Variational Autoencoders (VAEs) (chapter 20)

Generative Models

Aim: learn a model so that $\mathbf{p_{model}}(x)$ is as close as possible to $\mathbf{p_{data}}(x)$

Two main families of approaches

- Explicit density estimation: explicitly define and solve for $\mathbf{p_{model}}(x)$
  - up to now we have studied models from the Boltzmann Machine family:
    - they use an approximation of the density via Markov Chains (Gibbs sampling)
    - Variational Autoencoders (VAEs) use an approximation of the density via a variational approach (ELBO)
- Implicit density estimation: learn a model that can sample from $\mathbf{p_{model}}(x)$ without explicitly defining it
  - Generative Adversarial Networks (GANs) use a direct approach based on Game Theory

Both VAEs and GANs exploit differentiable generator networks

Evidence Lower Bound (ELBO)

$$\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q) = \log p(\boldsymbol{v}; \boldsymbol{\theta}) - \underbrace{D_{\mathrm{KL}}\left(q(\boldsymbol{h} \mid \boldsymbol{v}) \| p(\boldsymbol{h} \mid \boldsymbol{v}; \boldsymbol{\theta})\right)}_{\geq 0}$$

where q is an arbitrary probability distribution over $\boldsymbol{h}$

Kullback–Leibler divergence

$$\begin{aligned}
\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q) &= \log p(\boldsymbol{v}; \boldsymbol{\theta}) - D_{\mathrm{KL}}\left(q(\boldsymbol{h} \mid \boldsymbol{v}) \| p(\boldsymbol{h} \mid \boldsymbol{v}; \boldsymbol{\theta})\right) \\
&= \log p(\boldsymbol{v}; \boldsymbol{\theta}) - \mathrm{E}_{\mathrm{h\sim q}} \log \frac{q(\boldsymbol{h} \mid \boldsymbol{v})}{p(\boldsymbol{h} \mid \boldsymbol{v})} \\
&= \log p(\boldsymbol{v}; \boldsymbol{\theta}) - \mathrm{E}_{\mathrm{h\sim q}} \log \frac{q(\boldsymbol{h} \mid \boldsymbol{v})}{\frac{\mathrm{p(h,v;\theta)}}{\mathrm{p(v;\theta)}}} \\
&= \log p(\boldsymbol{v}; \boldsymbol{\theta}) - \mathrm{E}_{\mathrm{h\sim q}} \left[\log q(\boldsymbol{h} \mid \boldsymbol{v}) - \log p(\boldsymbol{h}, \boldsymbol{v}; \boldsymbol{\theta}) + \log p(\boldsymbol{v}; \boldsymbol{\theta})\right] \\
&= -\mathrm{E}_{\mathrm{h\sim q}} \left[\log q(\boldsymbol{h} \mid \boldsymbol{v}) - \log p(\boldsymbol{h}, \boldsymbol{v}; \boldsymbol{\theta})\right]
\end{aligned}$$

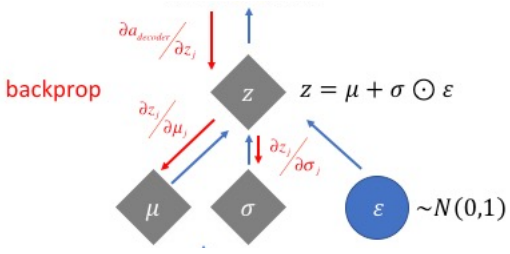
maximize ⟶ $\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q) = \mathrm{E}_{\mathrm{h\sim q}} \left[\log p(\boldsymbol{h}, \boldsymbol{v})\right] + H(q)$

