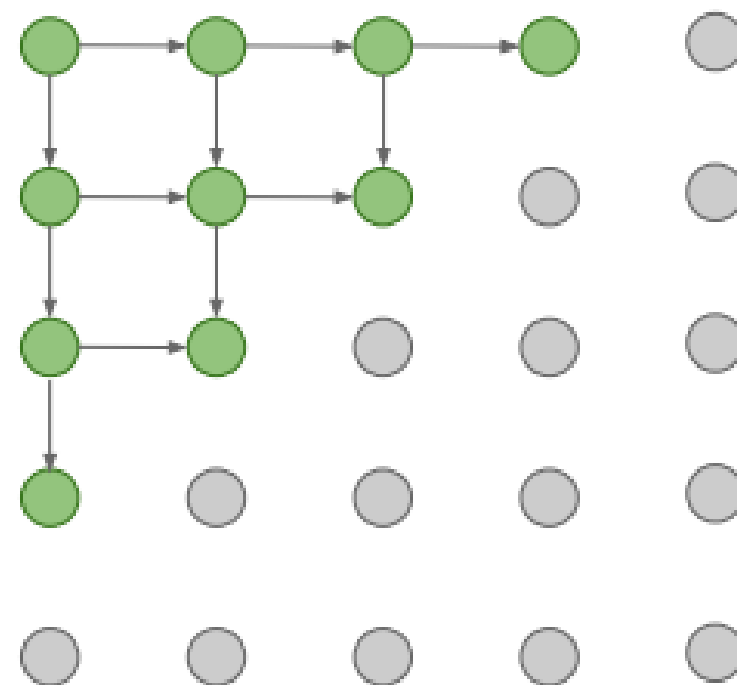
Likelihood of  
image x

Probability of i'th pixel value  
given all previous pixels

## Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!



# PixelCNN *[van der Oord et al. 2016]*

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

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

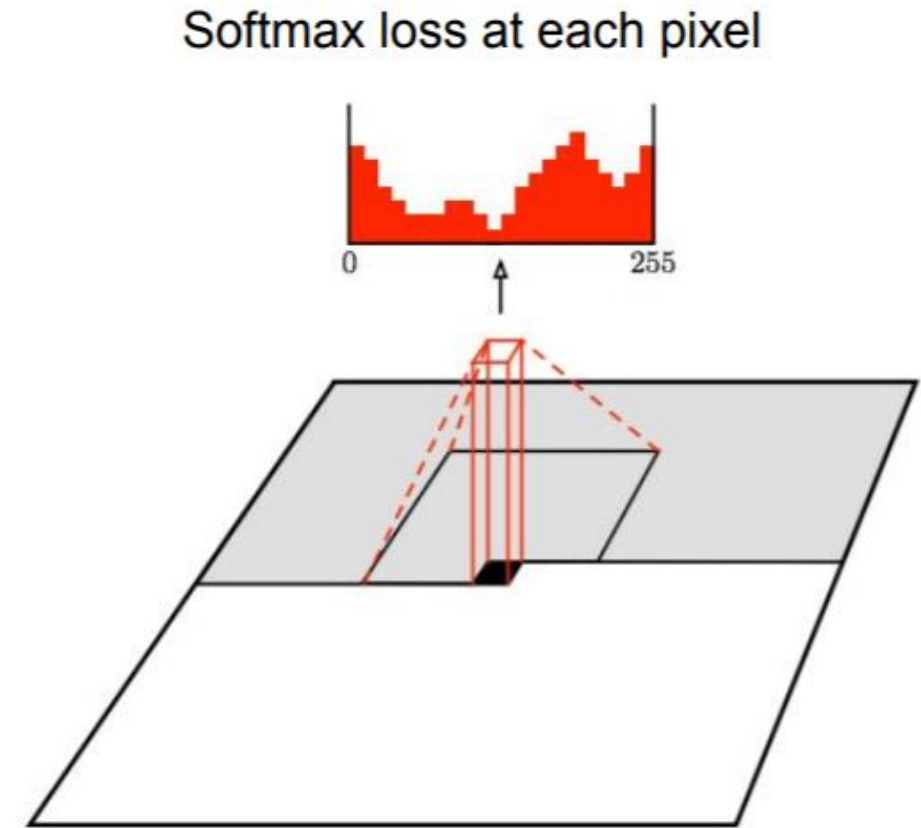


Figure copyright van der Oord et al., 2016. Reproduced with permission.

# PixelRNN/CNN

## Pros:

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

## Con:

- Sequential generation => slow

## Improving PixelCNN performance

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc...

