Generative Adversarial Networks (GANs)

DSE 220

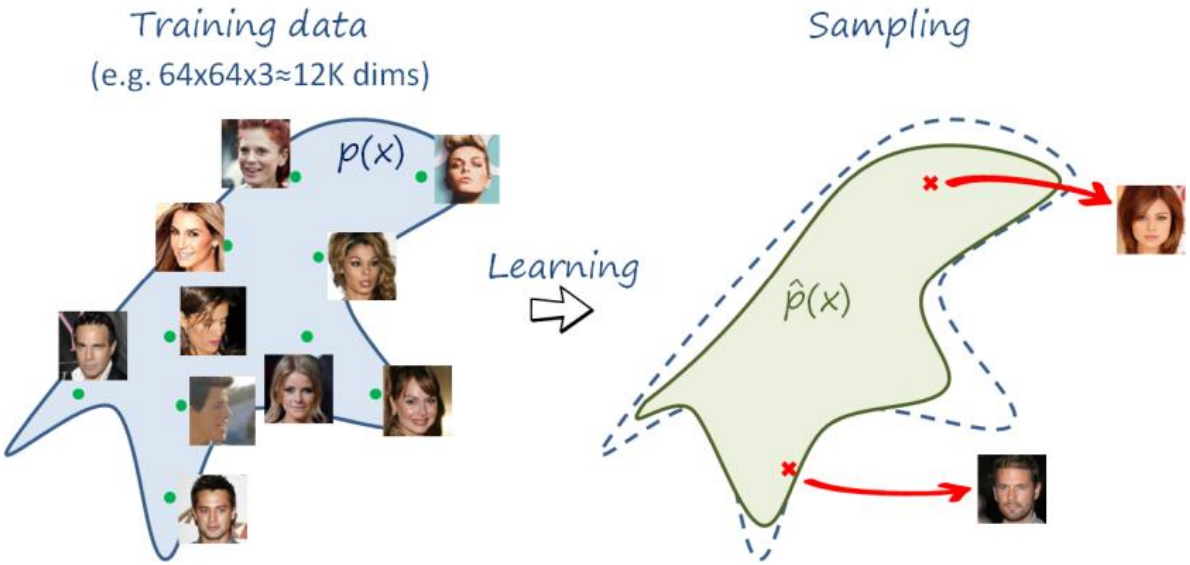
Some slides and images are due to Stanford 231n course

Overview

- Generative Models
- Autoencoders
- Variational Autoencoders
- GANs
  - Overview
  - Training
  - Results
- GAN Zoo (Types of GANs)
  - Deep Convolutional GAN
  - Conditional GAN
  - Cycle GAN
- Implementation

Generative Models

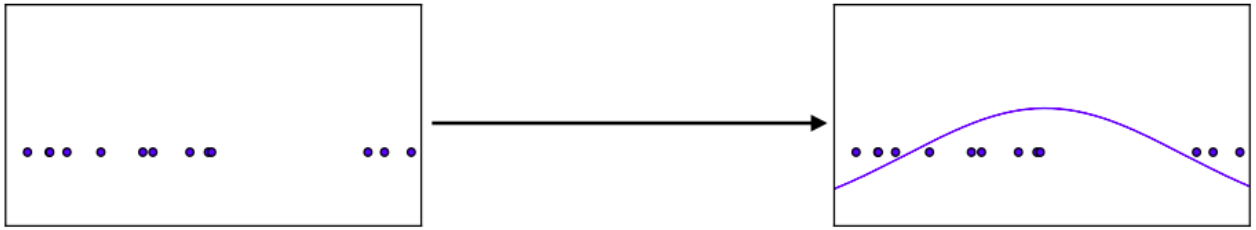
- **Objective**: Given training data, learn the distribution and/ or generate new samples from the same distribution



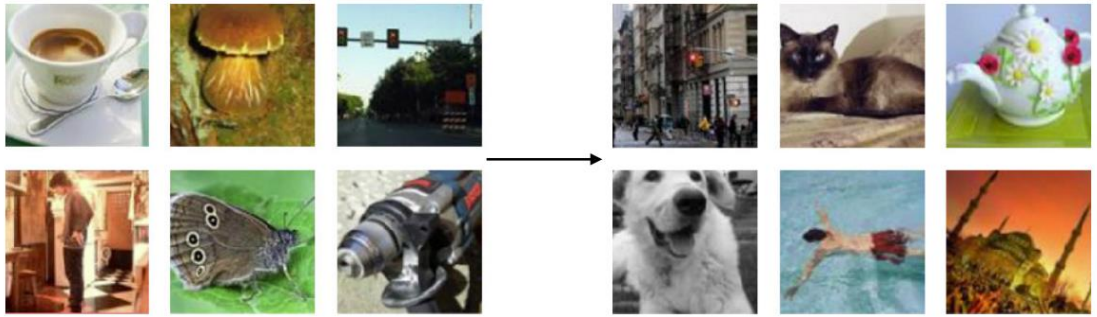
Generative Models

These are of two main types:

1. Explicit density estimation



2. Implicit density estimation (Only able to generate samples)



Why study Generative modeling?

Are generative models worth studying?
Especially those models that are only capable of generating data without the density function.

Yes, we can generate new images.

But the world has no shortage of images!

Why study Generative modeling?

Reasons to study generative models:

1. Training and sampling from generative models is an excellent test of our ability to model high-dimensional probability distributions.

2. Generative models can be incorporated into Reinforcement Learning.
   - RL is either model-based or model-free.
   - Generative models can be used in model-based algorithms.

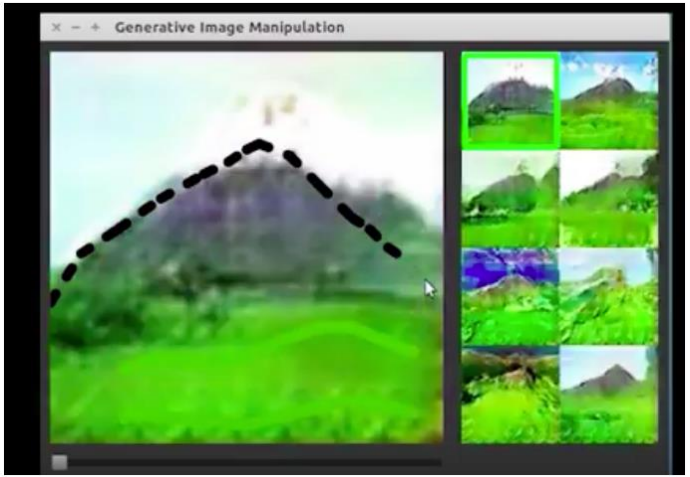
3. Generative models, and GANs, in particular, enable ML to work with *multi-modal* outputs.
   - A single input may correspond to multiple correct answers.
   - Traditional ML models minimizing MSE are not able to do this.

this small bird has a pink breast and crown, and black primaries and secondaries.

Why study Generative modeling?

Reasons to study generative models:

4. Tasks where the goal is to create art.



5. Image to image translation applications.

How do Generative Models work?

Let's start by discussing generative models that work via ***maximum likelihood.***

Some models use maximum likelihood and others don't.
GANs don't use it by default, but can be made to do so.

Basic idea: Define a model that provides an estimate of a probability distribution, parameterized by parameters $\boldsymbol{\theta}$.

We refer to likelihood as: $$\prod_{i=1}^{m} p_{model}\left(x^{(i)}; \theta\right)$$

Maximum Likelihood

Basic idea: Define a model that provides an estimate of a probability distribution, parameterized by parameters $\boldsymbol{\theta}$.

We refer to likelihood as: $\prod_{i=1}^{m} p_{model}\left(x^{(i)}; \theta\right)$

The principle of maximum likelihood says to choose the parameters to maximize the likelihood of the training data.

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \arg\max_{\boldsymbol{\theta}} \log \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right).\end{aligned}$$

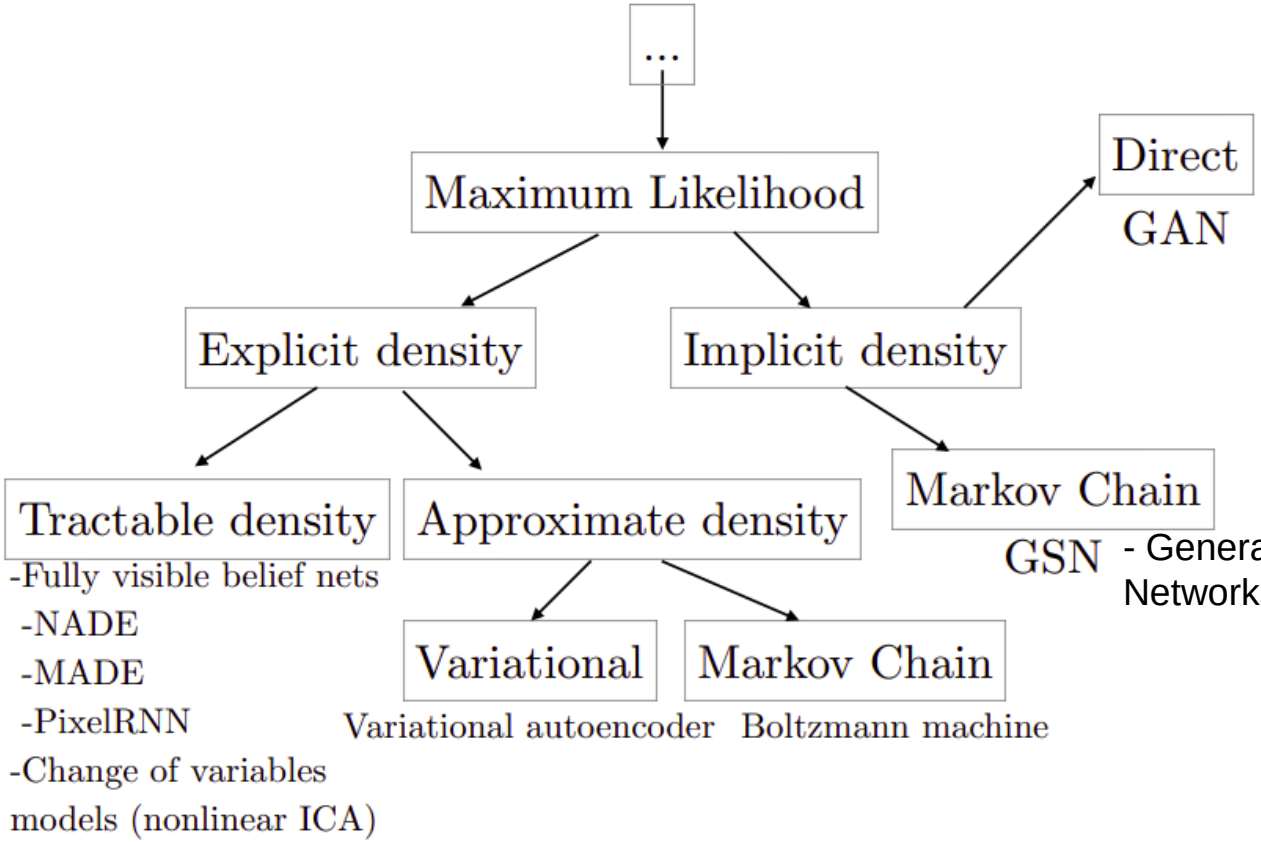
A taxonomy of deep generative models

We can compare different models based on how they compute the likelihood and their gradients, or its approximations.

All of these have some advantages and disadvantages.

GANs were designed to avoid many of these advantages, but they also introduced some new disadvantages.

A taxonomy of deep generative models



A taxonomy of deep generative models

We'll discuss the three most popular generative models that are used today.

1. Fully Visible Belief Networks - Pixel RNN/ Pixel CNN

1. Variational Autoencoder

1. Generative Adversarial Networks

Explicit Density Models

Maximization of the likelihood is straightforward.

- Plug the model's definition of the density function into the expression for likelihood
- Follow the gradient uphill

The main difficulty here is designing a model that can capture all of the complexity while still maintaining computational tractability.