Differentiable Generator Networks

Basic idea: parametrized functions for generating samples

A differentiable generator network is

- a differentiable function $g(\boldsymbol{z}; \boldsymbol{\theta}^{(g)})$ transforming samples of latent variables $\boldsymbol{z}$ to
  - samples $\boldsymbol{x}$ (direct) or
  - to distributions over samples $\boldsymbol{x}$ (indirect)
- typically represented by a (deep) neural network
  - the architecture provides the family of possible distributions to sample from
  - the parameters select a distribution from within that family
- they allow the definiton of a distribution $p_g(\boldsymbol{x})$ and SGD optimization of criteria defined on $p_g(\boldsymbol{x})$ via the reparametrization trick:
  - ∇ of stochastic transformations of $\boldsymbol{x}$?
  - augment the neural network with extra inputs $\mathcal{E}$ that are sampled from some simple probability distribution



Differentiable Generator Networks: Examples

- Example of direct sample generation
  - Drawing samples $\boldsymbol{x}$ from a normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
    - draw samples $\boldsymbol{z}$ from a normal distribution with *zero* mean and *identity* covariance
    - feed $\boldsymbol{z}$ to a simple generator network $g(\cdot)$ made of a single affine layer:

      $$\boldsymbol{x} = g(\boldsymbol{z}) = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{z},$$

      where $\boldsymbol{L}\boldsymbol{L}^\top = \boldsymbol{\Sigma}$ (Cholesky decomposition of $\boldsymbol{\Sigma}$)
- Example of indirect sample generation (define distribution over samples)
  - use $g(\cdot)$ with sigmoid outputs to provide the mean parameters of Bernoulli distributions:

    $$p(x_i = 1|\boldsymbol{z}) = g(\boldsymbol{z})_i$$

  - when using $g(\cdot)$ to define $p(\boldsymbol{x}|\boldsymbol{z})$, it is possible to impose a distribution over $\boldsymbol{x}$ by marginalizing $\boldsymbol{z}$:

    $$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}) = \sum_{\boldsymbol{z}} \underbrace{p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})}_{product\ rule} = \mathbb{E}_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z})$$

Variational Autoencoders

Basic idea:
assume training data $\mathcal{T} = \{\boldsymbol{x}^{(i)}\}_{i=1}^n$ is generated by latent variables $\boldsymbol{z}$

sample from true conditional $p_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}|\boldsymbol{z}^{(i)})$ $\boldsymbol{x}$

↑ decoder network

sample from true prior $p_{\hat{\boldsymbol{\theta}}}(\boldsymbol{z})$ $\boldsymbol{z}$

Goal: estimate the true parameters $\hat{\boldsymbol{\theta}}$ of this generative model

Solution:

- pick a simple distribution for prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, e.g. Gaussian
- use a generator network for the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$
- ...but need to maximize likelihood of $p_{\boldsymbol{\theta}}(\boldsymbol{x})$:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z} \quad \longrightarrow \quad \text{intractable!!}$$

Variational Autoencoders

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z} \quad \longrightarrow \quad \text{intractable!!}$$

Also the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) / p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is intractable due to $p_{\boldsymbol{\theta}}(\boldsymbol{x})$

Solution:
In addition to decoder network modeling $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, define additional encoder network $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ that approximates $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ using... ELBO!

$$\begin{aligned}
\mathcal{L}(q) &= \mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{z}, \boldsymbol{x}) + \mathcal{H}(q(\mathbf{z} \mid \boldsymbol{x})) \\
&= \mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{z}) - D_{\text{KL}}(q(\mathbf{z} \mid \boldsymbol{x}) || p_{\text{model}}(\mathbf{z})) \\
&\leq \log p_{\text{model}}(\boldsymbol{x})
\end{aligned}$$

decoder

encoder

forcing the encoder to be a Gaussian distribution (has closed-form solution)

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

Training: Maximize lower bound

Variational Autoencoders

$$\log p_{\text{model}}(\boldsymbol{z}, \boldsymbol{x}) = \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{z}) + \log p_{\text{model}}(\mathbf{z})$$

$$\mathcal{H}(q(\mathbf{z} \mid \boldsymbol{x})) = -\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} \log q(\mathbf{z} \mid \boldsymbol{x})$$

$$\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{z}, \boldsymbol{x}) + \mathcal{H}(q(\mathbf{z} \mid \boldsymbol{x}))$$