## See

- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)



# Variational Autoencoders (VAE)

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

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

VAEs define intractable density function with latent  $\mathbf{z}$ :

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

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

# Some background first: Autoencoders

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

$\mathbf{z}$  usually smaller than  $\mathbf{x}$   
(dimensionality reduction)

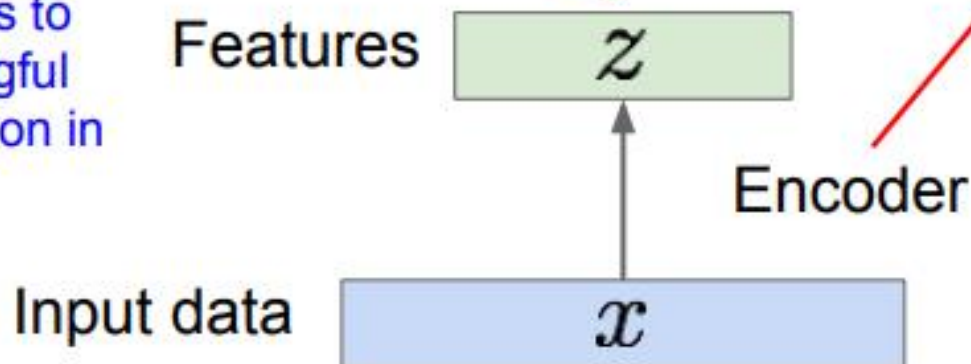
Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data

**Originally:** Linear + nonlinearity (sigmoid)

**Later:** Deep, fully-connected

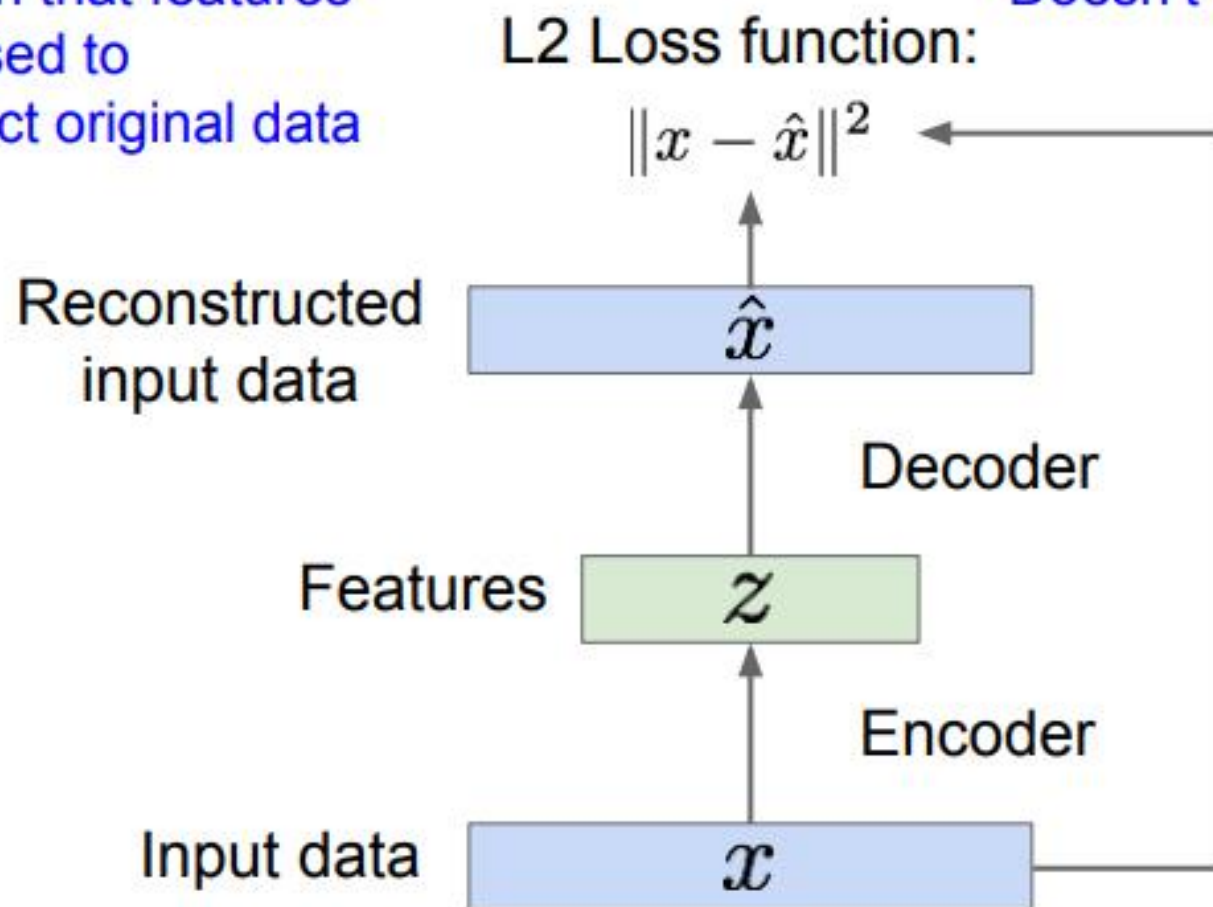
**Later:** ReLU CNN



# Some background first: Autoencoders

Train such that features  
can be used to  
reconstruct original data

Doesn't use labels!



