Deep Generative Models

CS772A: Probabilistic Machine Learning

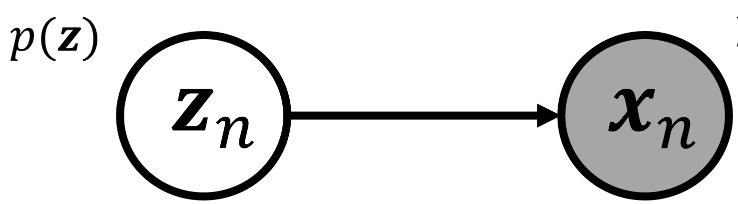
Piyush Rai

Plan

- Variational Autoencoders
- Generative Adversarial Networks
- Denoising Diffusion Models

Constructing Generative Models using Neural Nets

- We can use a neural net to define the mapping from a K-dim $\boldsymbol{z}_n$ to D-dim $\boldsymbol{x}_n$



$p(\boldsymbol{x}|\text{NN}(\boldsymbol{z};\mathbf{W}))$

Another alternative is to use a GP instead of a neural net

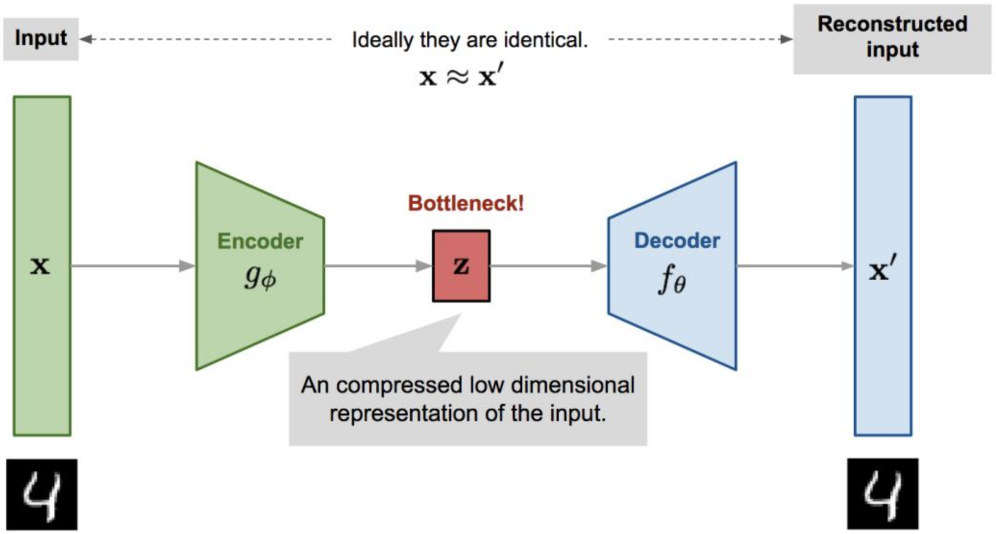
$$\mathbf{x}_n \sim \mathcal{N}(\text{NN}(\mathbf{z}_n;\mathbf{W}),\sigma^2\mathbf{I}_D)$$
$$\mathbf{x}_n \sim \text{ExpFam}(\text{NN}(\mathbf{z}_n;\mathbf{W}))$$

- If $\boldsymbol{z}_n$ has a Gaussian prior, such models are called deep latent Gaussian models (DLGM)
- Since NN mapping can be very powerful, DLGM can generate very high-quality data
  - Take the trained network, generate a random $\boldsymbol{z}$ from prior, pass it through the model to generate $\boldsymbol{x}$

Some sample images generated by Vector Quantized Variational Auto-Encoder (VQ-VAE), a state-of-the-art DLGM

Variational Autoencoder (VAE)

- VAE* is a probabilistic extension of autoencoders (AE)



$$L_{\mathrm{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$$

- The basic difference is that VAE assumes a prior $\boldsymbol{p(z)}$ on the latent code $\boldsymbol{z}$
  - This enables it to not just compress the data but also generate synthetic data
  - How: Sample $\boldsymbol{z}$ from a prior and pass it through the decoder
- Thus VAE can learn good latent representation + generate novel synthetic data
- The name has "Variational" in it since it is learned using VI principles