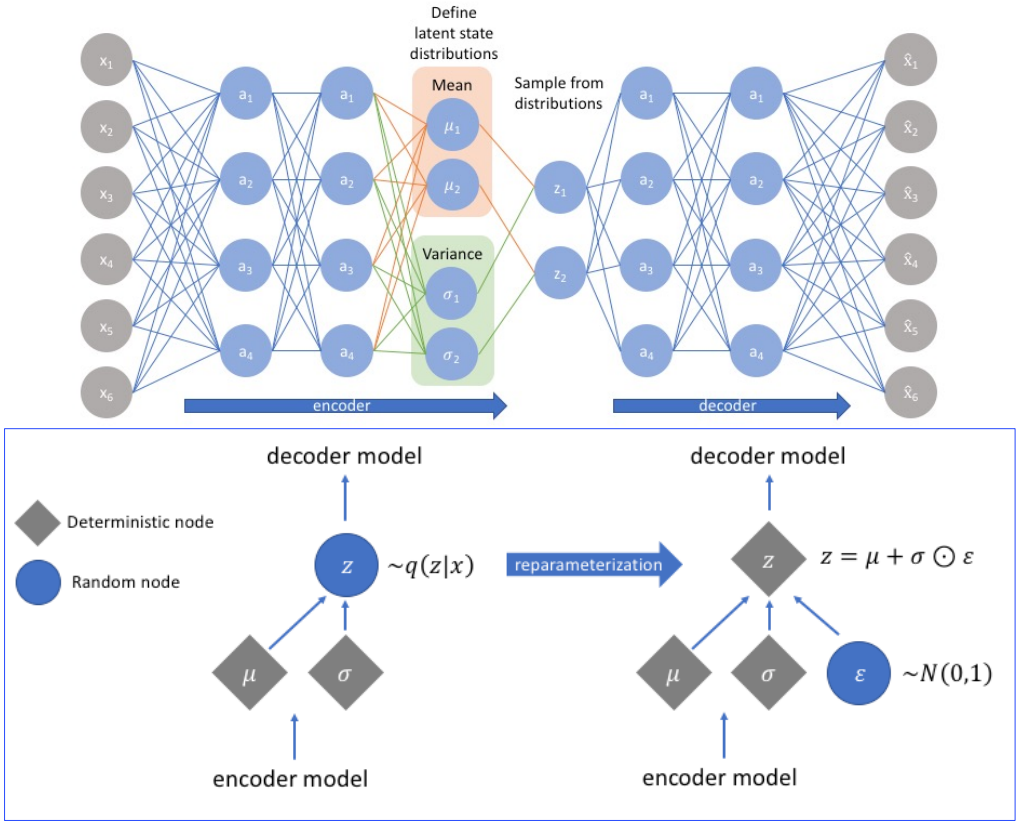
$$=$$

$$\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} [\log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{z}) + \log p_{\text{model}}(\mathbf{z})] - \mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} \log q(\mathbf{z} \mid \boldsymbol{x})$$

$$=$$

$$\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{z}) - \underbrace{\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} [\log q(\mathbf{z} \mid \boldsymbol{x}) - \log p_{\text{model}}(\mathbf{z})]}_{D_{\text{KL}}(q(\mathbf{z} \mid \boldsymbol{x}) || p_{\text{model}}(\mathbf{z}))}$$