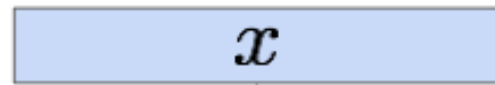
# Variational Autoencoders (VAE)

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  $\mathbf{z}$

Sample from  
true conditional

$$p_{\theta^*}(x \mid z^{(i)})$$



Sample from  
true prior

$$p_{\theta^*}(z)$$

**Intuition** (remember from autoencoders!):  
 $\mathbf{x}$  is an image,  $\mathbf{z}$  is latent factors used to  
generate  $\mathbf{x}$ : attributes, orientation, etc.

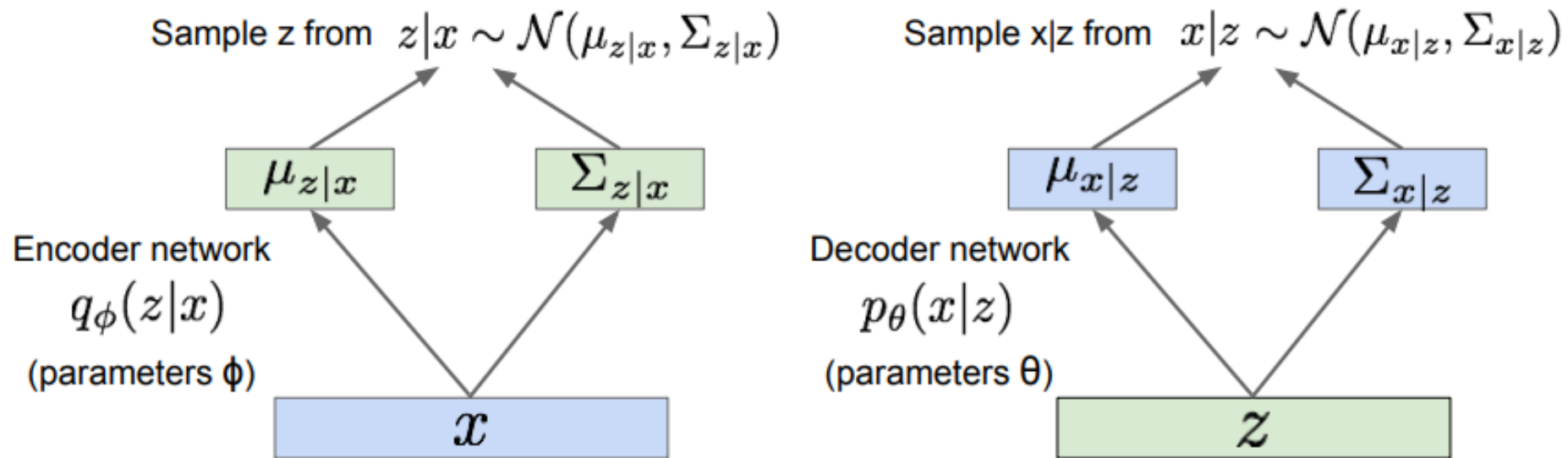
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014



# Variational Autoencoders (VAE)

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

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



Encoder and decoder networks also called  
“recognition”/“inference” and “generation” networks

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



# Variational Autoencoders

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

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \frac{q_{\phi}(z | x^{(i)})}{q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))\end{aligned}$$

↑  
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  $\geq 0$ .

# Variational Autoencoders

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

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)})) \text{ Does not depend on } z \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule}) \\ \text{Reconstruct the input data} &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] && (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] && (\text{Logarithms}) \\ &= \underbrace{\mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{> 0} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{> 0}\end{aligned}$$

Make approximate posterior distribution close to prior

$$\log p_{\theta}(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^N \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