*Autoencoding Variational Bayes (Kingma and Welling, 2013)

Pic source: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

Variational Autoencoder (VAE)

- VAE has three main components
  - A prior $p_\theta(z)$ over latent codes
  - A probabilistic decoder/generator $p_\theta(x|z)$, modeled by a deep neural net
  - A posterior or probabilistic encoder $p_\theta(z|x)$ approx. by an "inference network" $q_\phi(z|x)$

Here θ collectively denotes all the parameters of the prior and likelihood

Using the idea of "Amortized Inference" (next slide)

Here ϕ collectively denotes all the parameters that define the inference network

- VAE is learned by maximizing the ELBO

ELBO for a single data point

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right]$$
$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - \mathbb{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})\right)$$

Maximized to find the optimal θ and ϕ

q_ϕ should be such that data x is reconstruct well from z (high log-lik)

q_ϕ should also be simple (close to the prior)

- The Reparametrization Trick is commonly used to optimize the ELBO
- Posterior is inferred only over z, and usually only point estimate on θ and ϕ

Amortized Inference

- Latent variable models need to infer the posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ for each observation $\boldsymbol{x}_n$

- This can be slow if we have lots of observations because
  1. We need to iterate over each $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$
  2. Learning the global parameters needs wait for step 1 to finish for all observations

- One way to address this is via Stochastic VI
- Amortized inference is another appealing alternative (used in VAE and other LVMs too)

$$p(\boldsymbol{z}_n|\boldsymbol{x}_n) \approx q(\boldsymbol{z}_n|\phi_n) = q(\boldsymbol{z}_n|\mathrm{NN}(\boldsymbol{x}_n;\boldsymbol{W}))$$

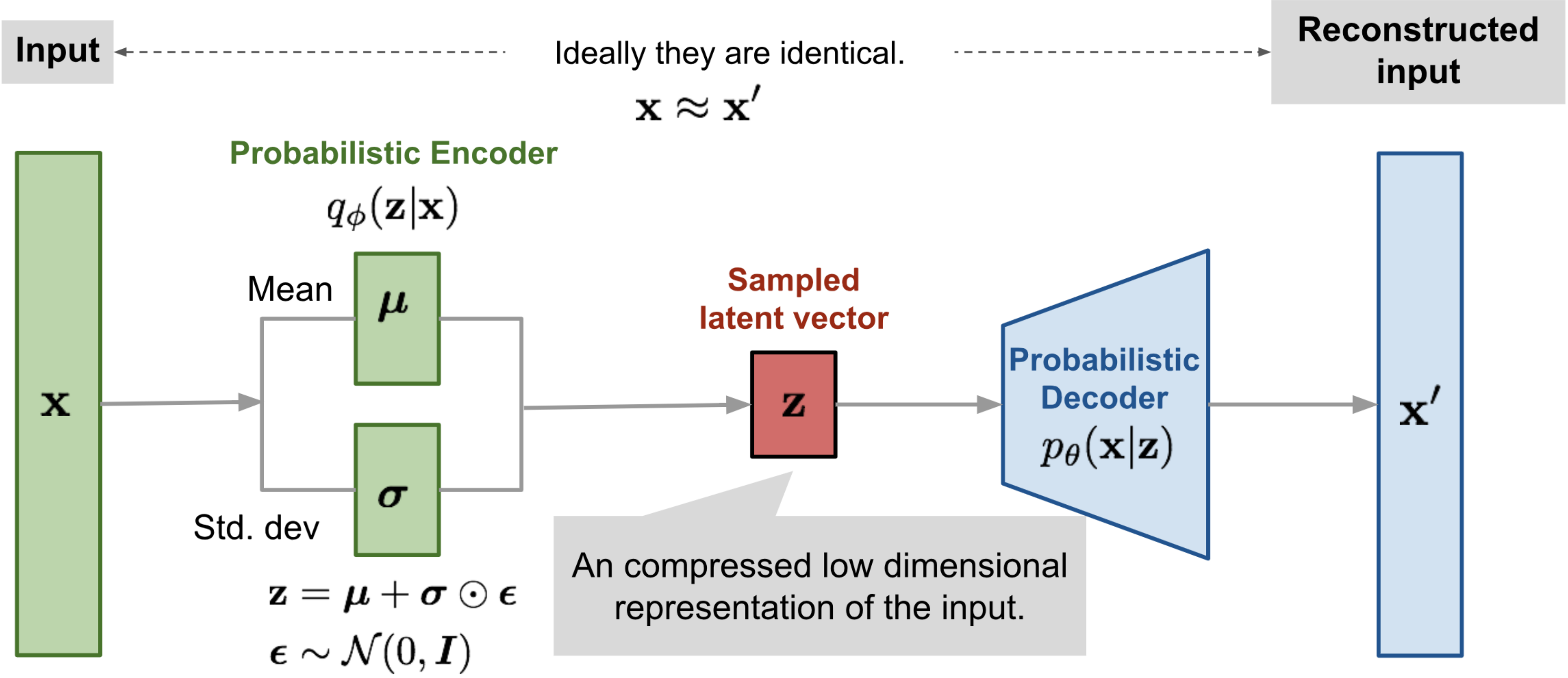
If $\boldsymbol{q}$ is Gaussian then the NN will output a mean and a variance