Encoding/Decoding implementation

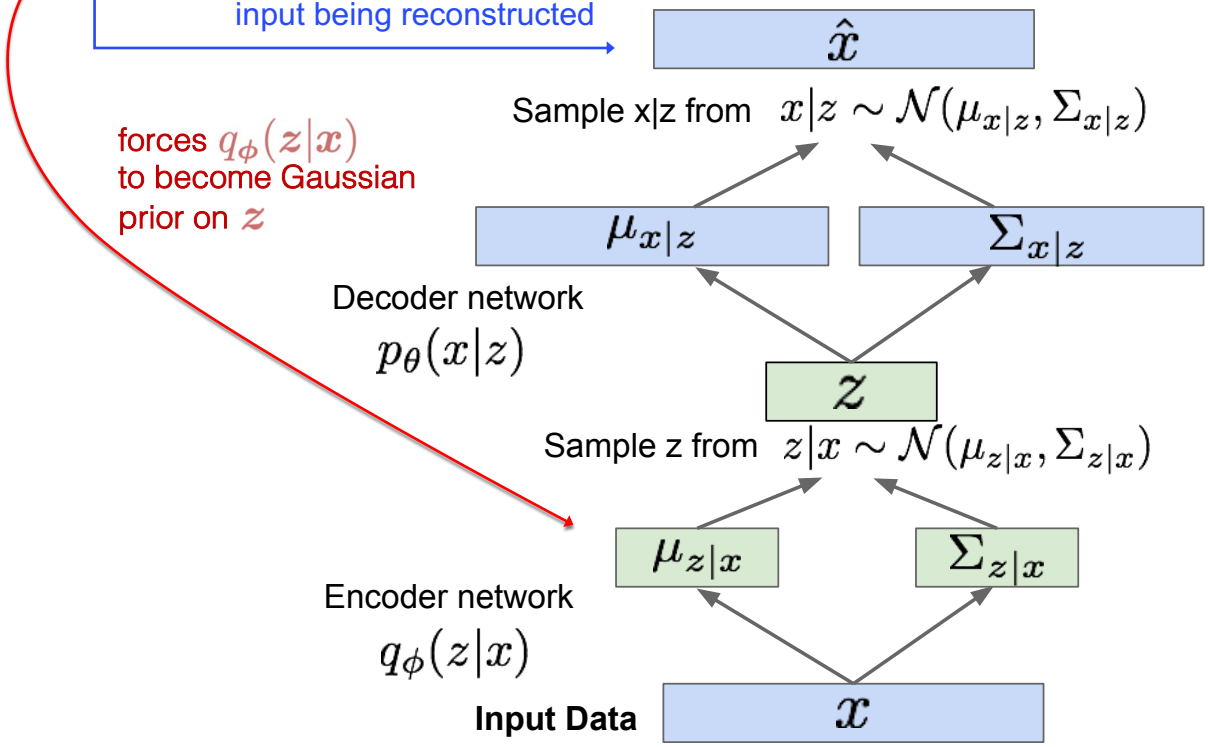


Training

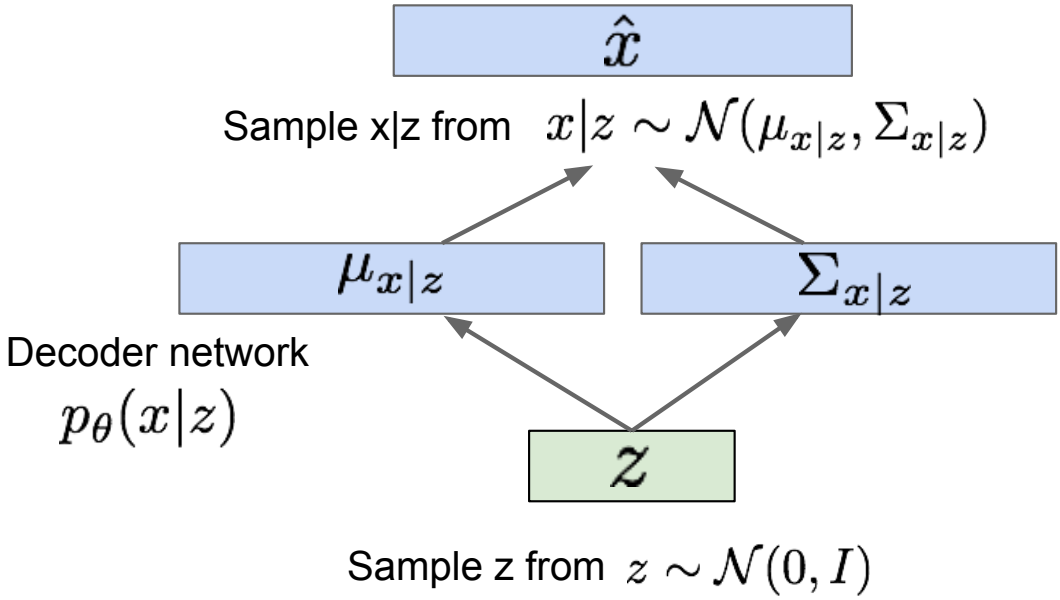
$$\mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{z}) - D_{\text{KL}}(q(\mathbf{z} \mid \boldsymbol{x}) || p_{\text{model}}(\mathbf{z}))$$