# Variational Autoencoders

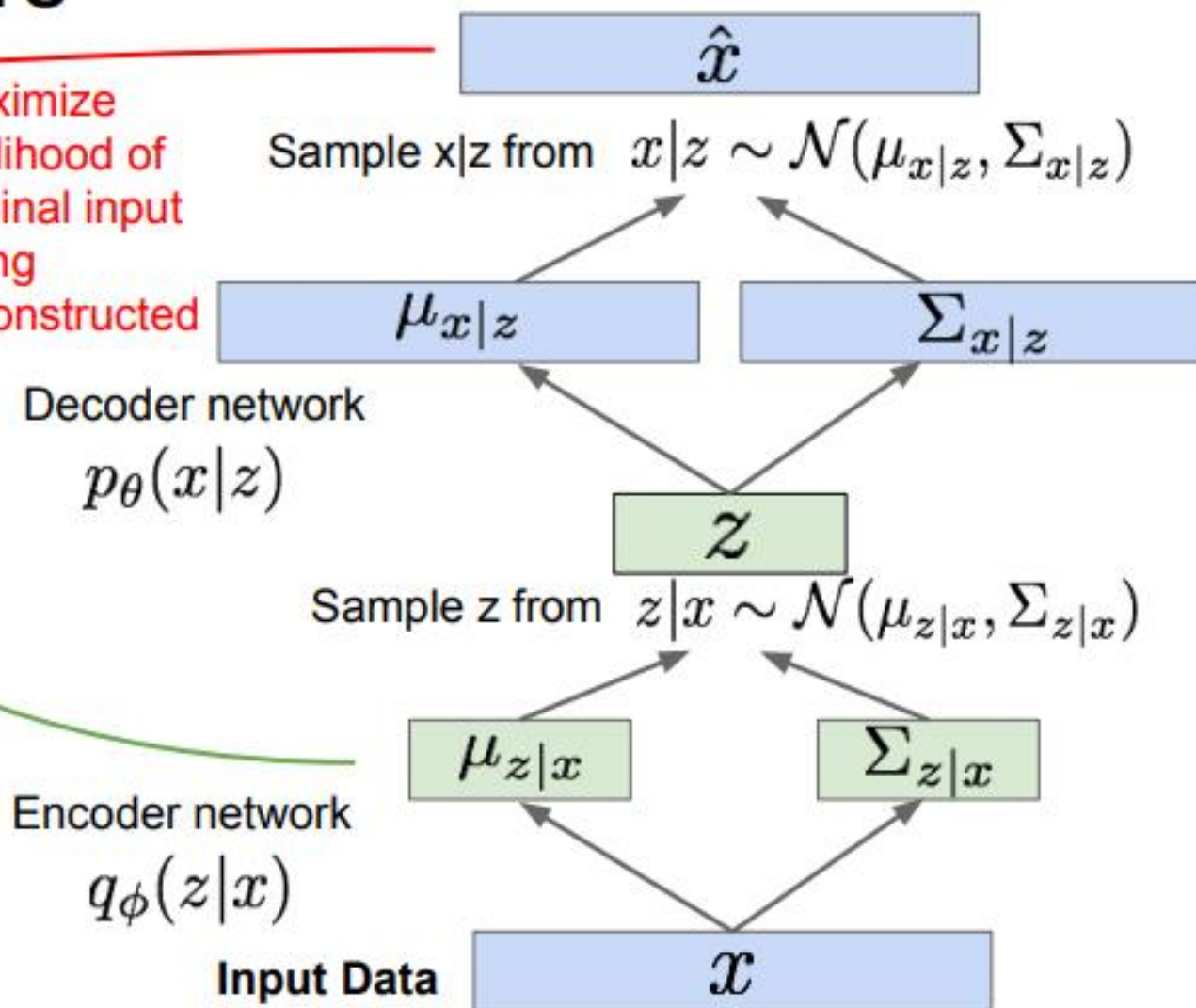
Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed

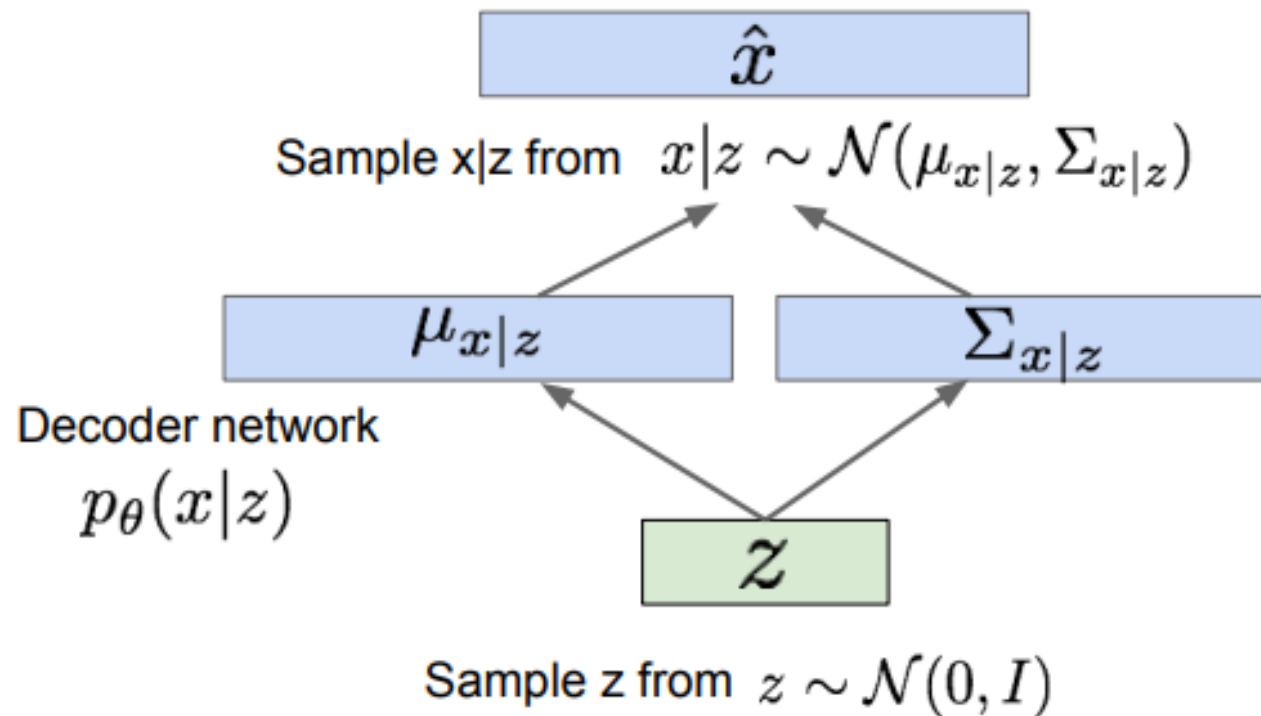
Make approximate posterior distribution close to prior