- Thus no need to learn $\boldsymbol{\phi}_n$'s (one per data point) but just a single NN with params $\boldsymbol{W}$
  - This will be our "encoder network" for learning $\boldsymbol{z}_n$
  - Also very efficient to get $p(\boldsymbol{z}_*|\boldsymbol{x}_*)$ for a new data point $\boldsymbol{x}_*$

Variational Autoencoder: The Complete Pipeline

- Both probabilistic encoder and decoder learned jointly by maximizing the ELBO

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})]$$
$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - \mathbb{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})\right)$$



Pic source: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

VAE and Posterior Collapse

Decoder is a neural net and can be arbitrarily powerful making this term very large

Consequently, KL will become close to zero collapsing posterior to the prior

- VAEs may suffer from posterior collapse

$$\mathcal{L}(\boldsymbol{\theta}, \phi|\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - \mathbb{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})\right)$$

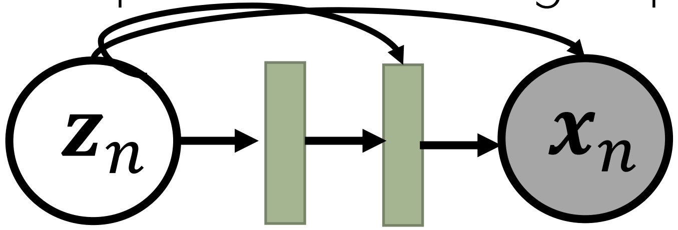
- Thus, due to posterior collapse, reconstruction will still be good but the code $\boldsymbol{z}$ may be garbage (not useful as a representation for $\boldsymbol{x}$)
- Several ways to prevent posterior collapse, e.g.,
  - Use KL annealing

A carefully tuned value between 0 and 1

$$\mathcal{L}(\boldsymbol{\theta}, \phi|\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - \beta\,\mathbb{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})\right)$$

For example, keep the variance of $\boldsymbol{q}$ as fixed

  - Avoid KL from becoming 0 using some $\boldsymbol{q}$ that doesn't collapse to the prior
  - More tightly couple $\boldsymbol{z}$ with $\boldsymbol{x}$ using skip-connections (Skip-VAE)