Maximize likelihood of original input being reconstructed

forces $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ to become Gaussian prior on $\boldsymbol{z}$

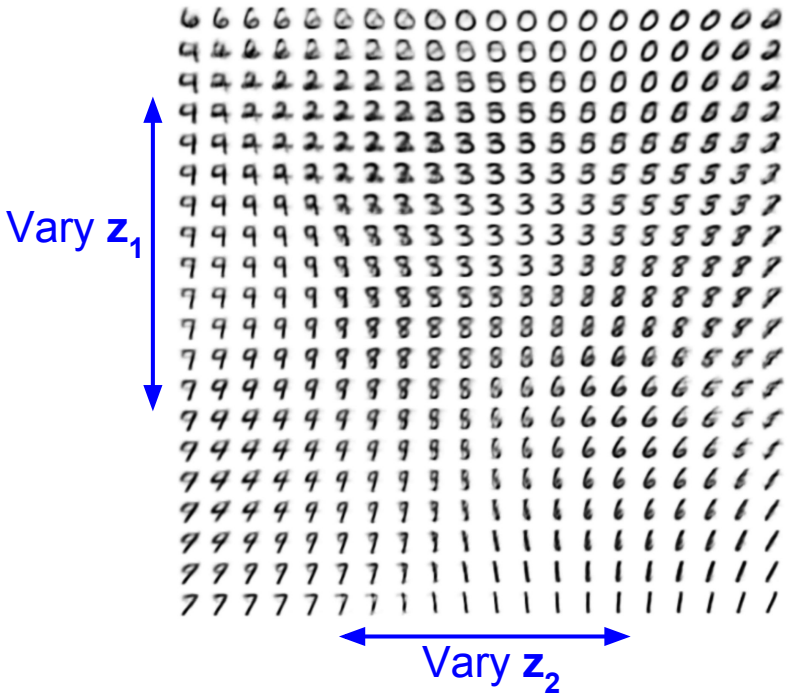


Data Generation



Example of Data Generation

Data manifold for 2-d **z**



Application to Generation of Sequences: Draw

Iterative construction of images by a spatial attention mechanism

- sequential auto-encoding framework where both the encoder and decoder are RNN
- strictly coupled encoder-decoder $\rightarrow$ encoder adapts to decoder output
- image generated incrementally by the decoder (as opposed to emitting it in a single step)
- dynamically updated attention mechanism: at each time-step "where to read", "where to write", "what to write"