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

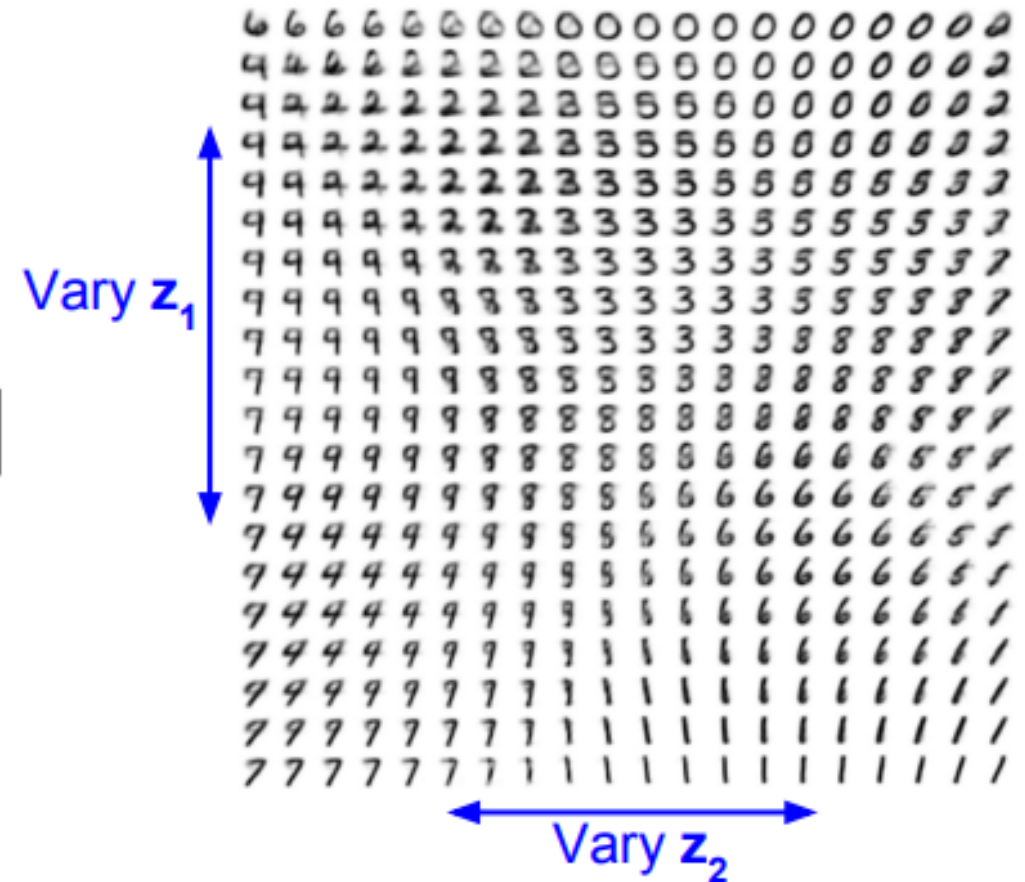


# Variational Autoencoders: Generating Data!

Use decoder network. Now sample  $z$  from prior!



Data manifold for 2-d  $z$



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014



# Variational Autoencoders: Generating Data!

Diagonal prior on  $\mathbf{z}$   
=> independent  
latent variables

Different  
dimensions of  $\mathbf{z}$   
encode  
interpretable factors  
of variation

Also good feature representation that  
can be computed using  $q_\phi(\mathbf{z}|\mathbf{x})$ !