Two strategies are used to handle this:

1. Careful construction of models that allow tractability

1. Models that allow tractable approximations

Tractable Explicit Models

Fully Visible Belief Networks:

They use the chain rule of probability to decompose a probability distribution over an n-dimensional vector ***x*** into a product of one-dimensional probability distributions.

$$p_{\text{model}}(\boldsymbol{x}) = \prod_{i=1}^{n} p_{\text{model}}\left(x_i \mid x_1, \ldots, x_{i-1}\right).$$

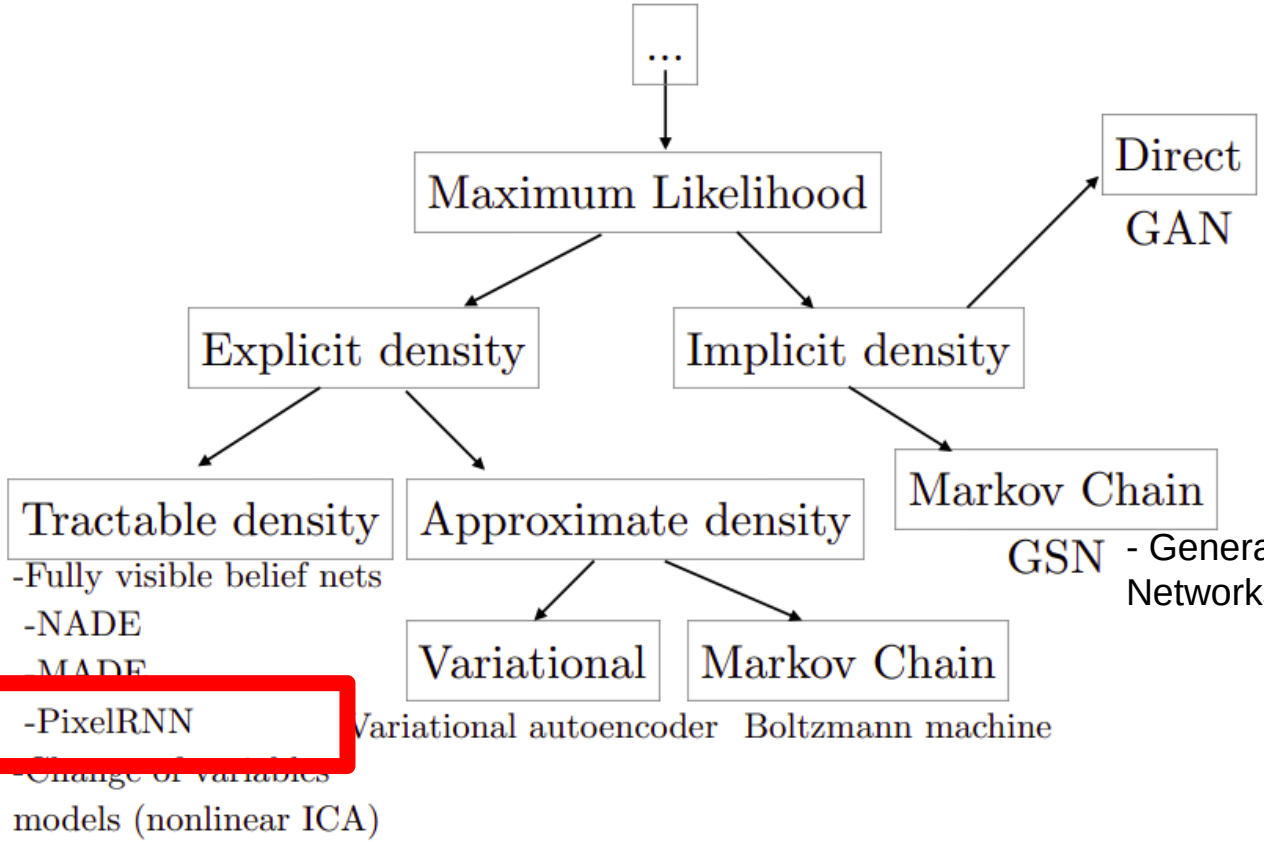
The main drawback of FVBNs is that samples must be generated one entry at a time.

- First ***x1,*** then ***x2***, and so on.
- The cost of generating a sample is O(n).

GANs were designed to be able to generate all of ***x*** in parallel.

Let's take a look at one example of FVBNs.

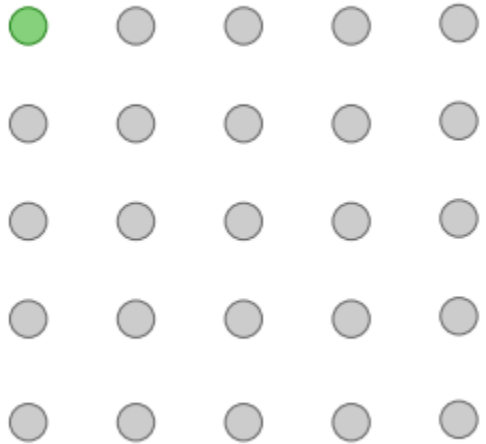
A taxonomy of deep generative models



Pixel RNN

Generate pixels starting from the corner.

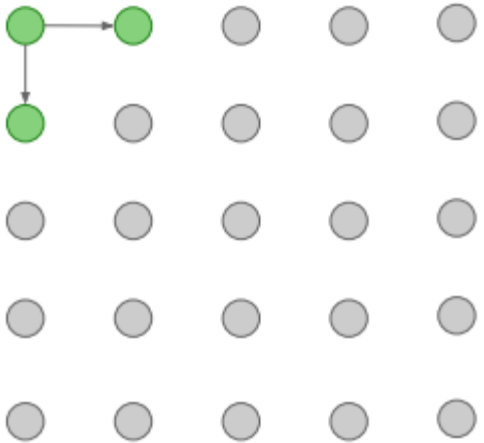
Dependency on previous pixels modeled using an RNN (LSTM).



Pixel RNN

Generate pixels starting from the corner.

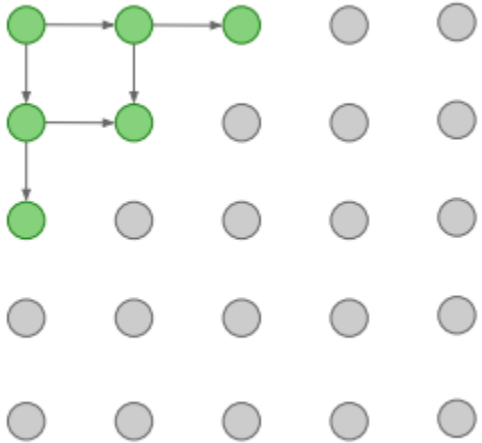
Dependency on previous pixels modeled using an RNN (LSTM).



Pixel RNN

Generate pixels starting from the corner.

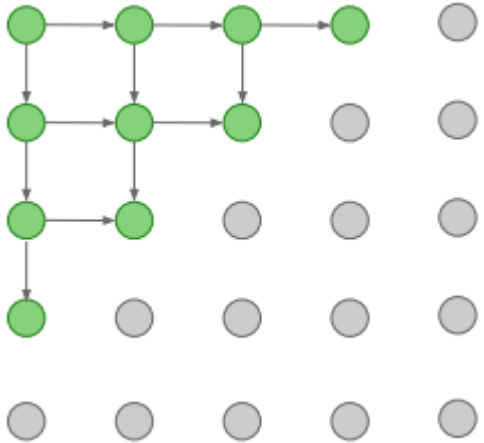
Dependency on previous pixels modeled using an RNN (LSTM).



Pixel RNN

Generate pixels starting from the corner.

Dependency on previous pixels modeled using an RNN (LSTM).

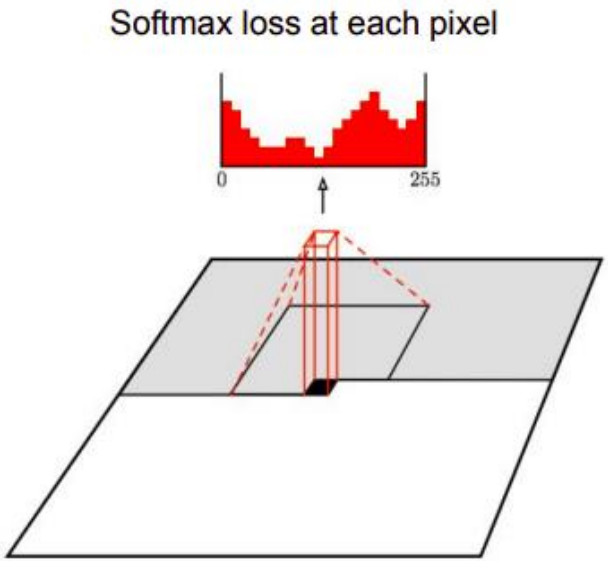


Drawback: Sequential generation is slow!

Pixel CNN

Generate pixels starting from the corner.

Dependency on previous pixels modeled using a CNN over the context region.



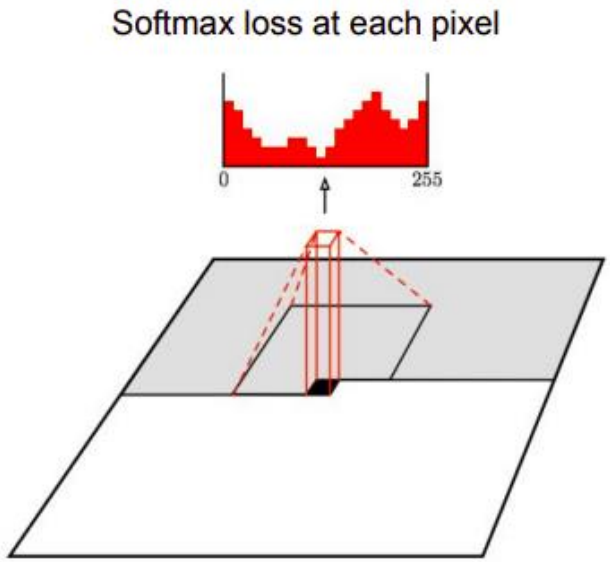
Training is faster than PixelRNN (can parallelize convolutions since context region values known from training images)

Generation must still proceed sequentially => still slow

Pixel CNN

Generate pixels starting from the corner.

Dependency on previous pixels modeled using a CNN over the context region.



- **Training is faster than PixelRNN**
- **Generation must still proceed sequentially => still slow**

Pixel RNN and Pixel CNN

Advantages:

1. Can explicitly compute likelihood P(x)

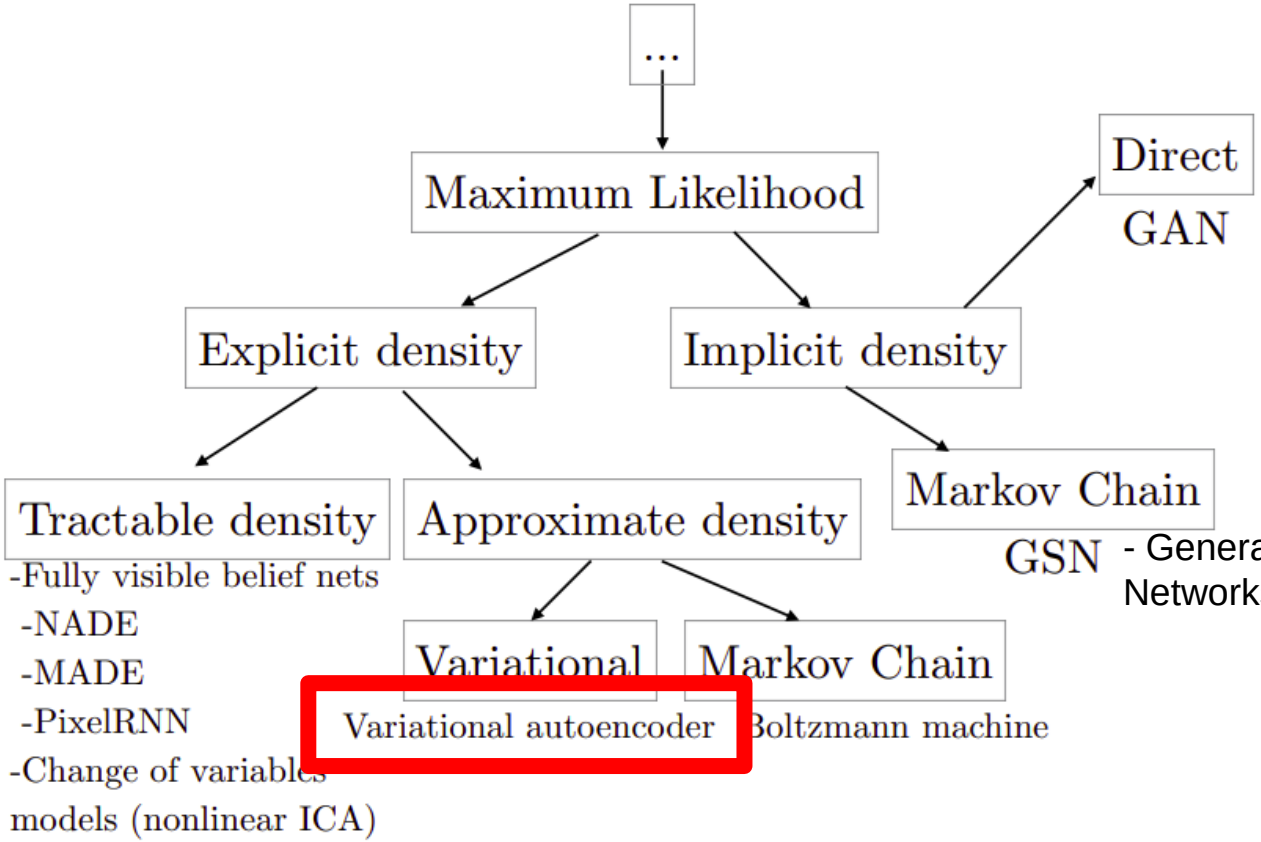
1. Explicit likelihood of training data gives good evaluation metric (GANs lack this)

1. Generates good samples

Drawbacks:

1. Sequential generation is ***slow***.

A taxonomy of deep generative models



Variational Autoencoders (VAE)

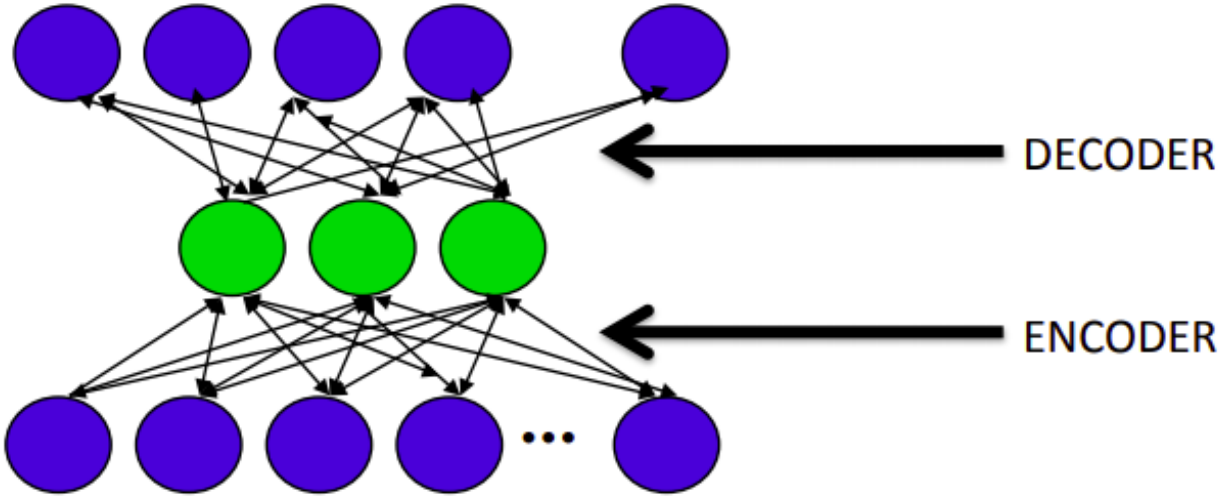
VAEs define an intractable density function with latent ***z.***

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

- Cannot optimize directly.
- Derive and optimizer lower bound on likelihood instead.