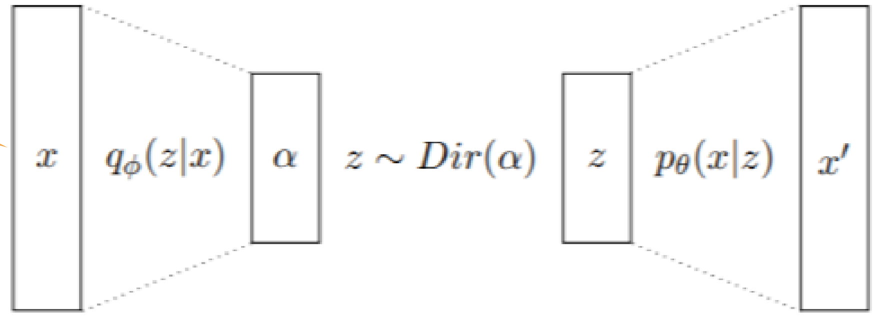
Hidden layers of NN

Besides these, MCMC (sometimes used for inference in VAE), or improved VI techniques can also help in preventing posterior collapse in VAEs

VAE: Some Comments

- One of the state-of-the-art latent variable models
- Useful for both generation as well as representation learning
- Many improvements and extensions, e.g.,
  - For text data and sequences (VAE for topic models or "neural topic models")

Document (e.g., as a vector of word counts)



(a) Dirichlet VAE (DVAE)

  - VAE-style models with more than one layer of latent variables (Sigmoid Belief Networks, hierarchical VAE, Ladder VAE, Deep Exponential Families, etc)

Decoupling Sparsity and Smoothness in the Dirichlet Variational Autoencoder Topic Model (Burkhardt and Kramer, 2020)

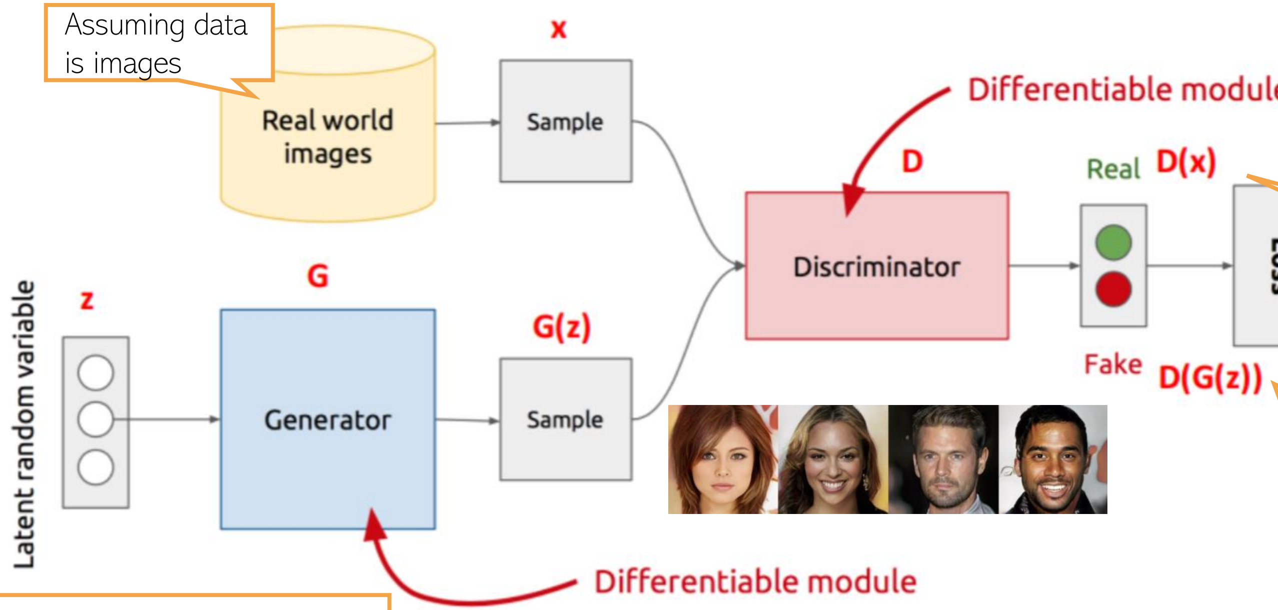
Generative Adversarial Network (GAN)

Unlike VAE, no explicit parametric likelihood model $p(x|z)$

- GAN is an implicit generative latent variable model
- Can generate from it but can't compute $p(x)$ - the model doesn't define it explicitly
- GAN is training using an adversarial way (Goodfellow et al, 2013)