Degree of smile

Vary  $\mathbf{z}_1$



Vary  $\mathbf{z}_2$

Head pose

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014



# Variational Autoencoders (VAE)



32x32 CIFAR-10



Labeled Faces in the Wild

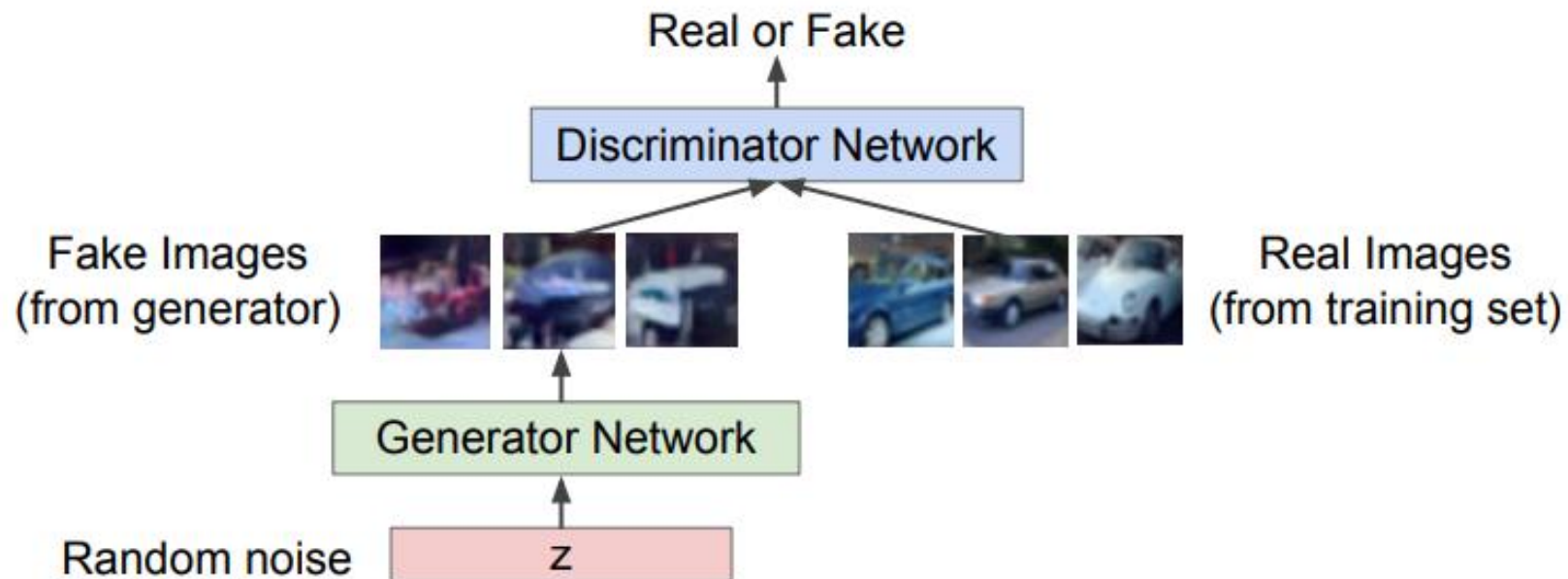
# GAN(Generative Adversarial Networks)

## Training GANs: Two-player game

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

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

**Discriminator network:** try to distinguish between real and fake images



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

# GAN(Generative Adversarial Networks)

Train jointly in **minimax game**

Minimax objective function:

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

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

Alternate between:

1. **Gradient ascent** on discriminator

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

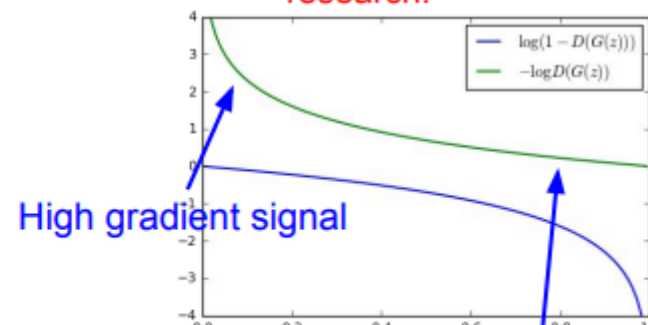
2. **Instead: Gradient ascent** on generator, **different objective**

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

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Same objective of fooling discriminator, but now higher gradient

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.



# GAN(Generative Adversarial Networks)

Putting it together: GAN training algorithm

for number of training iterations **do**  
  for  **$k$  steps** **do**  
    • Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .  
    • Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .  
    • Update the discriminator by ascending its stochastic gradient:  
      
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$
  
  **end for**  
    • Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .  
    • Update the generator by ascending its stochastic gradient (improved objective):  
      
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$
  
  **end for**

Some find  $k=1$  more stable, others use  $k > 1$ , no best rule.