Background: Autoencoders

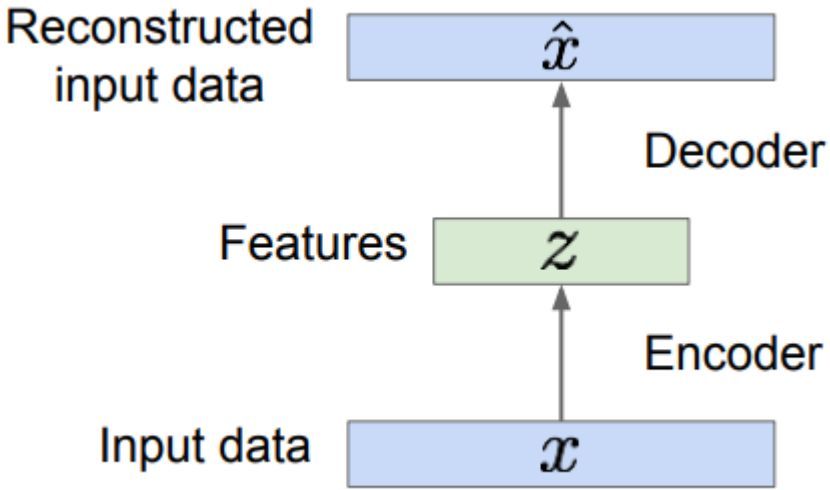
- Learn a lower dimensional feature representation from unlabeled data.
- Why lower dimensional? To capture meaningful data only.



How to do PCA with a neural network (Cottrell, Munro & Zipser, 1987)

Background: Autoencoders

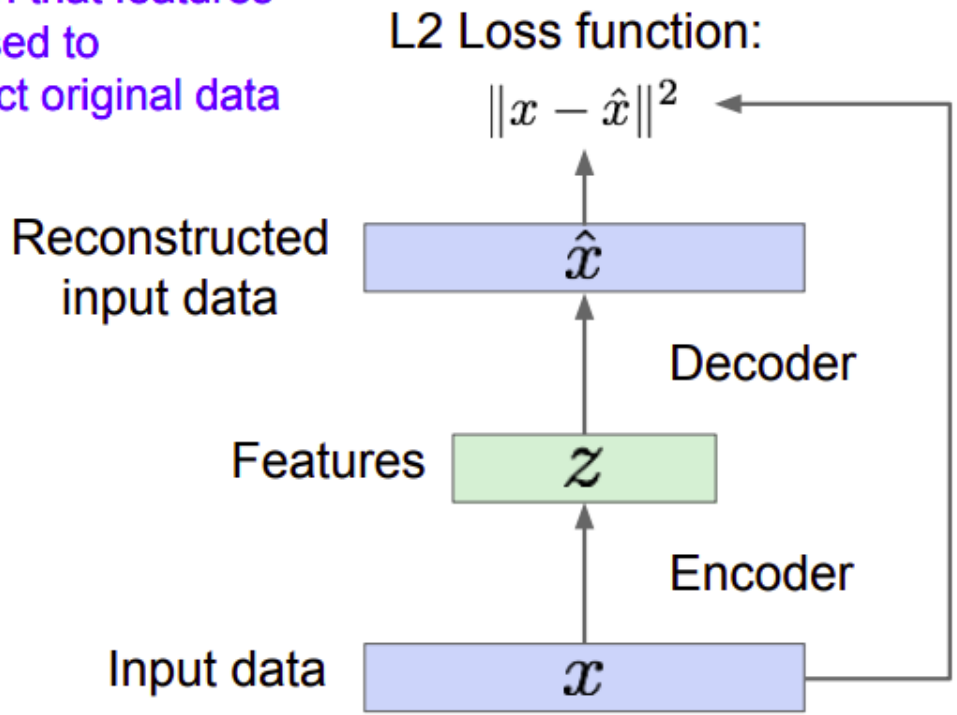
- How to learn the feature representation Z?
  - Train such that features can be used to reconstruct original data.
  - Train using L2 loss function.



Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN

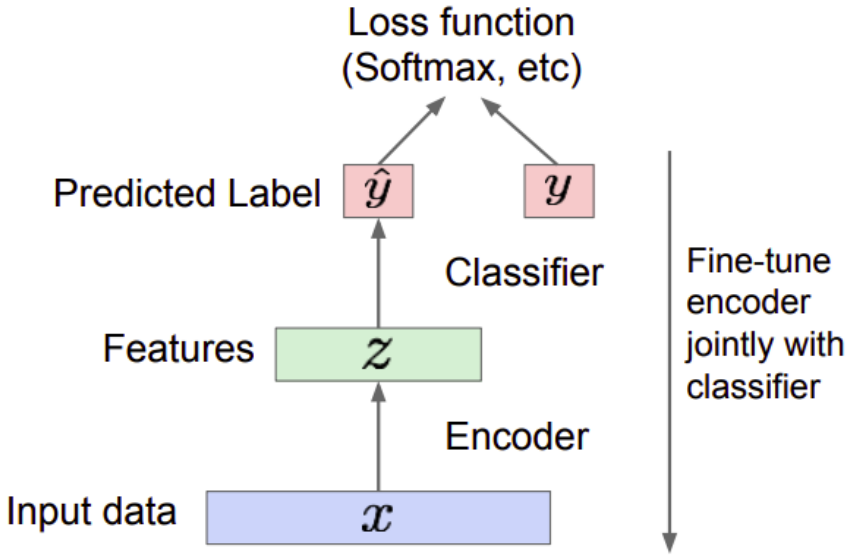
Background: Autoencoders

Train such that features can be used to reconstruct original data



Background: Autoencoders

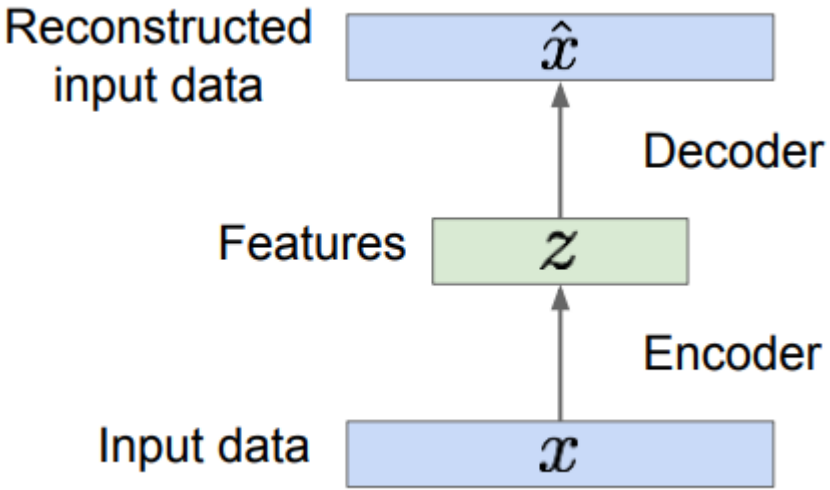
After training the autoencoder, it can be used to initialize a supervised model for classification tasks.



Background: Autoencoders

Autoencoders can reconstruct data, and can learn features to initialize a supervised model.

Features capture factors of variation in the training data.



Can we generate new images using an autoencoder?

Yes, using ***Variational Autoencoders.***

Variational Autoencoders

- Probabilistic spin on autoencoders
- Will let us sample from the model to generate data.

Variational methods define a lower bound:

$$\mathcal{L}(\boldsymbol{x}; \boldsymbol{\theta}) \leq \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}).$$

A learning algorithm that maximizes L obtains as high a value of log likelihood as it does of L.

Variational Autoencoders

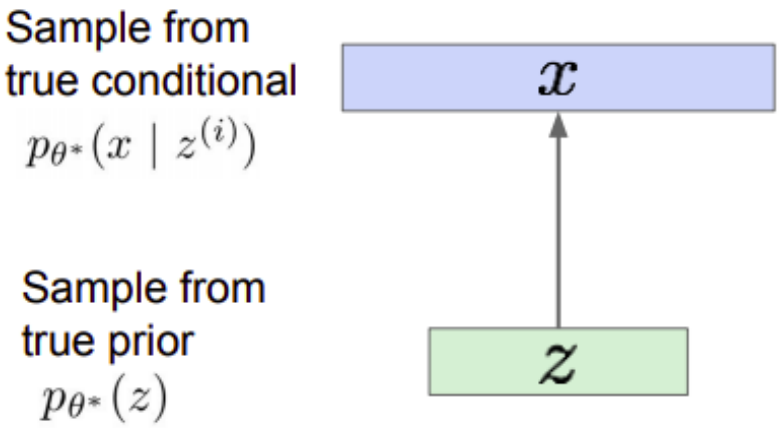
Main drawback:
When L is not a good approximation, then the model doesn't learn the data distribution.

GANs were designed such that with a large enough model and infinite data, the Nash equilibrium for a GAN game corresponds to recovering p_{data} exactly.

Compared to FVBNs, VAEs are more difficult to optimize, but GANs don't optimize this..

Variational Autoencoders

- Assume that training data is generated from an underlying representation ***Z***.



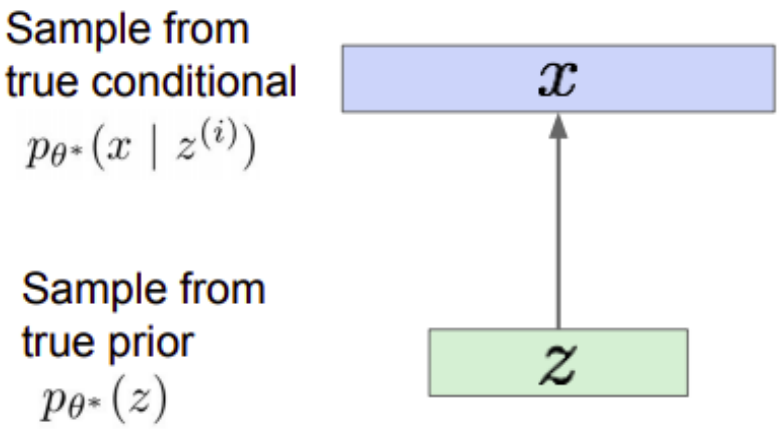
Intuition: ***X*** is an image, ***z*** is the latent factors used to generate the image: attributes, orientation, etc.

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

Variational Autoencoders

- Assume that training data is generated from an underlying representation ***Z***.

We want to estimate the true parameters θ^* of this generative model.



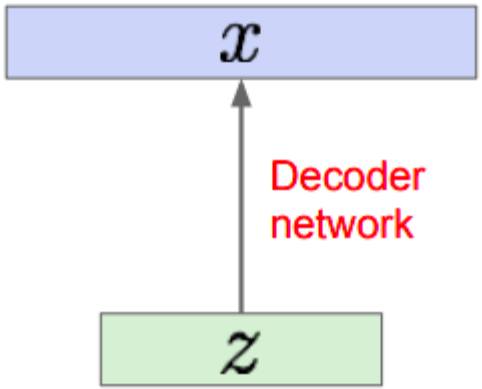
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

- Assume that training data is generated from an underlying representation ***Z***.