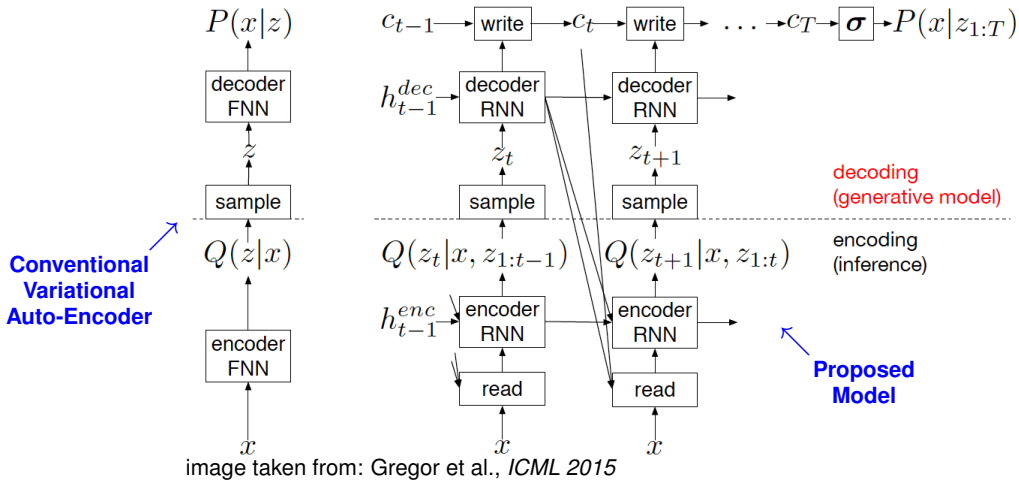


image taken from: Gregor et al., *ICML 2015*

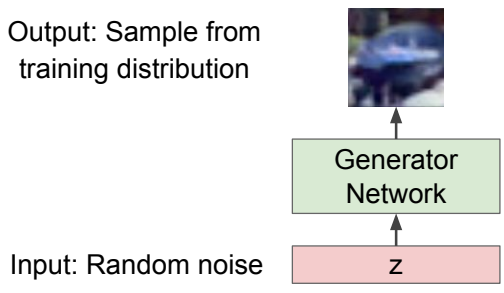
Application to Generation of Sequences: Draw

Generative Adversarial Networks (GANs) (chapter 20)

Generative Adversarial Networks

Unlike a VAE, a Generative Adversarial Network does not use an explicit density function, instead learns from the training data to draw samples

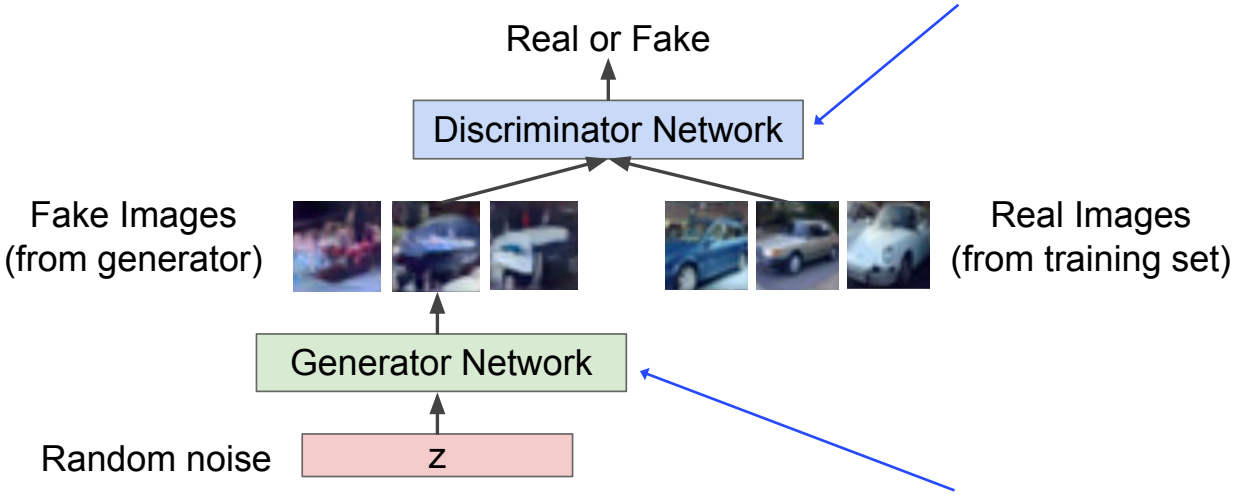
- Game-theoretic approach: learn to generate from training distribution through 2-player game
- Sample from a simple distribution, e.g. random noise, and learn transformation (generator network) to training distribution
- Transformation should be complex enough: a (deep) neural network as generator network!