Recent work (e.g. Wasserstein GAN) alleviates this problem, better stability!



# GAN(Generative Adversarial Networks)

Generated samples (CIFAR-10)



Nearest neighbor from training set

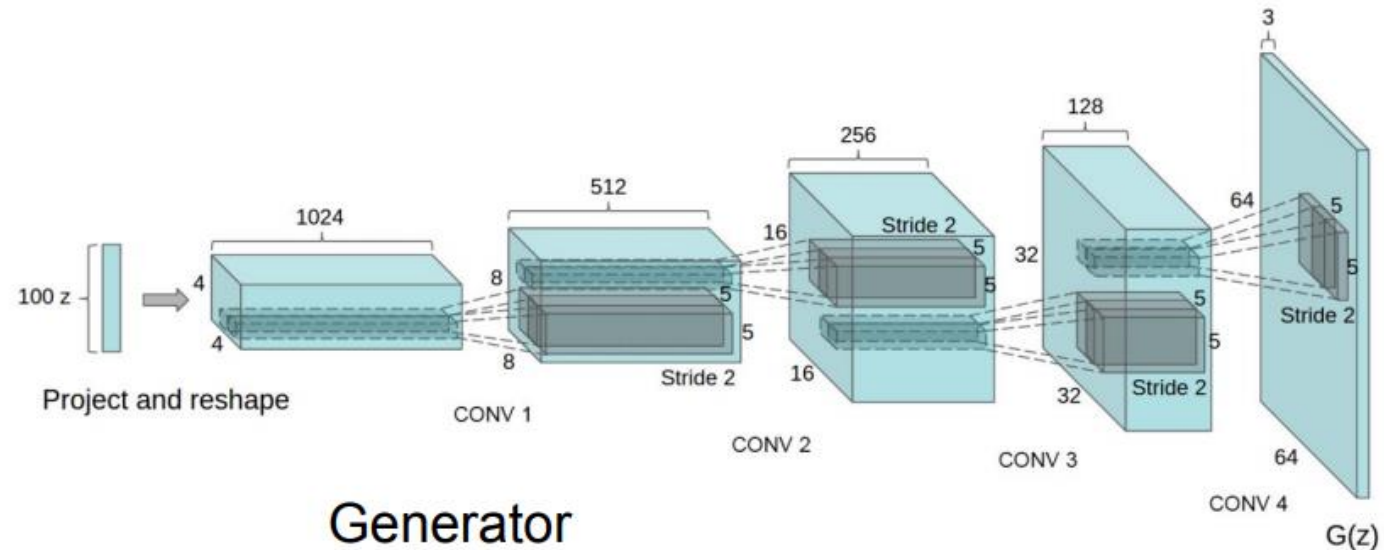
Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.



# GAN(Generative Adversarial Networks)

## Architecture guidelines for stable Deep Convolutional GANs

- 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.



# GAN(Generative Adversarial Networks)

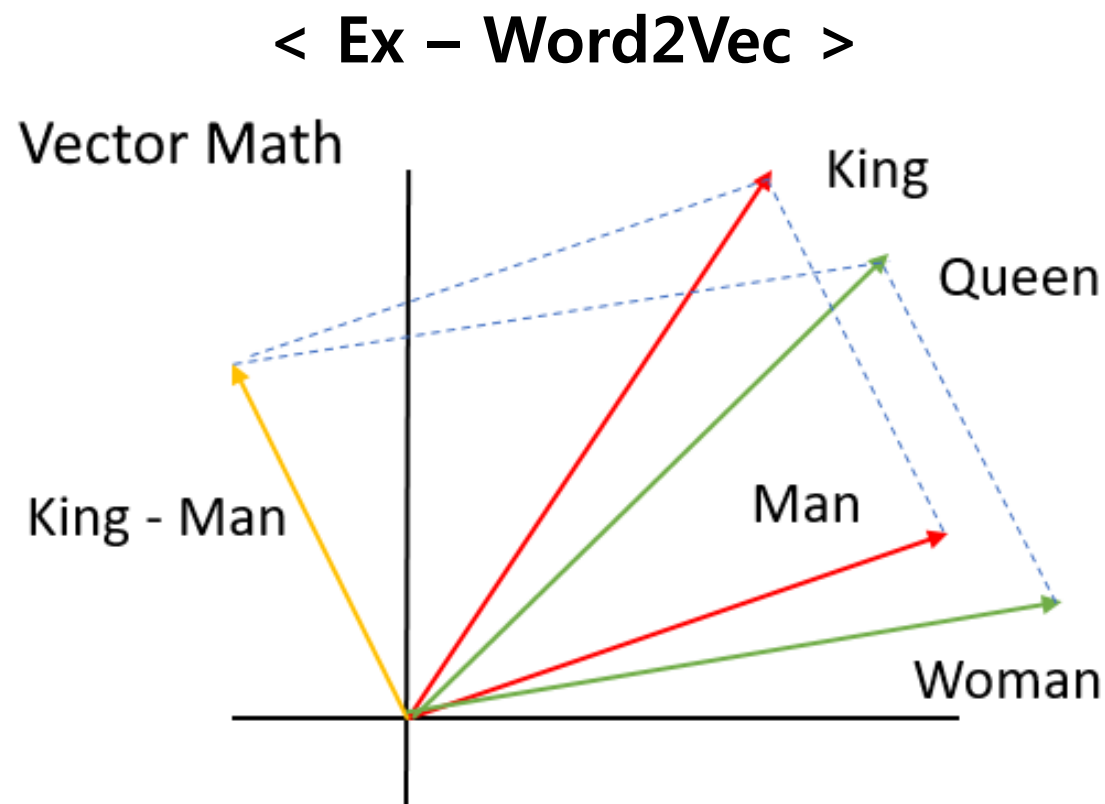
Samples  
from the  
model look  
amazing!