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

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



We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

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

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

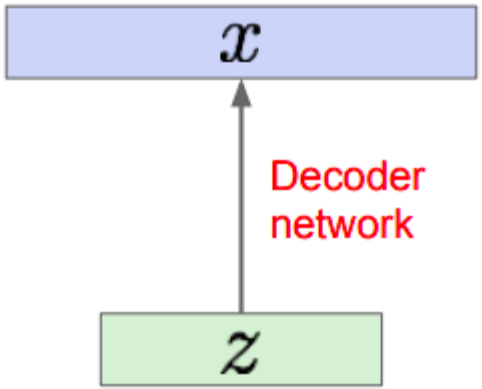
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

- Assume that training data is generated from an underlying representation ***Z***.

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

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



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

Learn model parameters to maximize the likelihood of the training data.

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

But computing this integral over all z is intractable.

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

Variational Autoencoders: Intractability

Data Likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

Intractable to compute p(x | z) for every z.

Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$

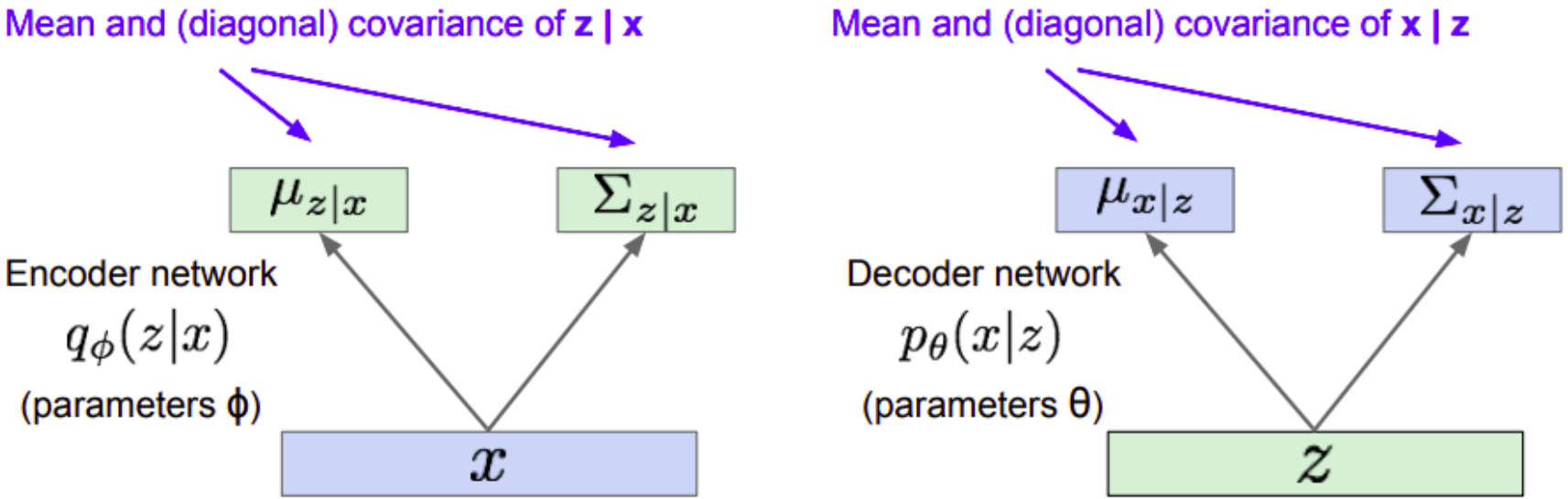
Solution:

In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$.

Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

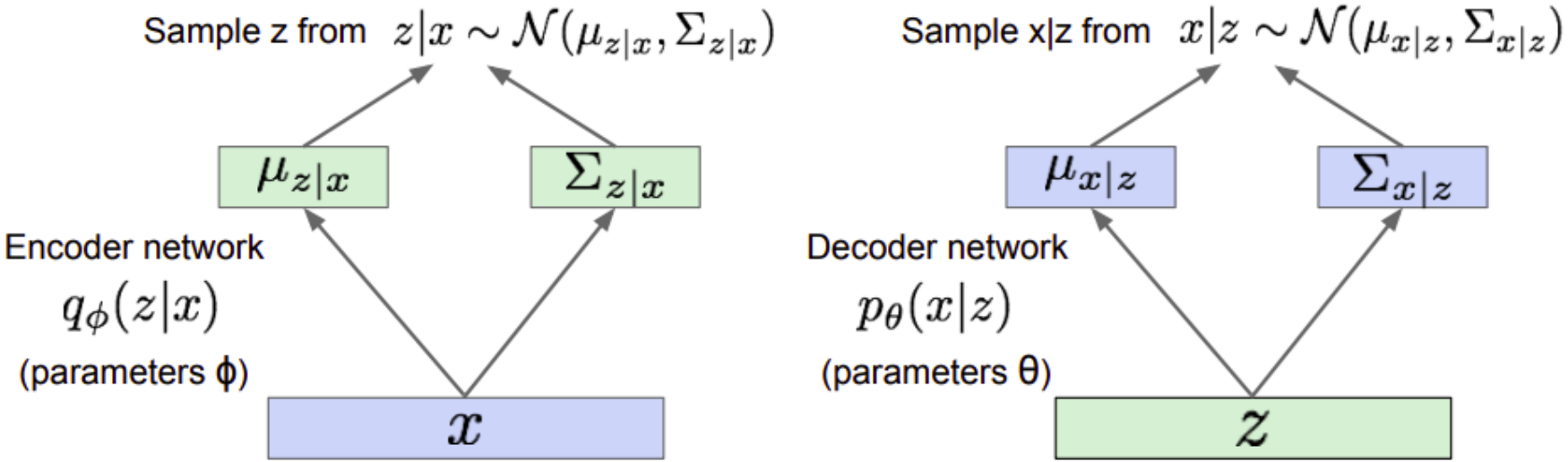
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders



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

Variational Autoencoders



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:

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

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

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

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

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

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:

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

Decoder network gives p_θ (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_θ (z|x) intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

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:

$$\log p_\theta(x^{(i)}) = \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

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

Variational lower bound ("ELBO")

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

Training: Maximize lower bound

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

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

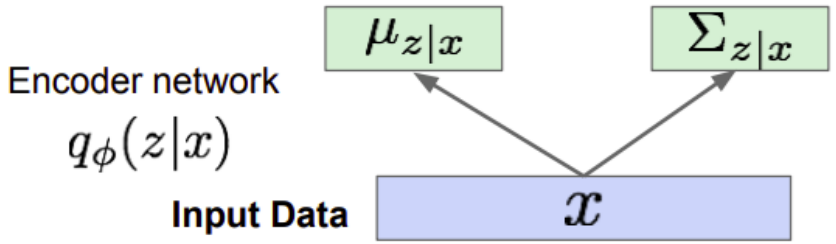
Let's look at computing the bound (forward pass) for a given minibatch of input data