Generative Adversarial Networks: Basic Idea

Instance of minimax game

The task of the Discriminator Network is to try to distinguish between real images (target = 1) and generated real-looking images (target = 0)



The task of the Generator Network is to try to fool the discriminator by generating real-looking images

Objective Function

Train jointly in **minimax game**

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

Minimax objective function:

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

For G fixed, the optimal discriminator D is

$$D^*_G(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right] + \mathbb{E}_{\boldsymbol{x} \sim p_g} \left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right]$$

Training Procedure

Training procedure

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

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

Convergence *If G and D have enough capacity, and at each step the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D^*_G(\boldsymbol{x}))]$$

then p_g converges to p_{data}

Training Procedure

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.

Training Procedure Modification

Training procedure

Alternate between:

1. **Gradient ascent** on discriminator

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

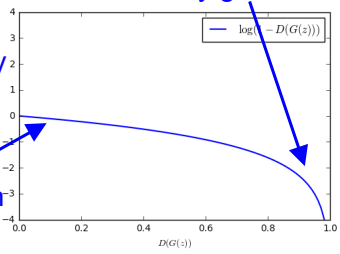
2. **Gradient descent** on generator

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

Problem: does not work well in practice!

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 Procedure Modification

Training procedure

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

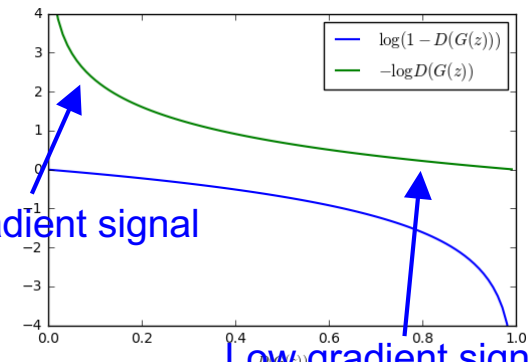
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Change with

Gradient ascent

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

maximize log-likelihood of discriminator being wrong



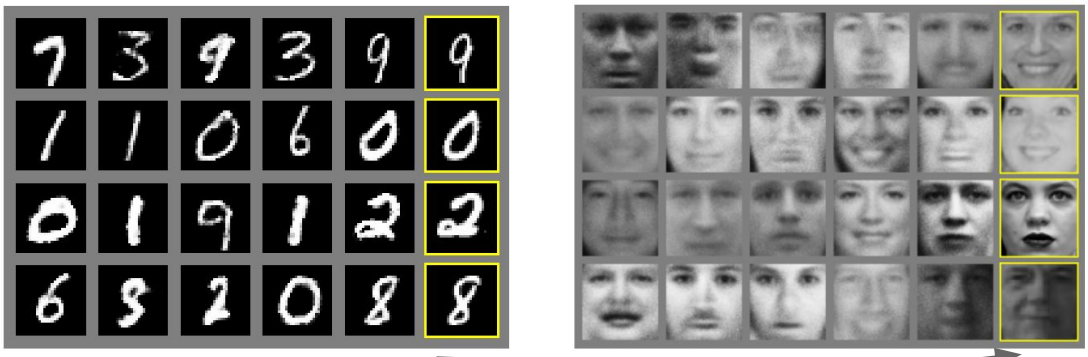
High gradient signal

Low gradient signal

Generative Adversarial Networks: Examples

After training, use generator network to generate new images

Generated samples



Nearest neighbor from training set

Figures copyright Ian Goodfellow et al., 2014.

VAEs vs GANs

- Variational Autoencoders (VAE)
  - Optimize variational lower bound on likelihood
  - Useful latent representation that allows for inference queries
  - Usually sample quality is not the best
- Generative Adversarial Networks (GANs)
  - Game-theoretic approach
  - Best samples!
  - Can be tricky and unstable to train, no inference queries