Radford et al,  
ICLR 2016



# GAN(Generative Adversarial Networks)

## Vector Arithmetic



# GAN(Generative Adversarial Networks)

Glasses man



No glasses man

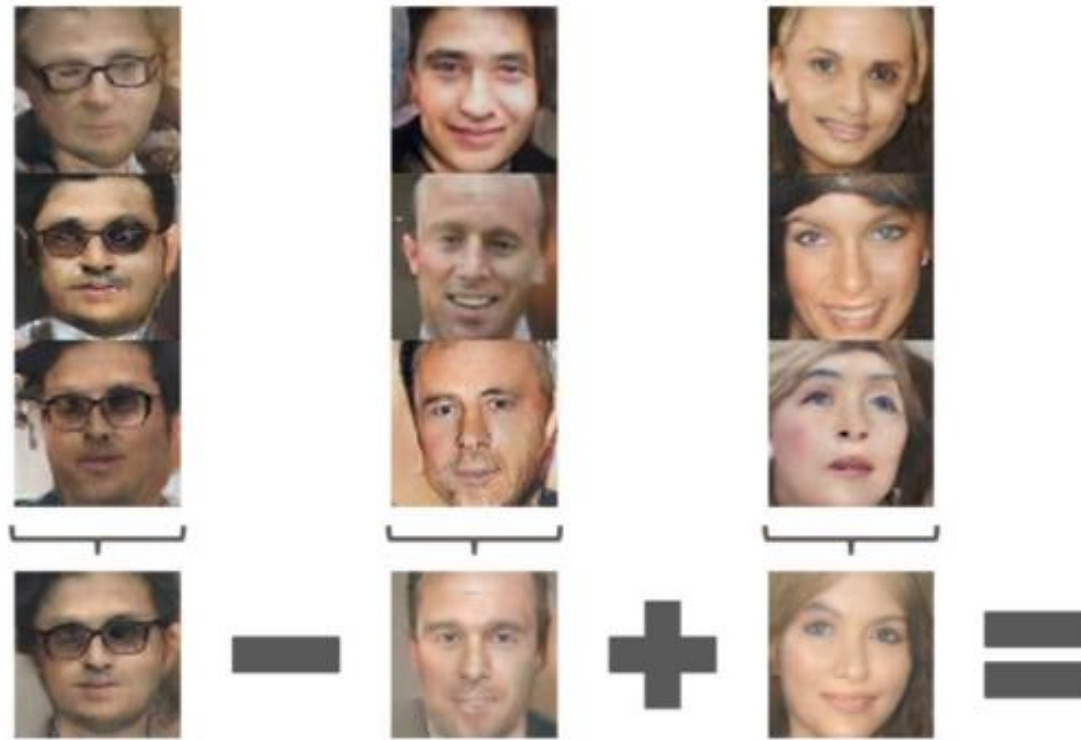


No glasses woman



Radford et al,  
ICLR 2016

Woman with glasses





# Summary

## Generative Models

- PixelRNN and PixelCNN Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.
- Variational Autoencoders (VAE) Optimize variational lower bound on likelihood. Useful latent representation, inference queries. But current sample quality not the best.
- Generative Adversarial Networks (GANs) Game-theoretic approach, best samples! But can be tricky and unstable to train, no inference queries.

Also recent work in combinations of these types of models! E.g. Adversarial Autoencoders (Makhani 2015) and PixelVAE (Gulrajani 2016)



## <Reference>

- 모두의 딥러닝 개정 2판(조태호), 길벗
- <https://leechamin.tistory.com/122>
- [https://taeyoung96.github.io/cs231n/CS231n\\_13/](https://taeyoung96.github.io/cs231n/CS231n_13/)
- <https://zzsza.github.io/data/2018/06/11/cs231n-generative-models/>

- 감사합니다😊!