Input Data x

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

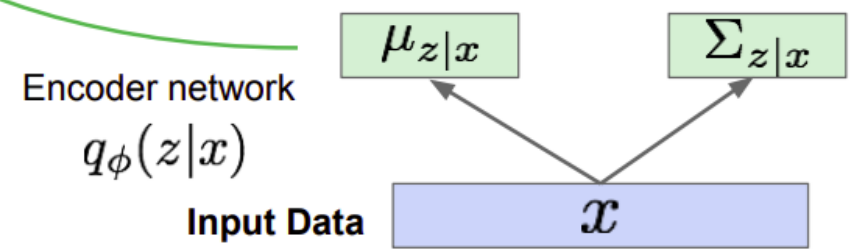


Variational Autoencoders

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

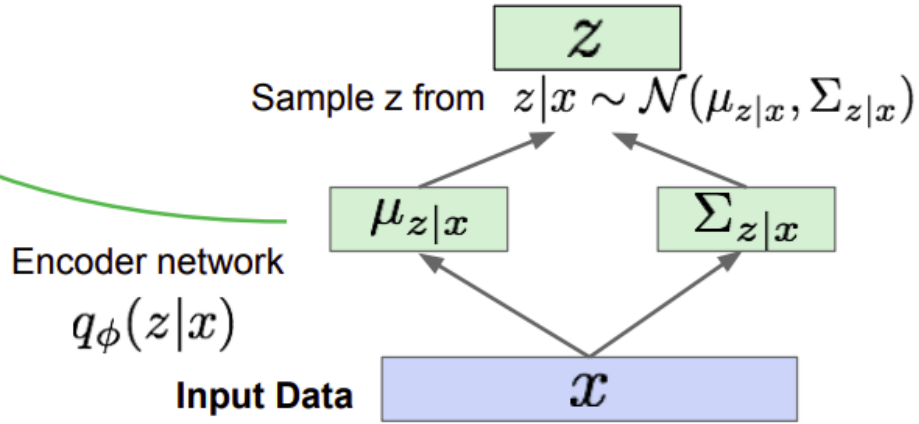


Variational Autoencoders

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

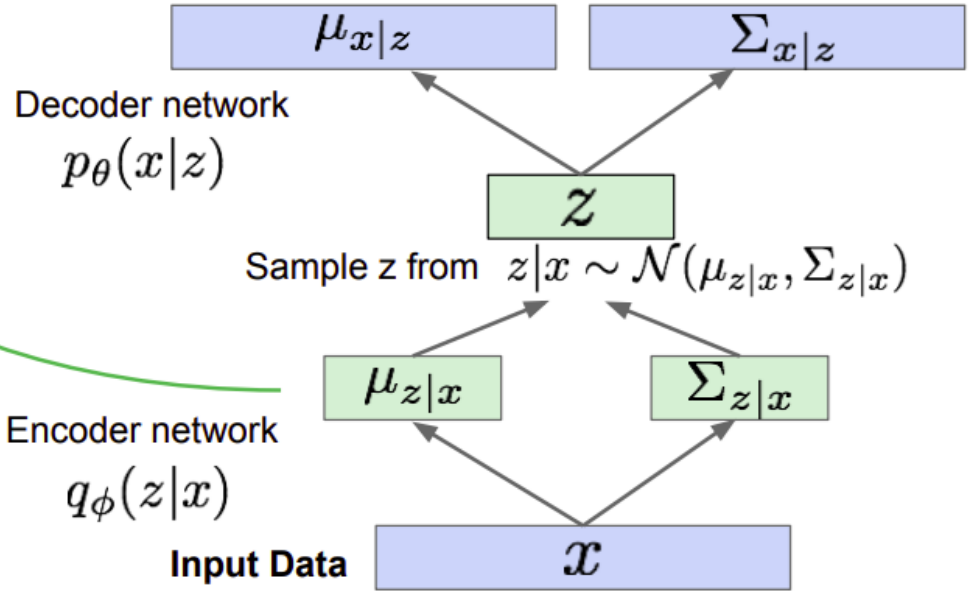


Variational Autoencoders

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

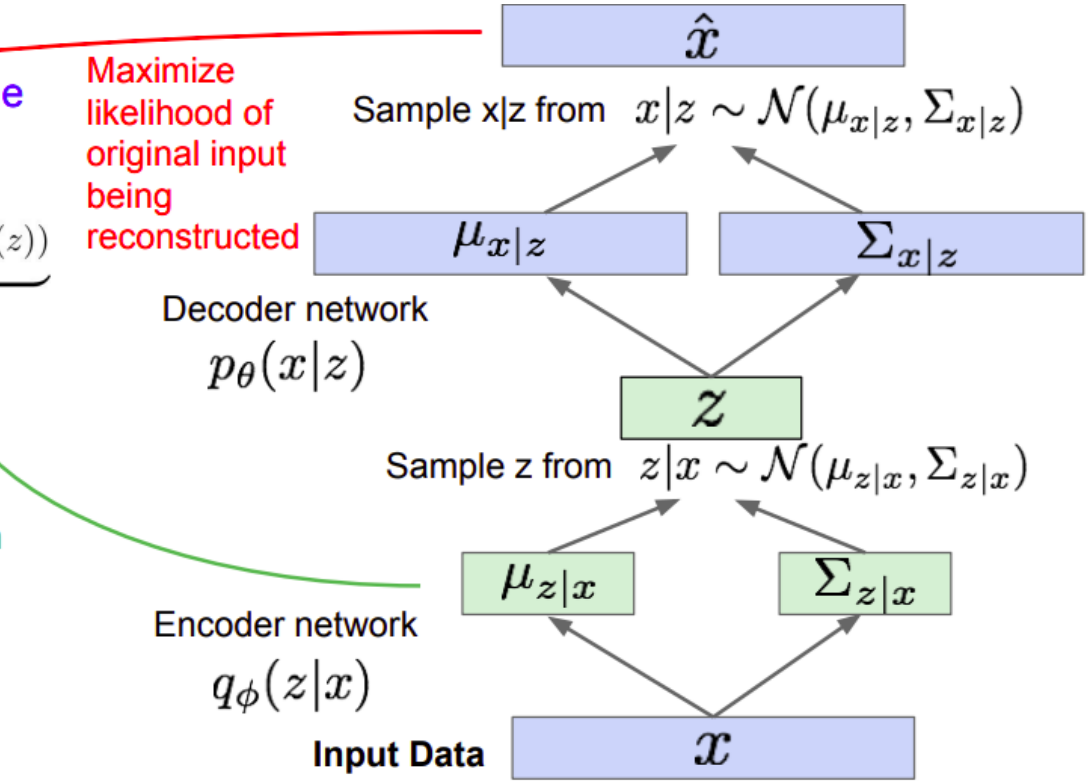


Variational Autoencoders

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



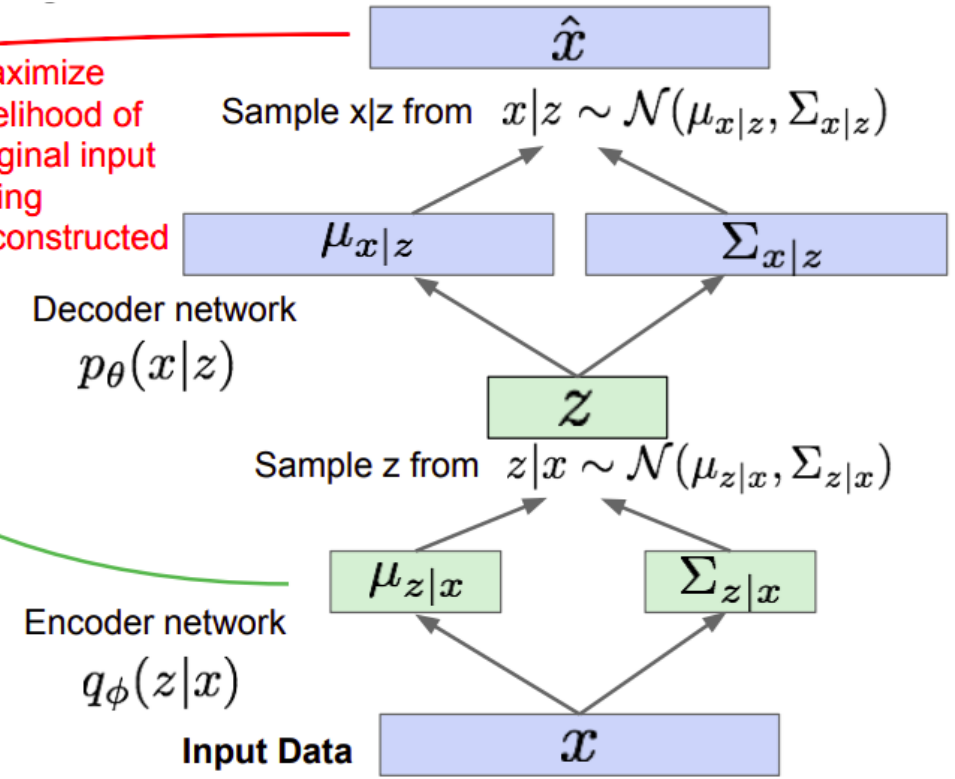
Make approximate posterior distribution close to prior

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

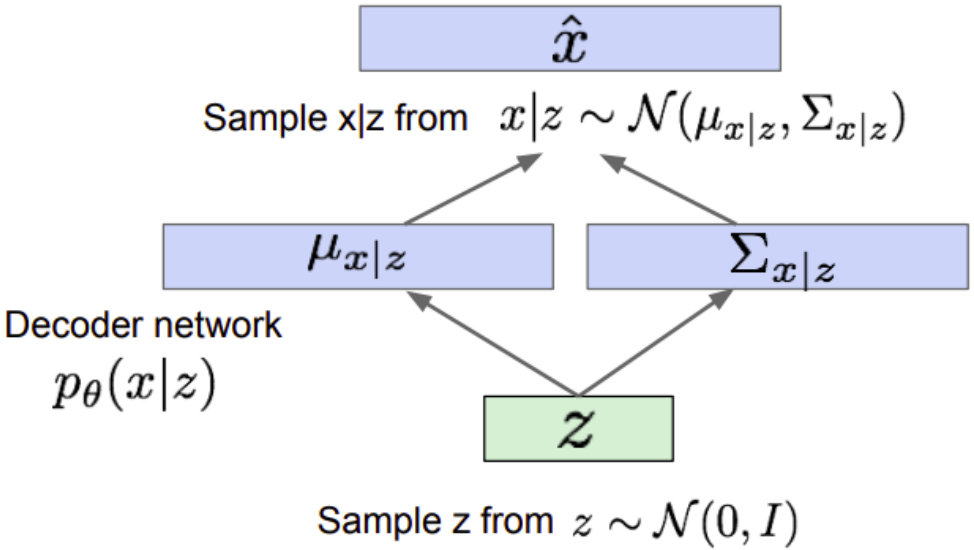


Make approximate posterior distribution close to prior

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

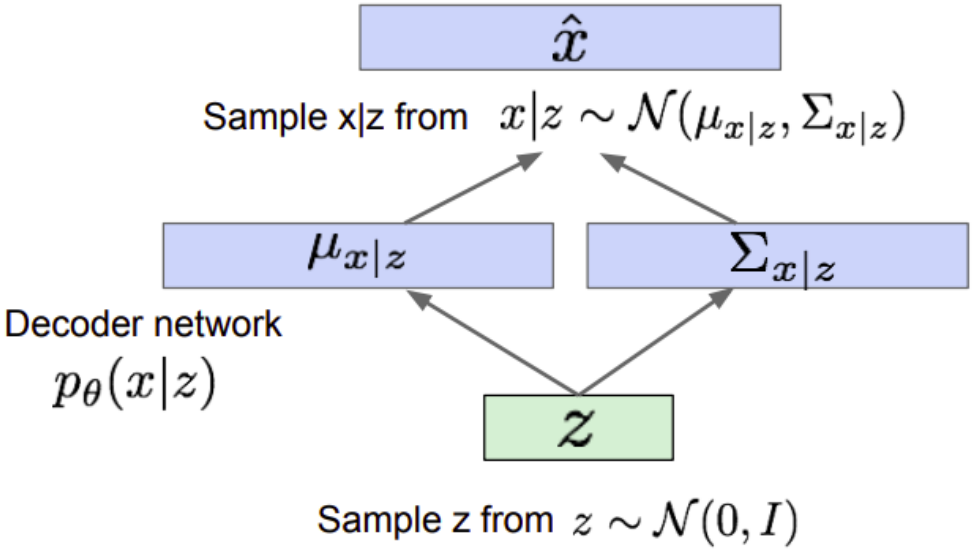
Variational Autoencoders: Generating Data

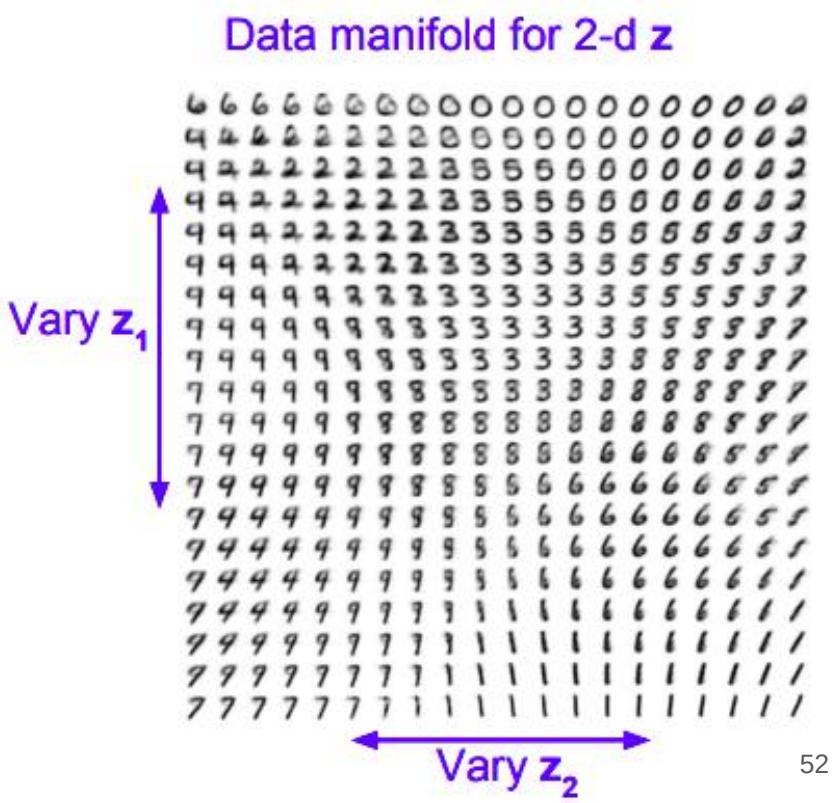
Use decoder network. Now sample z from prior!



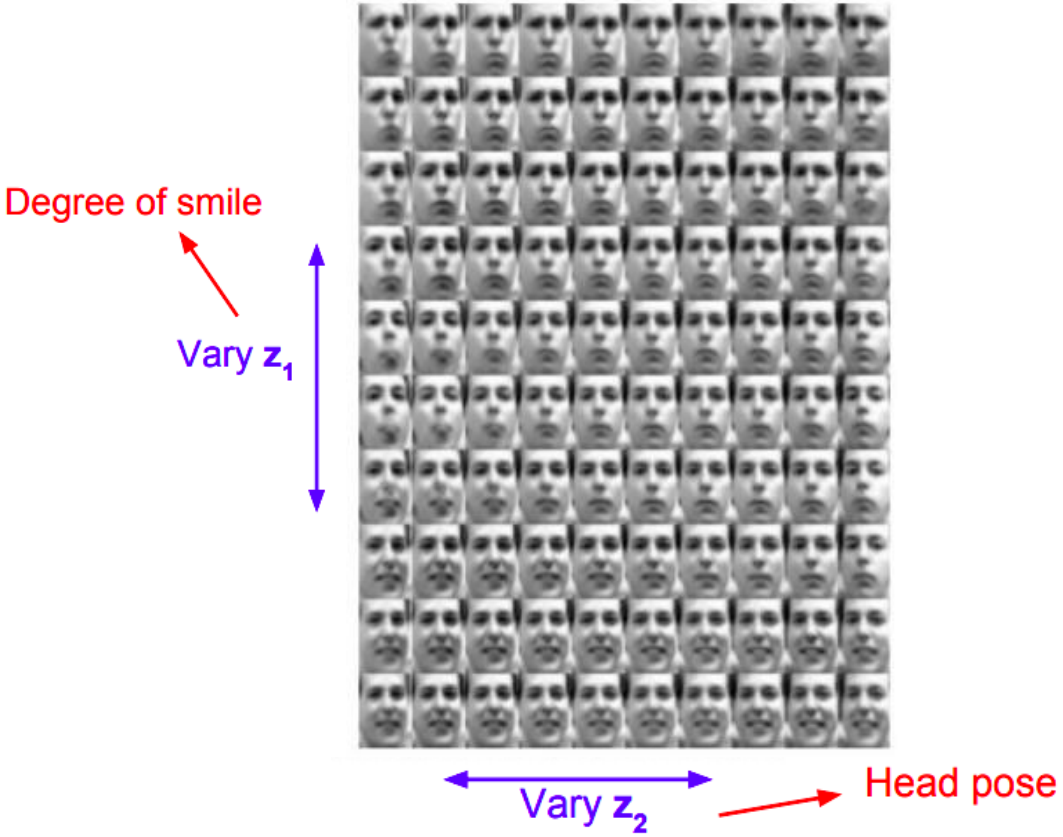
Variational Autoencoders: Generating Data

Use decoder network. Now sample z from prior!





Variational Autoencoders: Generating Data



Variational Autoencoders: Generating Data

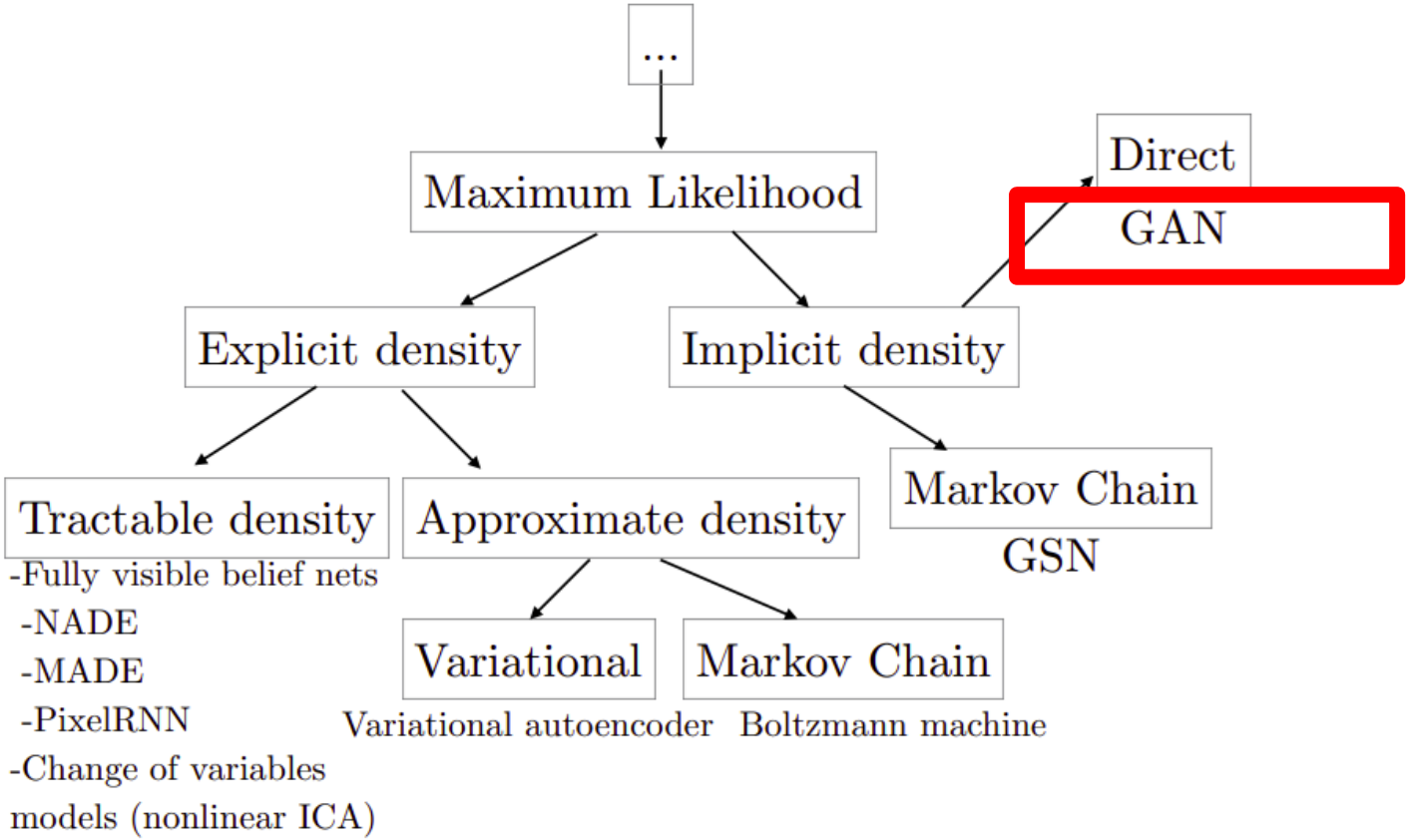
Advantages:

1. Principled approach to generative models, learning distributions
1. Allows learning q(z | x), which can be used for other tasks

Cons:

1. Maximizes lower bound of likelihood: not as good as Pixel CNN/ RNN
1. Samples blurrier and lower quality compared to state-of-the-art

A taxonomy of deep generative models



Implicit density models

Some generative models can be trained without needing to define an explicit density function.

These models allow a way to train the model while interacting indirectly with p_{model} .

Usually, sampling based approximations work reasonably well, as long as a fair sample can be generated quickly.

Some models require the generation of more expensive samples using Markov chains.

Implicit density models

A markov chain is a process for generating samples by repeatedly drawing a sample $\boldsymbol{x' \sim q\,(x' \mid x)}$.
It repeatedly updates $\boldsymbol{x}$ according to the transition operator q.

Markov chain models can sometimes guarantee that $\boldsymbol{x}$ will eventually converge to a sample from p_{model} (x). However, this can be very slow.

Markov chain models fail to scale to high dimensional spaces, and impose increased computational cost.

GANs were designed to avoid this problem.

GANs vs Other Generative Models

- They can generate samples in parallel.
  - This is in advantage of FVBNs.
- The design of the generator function has very few restrictions.
- No markov chains are needed.
- No variational bound is needed.
- GANs are subjectively regarded as producing better samples than other methods.

GANs

- They are like Autoencoders – but there is no encoder.
- There is only a decoder, also called as Generator Network.
- But, there is also an adversary: Discriminator Network.
- The latent dimension Z is random noise.

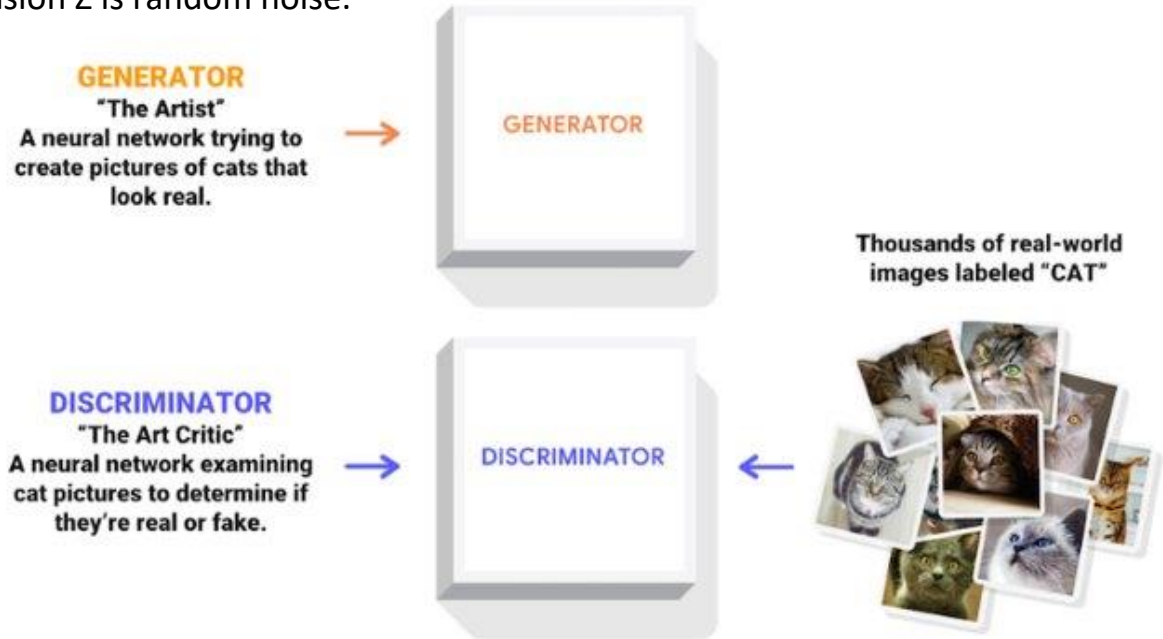


Image: TensorFlow Tutoria

GANs

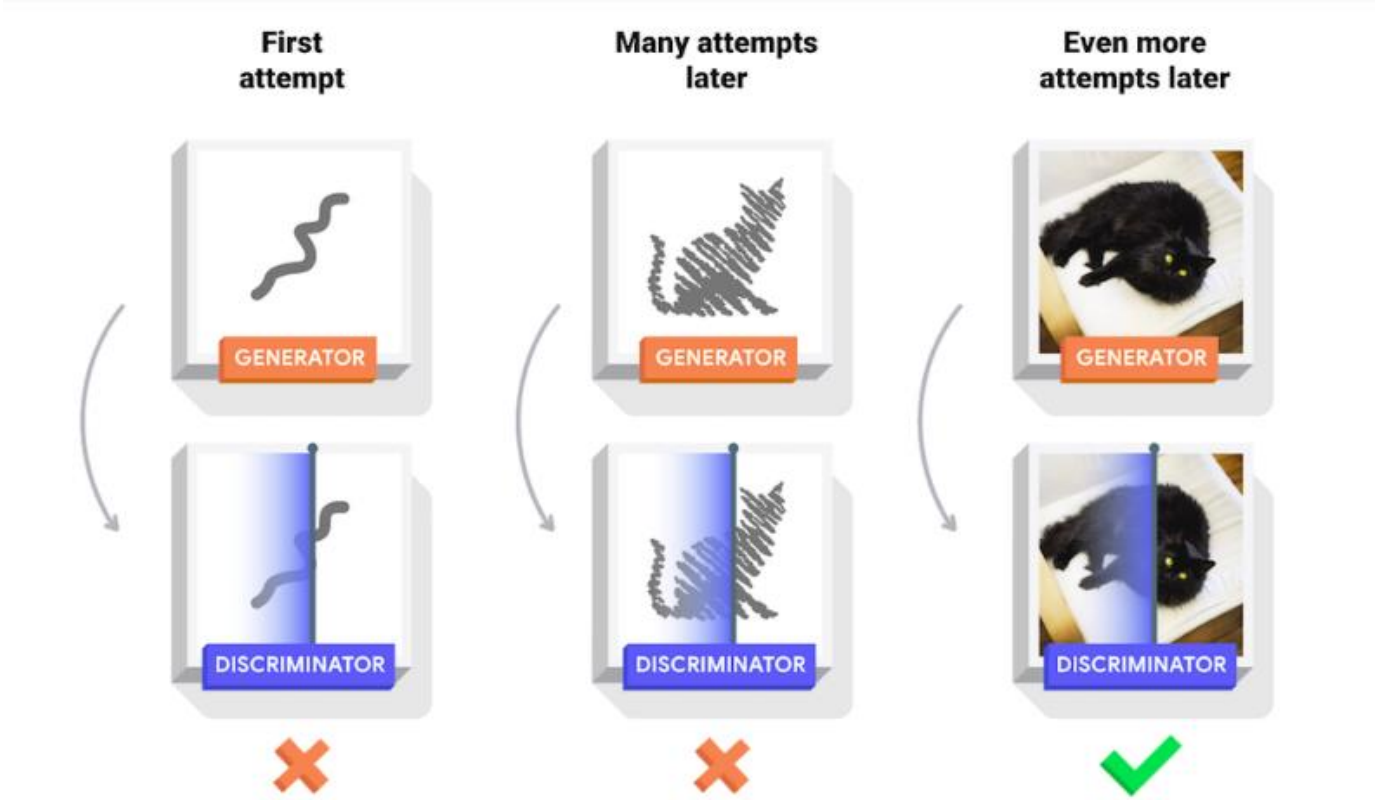
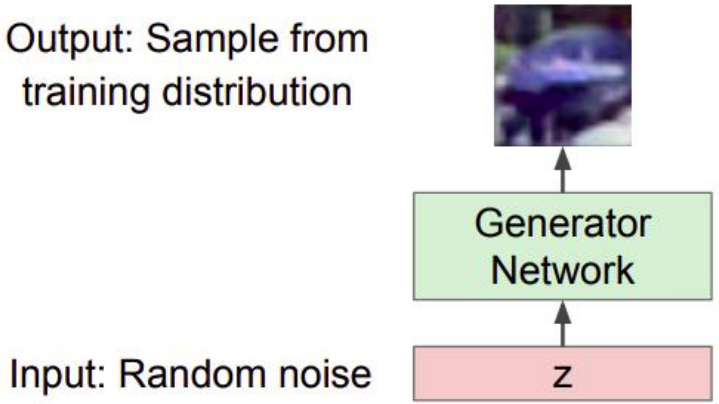


Image Credit: TensorFlow Tutorial

GANs

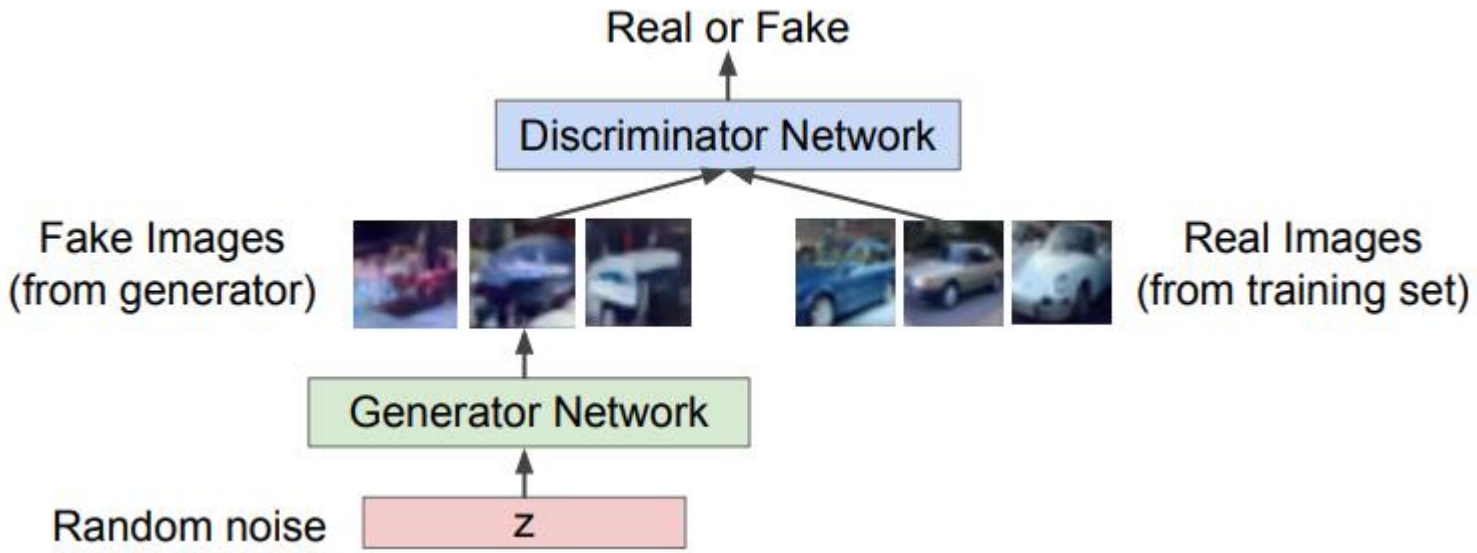
Goal:

- Learn the distribution of the training data
- Starting from random noise
- Learn transformation to training distribution



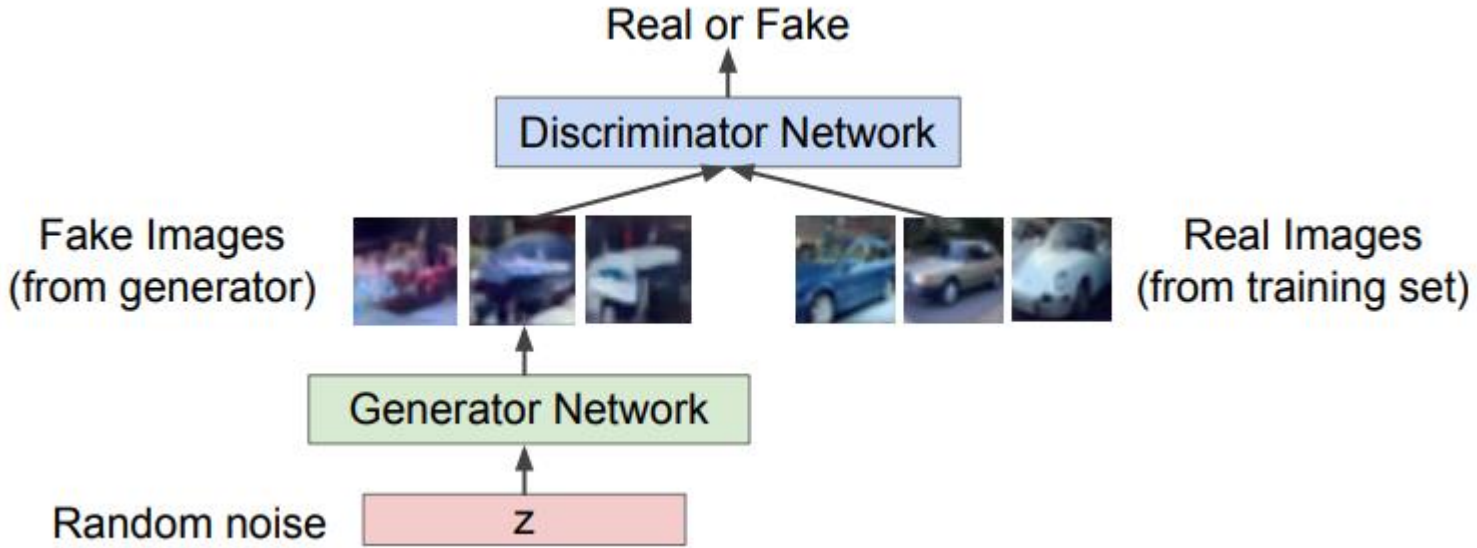
Training GANs: Two-player game

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

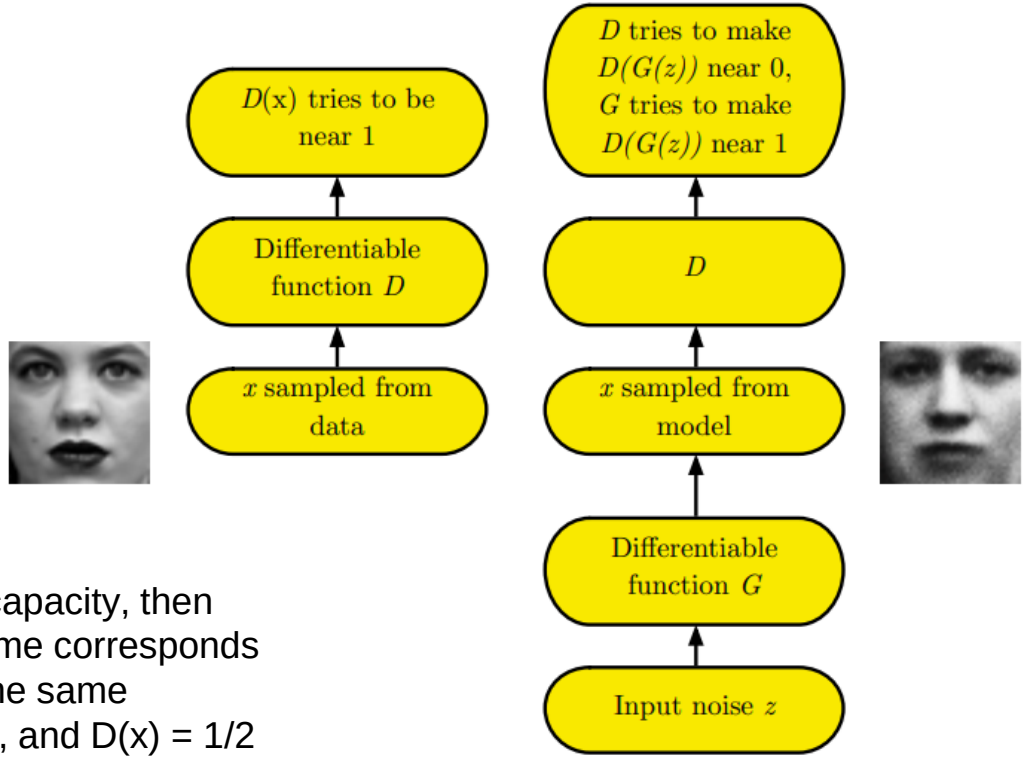


Training GANs: Two-player game

- The generator is trained by backpropagation from the discriminator
- Initially, it will generate garbage.
- But the discriminator will tell it "*that's not an image!*"
- The generator tries to go uphill in discriminator's output
- To fool it, it has to learn to generate good images



Training GANs: Two-player game



If both models have sufficient capacity, then the Nash equilibrium of this game corresponds to the G(z) being drawn from the same distribution as the training data, and D(x) = 1/2 for all x.

Cost Functions

All of the different games designed for GANs so far use the same cost function for discriminator.

They only differ in terms of the cost function used for the generator.

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(z)\right)\right).$$

It can be shown that by training the discriminator, we are able to obtain an estimate to the ratio, $p_{data}(x) / p_{model}(x)$ at every point x.

This is the key approximation that sets apart GANs from variational autoencoders and Boltzman machines.

Cost Functions

We've specified the cost function only for the discriminator.
A complete specification of the game requires the cost function for the generator as well.

The simplest version of the game is a **zero-sum** game, in which the sum of all players' cost is zero.

So,

$J^{(G)} = - J^{(D)}$

Zero-sum games are also called minimax games.

Note: $J^{(G)}$ does not make reference to the training data directly at all; all information about the training data comes only through what the discriminator has learned.
This makes GANs resistant to overfitting, because the generator has no opportunity in practice to directly copy training examples.

Training GANs: Two-player game

- Train jointly in **minimax game**

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

Minimax objective function:

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

Discriminator output for real data x

Discriminator output for generated fake data G(z)

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

Training GANs: Two-player game

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

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

Training GANs: Two-player game

But there is a problem here.

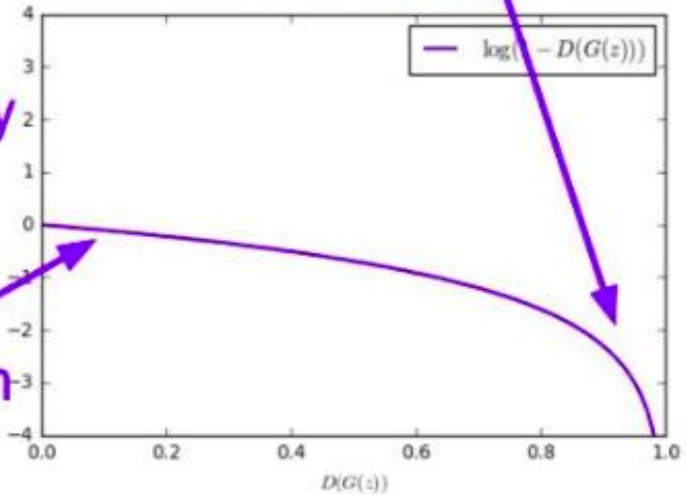
In practice, the generator doesn't do very well with this objective function.

Gradient descent on generator

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

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Training GANs: Two-player game

So, change the objective function of the generator.

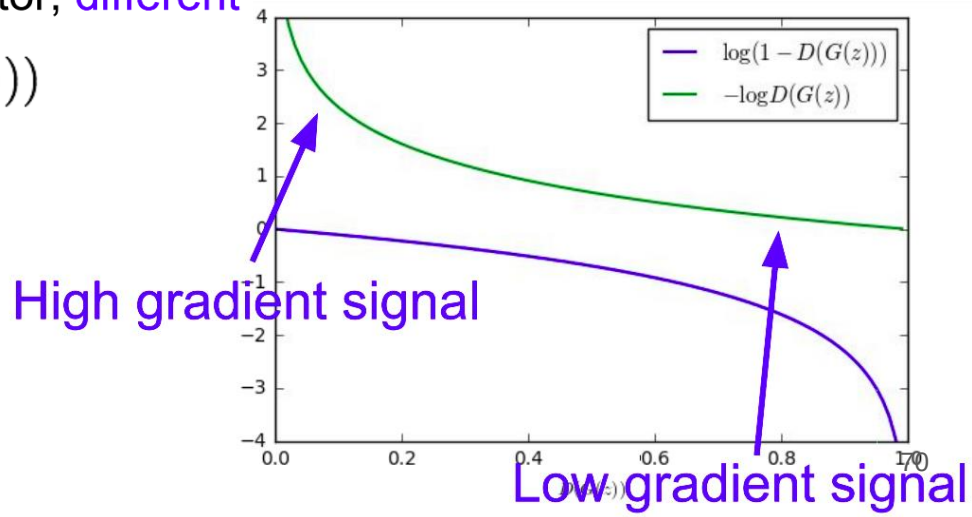
Alternate between:

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

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



Training GANs

The discriminator wishes to minimize $J^{(G)}$ and must do so only while controlling its own parameters.

The generator wishes to maximize $J^{(G)}$ and must do so only while controlling its own parameters.

Because each player's cost depends on the other player's parameters, but each player cannot control the other player's parameters, this scenario is best described as a game rather than an optimization problem.

Training GANs

The solution to an optimization problem is a (local) minimum.

The solution to a game is a Nash equilibrium - A Nash equilibrium is a tuple $\left(\theta^{(D)}, \theta^{(G)}\right)$ that is a local minimum of J(D) with respect to $\theta^{(D)}$ and a local minimum of J(G) with respect to $\theta^{(G)}$.

Training GANs : Algorithm

for *number_iterations* do:

for *k* steps do:

- Sample a minibatch of m fake samples from the generator
- Sample a minibatch of m real examples from the data $p_{data}(x)$.
- Update the discriminator by ***ascending*** its stochastic gradient.

end for

- Sample a minibatch of m fake samples from the generator
- Update the generator by **ascending** its stochastic gradient.

end for

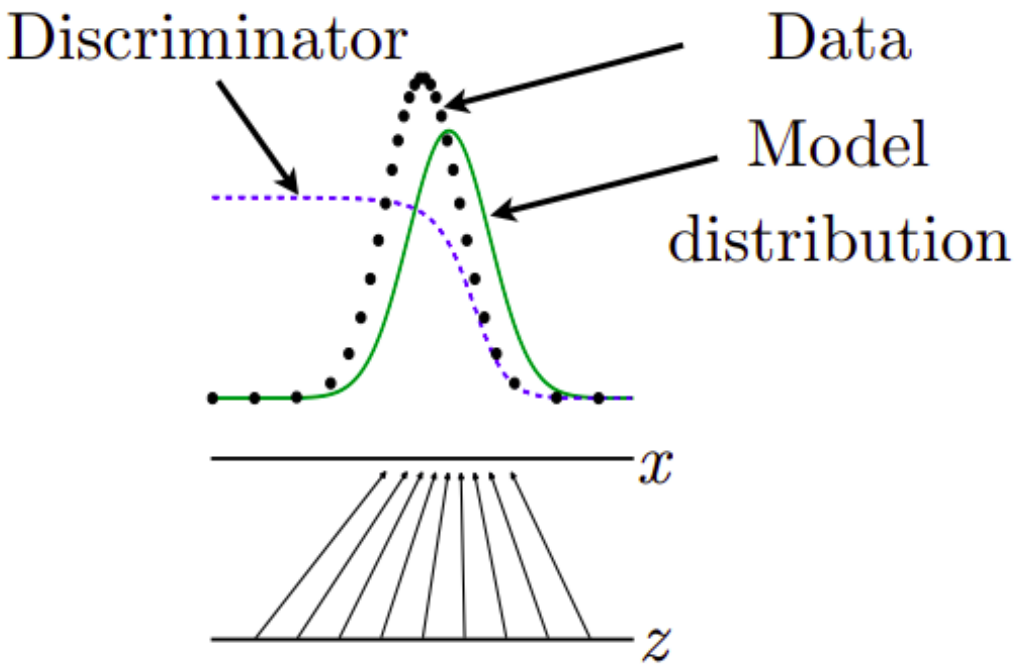
Question: When should you stop training?

Training GANs : Stopping

When to stop training?
- When the generator and the discriminator reach equilibrium.
- The losses stabilize.

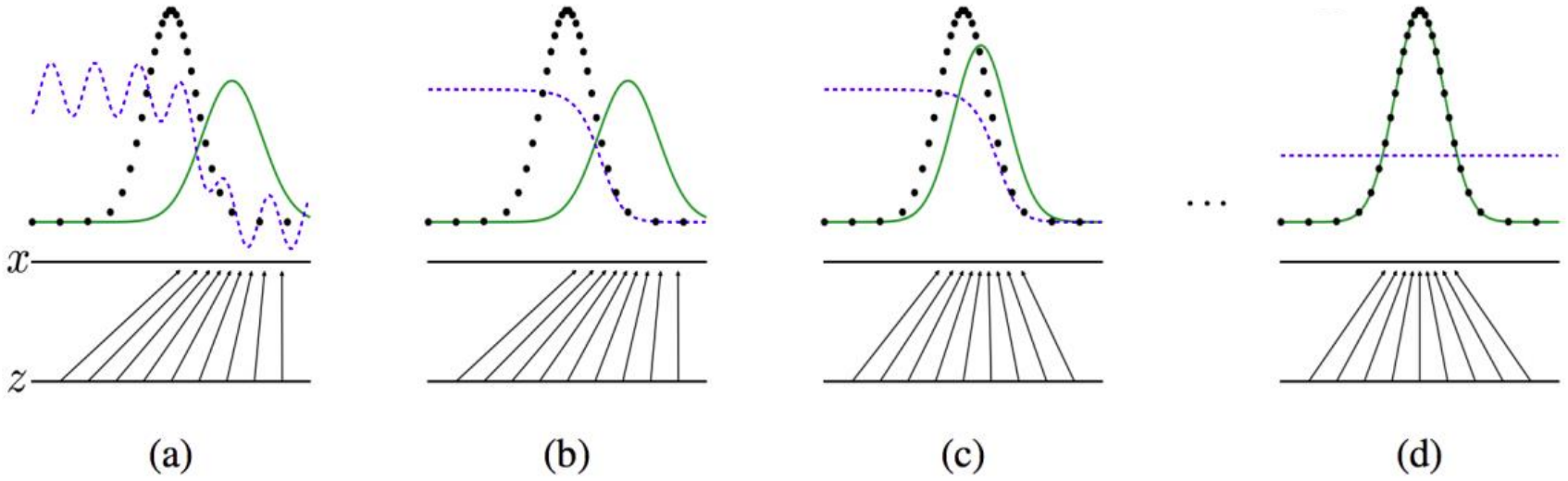
The value log(2) = 0.69 is a good reference point for these losses, since it indicates that the discriminator is equally uncertain about the real and the fake images.

Training GANs : Stopping



In this example, we assume that both *x* and *z* are one dimensional.

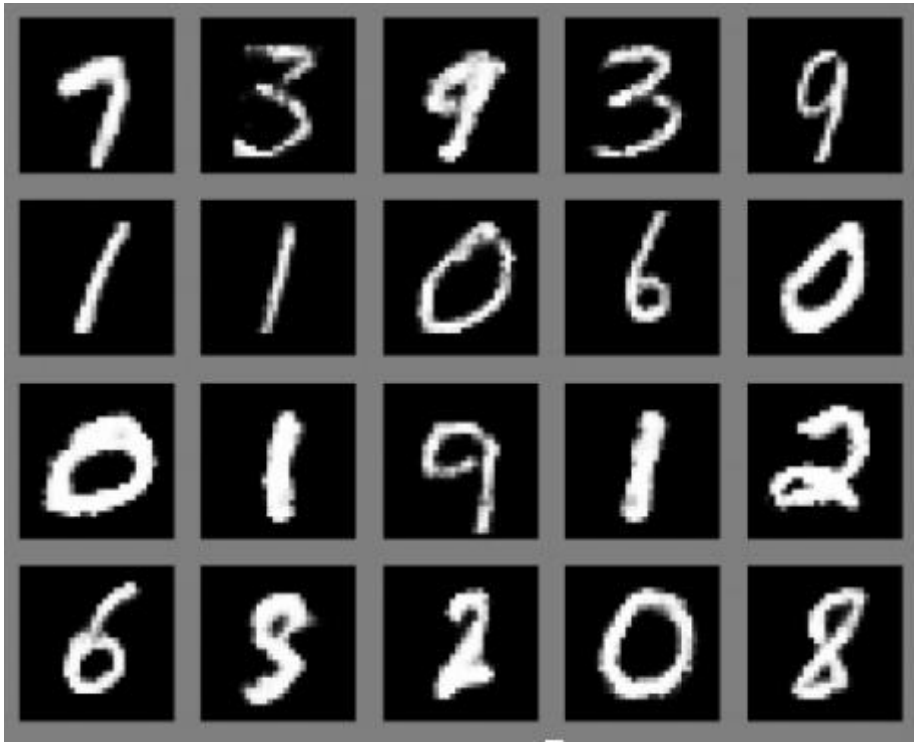
Training GANs : Abstract View



(b) The discriminator has learnt to distinguish real from fake
(c) The generator has learned to move its distribution closer to the real one
(d) The discriminator and generator have reached equilibrium, and the discriminator can no longer tell real from fake

Ian Goodfellow et. Al. - Generative Adversarial Nets

GANs : Results



Original results on the MNIST dataset. (2014)

Ian Goodfellow et. Al. - Generative Adversarial Nets

GANs : Results

Question: Why don't we just memorize the input data and output a random image from that when asked to generate a new image?

- Because we won't have any ***new*** generated data.

Question: How do we know the model is not memorizing data?

GANs : Results

The image in the left column is generated by GAN.
The image in the right column is the nearest neighbour of the generated image in the training set.

It shows that the model has not just memorized the input data, but has really ***learnt*** something.

Ian Goodfellow et. Al. - Generative Adversarial Nets

GANs : Results (2017)

(a) Church outdoor.

(b) Dining room.

(c) Kitchen.

Images generated with LSGAN – Mao et. al

The GAN Zoo

There has been an explosion in the types of GANs after the original paper by Ian Goodfellow et al.

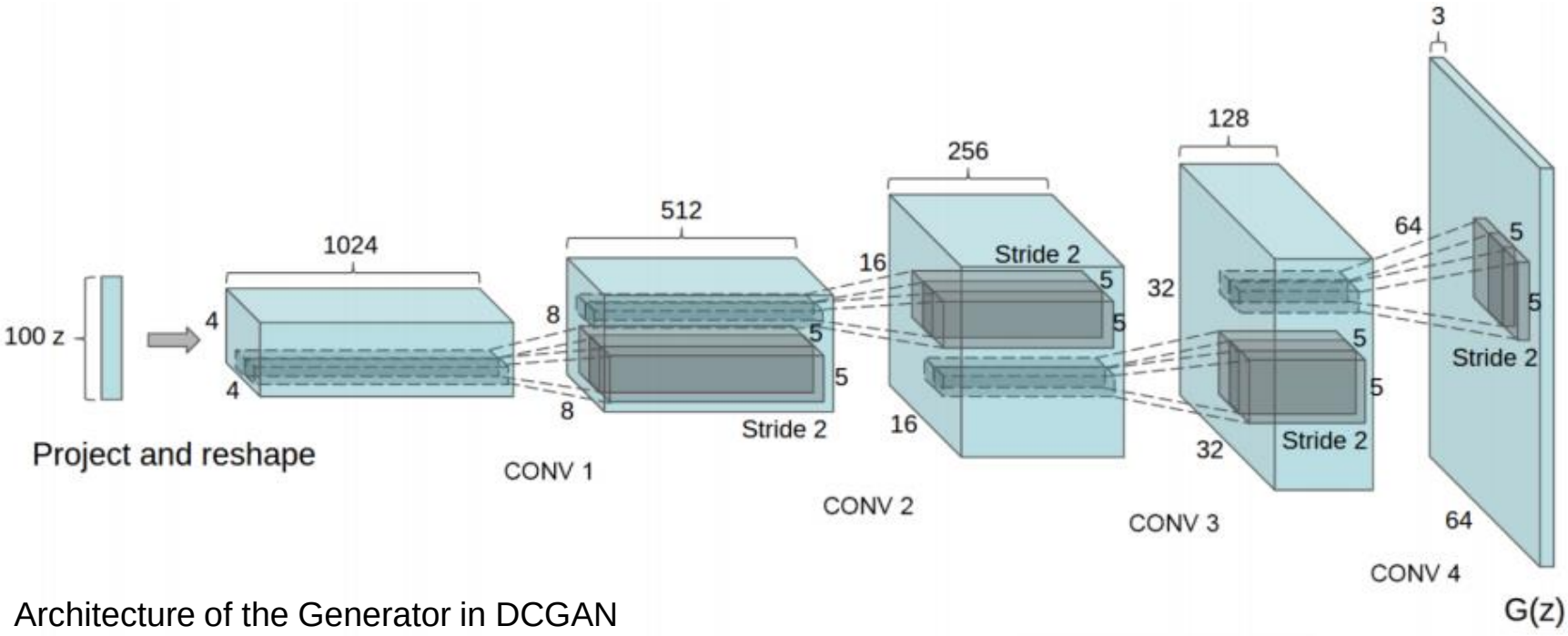
Examples:

Deep Convolution GAN: Using CNNs as the Generator and Discriminator networks

Conditional GANs: Generating data based on a prior condition

Cycle GANs. : Learning to map from one domain to another without any supervision.

Convolutional GAN



Architecture of the Generator in DCGAN

(Radford et al, “Unsupervised Representation Learning with Deep Convolutional GANs”, ICLR 2016)

Guidelines for training: Convolutional GAN

Guidelines for training deep convolutional GANs:

- Use batch normalization in both the generator and the discriminator
- Remove fully connected layers for deeper architectures
- Use ReLU activation in generator for all layers except the output, which uses Tanh
- Use Leaky ReLU activation in the discriminator for all layers

(Radford et al, “Unsupervised Representation Learning with Deep Convolutional GANs”, ICLR 2016)

Conditional GAN

How do you generate images of a specific class?
Say, images of class ‘3’ in MNIST, or images of only ‘cars’.

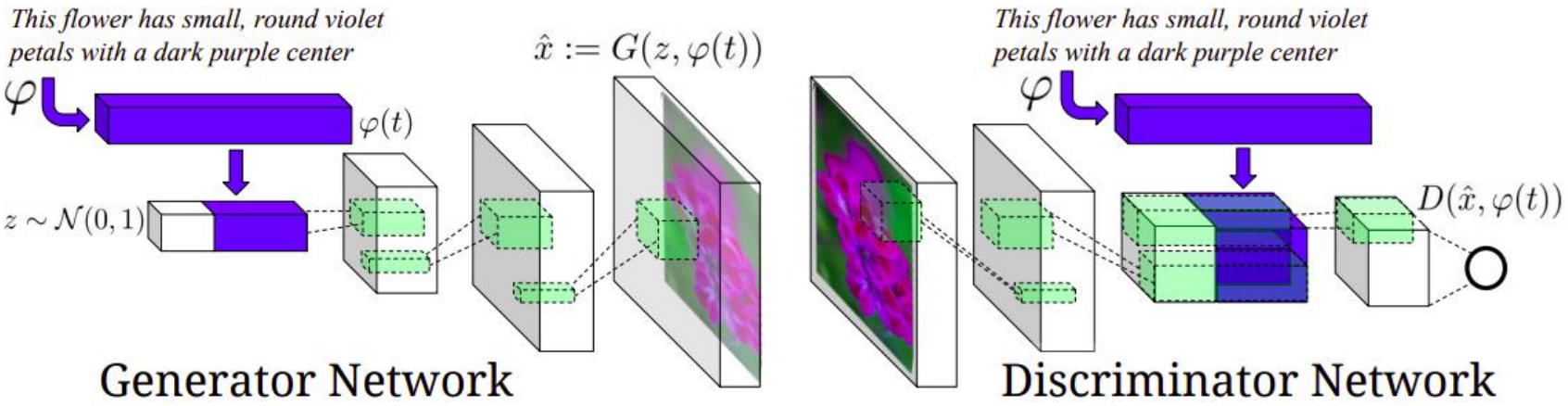
Generate images based on an input condition. This is used along with the noise vector to generate the output image.

Here, each row is conditioned on a class.

Conditional GANs - Mehdi Mirza, Simon Osindero (2014)

Conditional GAN

Use case: Text to image synthesis



Hands-on !!

Generative Adversarial Text to Image Synthesis – Scott Reed et. al

Cycle GAN

Use case: Image to image translation using GANs

Hands-on !!

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks – Zhu et. al

Recap...

- Works without an explicit density function.
- Learns to generate from the training distribution through a minimax game.

Pros:
- Beautiful, state of the art images
- Check this out: https://www.thispersondoesnotexist.com/

Cons:
- Unstable to train
- Can’t solve inference queries, such as p(x), p (z|x)

Recap...

Pixel RNN/ CNNs:
- Explicit density model
- Optimizes exact likelihood
- Good samples
- Inefficient sequential generation

Variational Autoencoders (VAE):
- Optimize variational lower bound on likelihood
- Useful latent representation
- Sample quality not the best

GANs:
- Game theoretic approach
- State-of-the-art samples
- Unstable to train
- No inference queries

How to implement?

1. Basic GAN on MNIST
1. Conditional GAN - Text to Image Synthesis
1. Cycle GAN - Image to Image translation

Go to Hands-on.