Thus can't train using methods that require likelihood (MLE, VI, etc)

Assuming data is images



The discriminator can be a binary classifier or any method that can compare b/w two distributions (real and fake here)

Discriminator network is trained to make $D(x)$ close to 1

Discriminator network is trained to make $D(G(z))$ close to 0 and generator network is trained to make it to be close to 1 to fool the discriminator into believing that $G(z)$ is a real sample

Min-max optimization

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

Generative Adversarial Network (GAN)

- The GAN training criterion was

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

- With $\boldsymbol{G}$ fixed, the optimal $\boldsymbol{D}$ (exercise)

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Distribution of real data

Distribution of synthetic data

- Given the optimal $\boldsymbol{D}$, The optimal generator $\boldsymbol{G}$ is found by minimizing

$$V(D_G^*, G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)}\right]$$

$$= \mathrm{KL}\left[p_{data}(x) \middle\| \frac{p_{data}(x) + p_g(x)}{2}\right] + \mathrm{KL}\left[p_g(x) \middle\| \frac{p_{data}(x) + p_g(x)}{2}\right] - \log 4$$

Jensen-Shannon divergence between p_{data} and p_g. Minimized when $p_g = p_{data}$

Thus GAN can learn the true data distribution if the generator and discriminator have enough modeling power

GAN Optimization

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

- The GAN training procedure can be summarized as

θ_g and θ_d denote the params of the deep neural nets defining the generator and discriminator, respectively

Initialize $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_d$;
for *each training iteration* **do**
  for K *steps* **do**
    Sample minibatch of M noise vectors $\mathbf{z}_m \sim q_z(\mathbf{z})$;
    Sample minibatch of M examples $\mathbf{x}_m \sim p_D$;
    Update the discriminator by performing stochastic gradient *ascent* using this gradient:
    $\nabla_{\boldsymbol{\theta}_d} \frac{1}{M} \sum_{m=1}^{M} [\log D(\mathbf{x}_m) + \log(1 - D(G(\mathbf{z}_m)))]$. ;
  Sample minibatch of M noise vectors $\mathbf{z}_m \sim q_z(\mathbf{z})$;
  Update the generator by performing stochastic gradient *descent* using this gradient:
  $\nabla_{\boldsymbol{\theta}_g} \frac{1}{M} \sum_{m=1}^{M} \log(1 - D(G(\mathbf{z}_m)))$. ;
Return $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_d$

In practice, for stable training, we run $K > 1$ steps of optimizing w.r.t. D and 1 step of optimizing w.r.t. G

In practice, in this step, instead of minimizing $\log(1 - D(G(z))$, we maximize $\log\left(D(G(z))\right)$

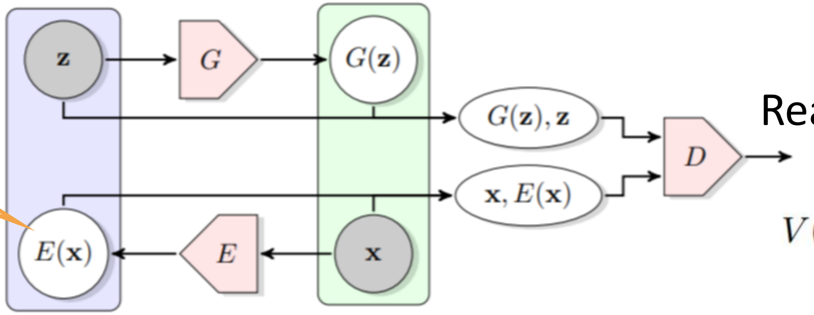
Reason: Generator is bad initially so discriminator will always predict correctly initially and $\log(1 - D(G(z))$ will saturate

GANs that also learn latent representations

- The standard GAN can only generate data. Can't learn the latent $\boldsymbol{z}$ from $\boldsymbol{x}$
- Bidirectional GAN* (BiGAN) is a GAN variant that allows this

Consists of an encoder as well

Can be shown* to "invert" $\boldsymbol{G}$



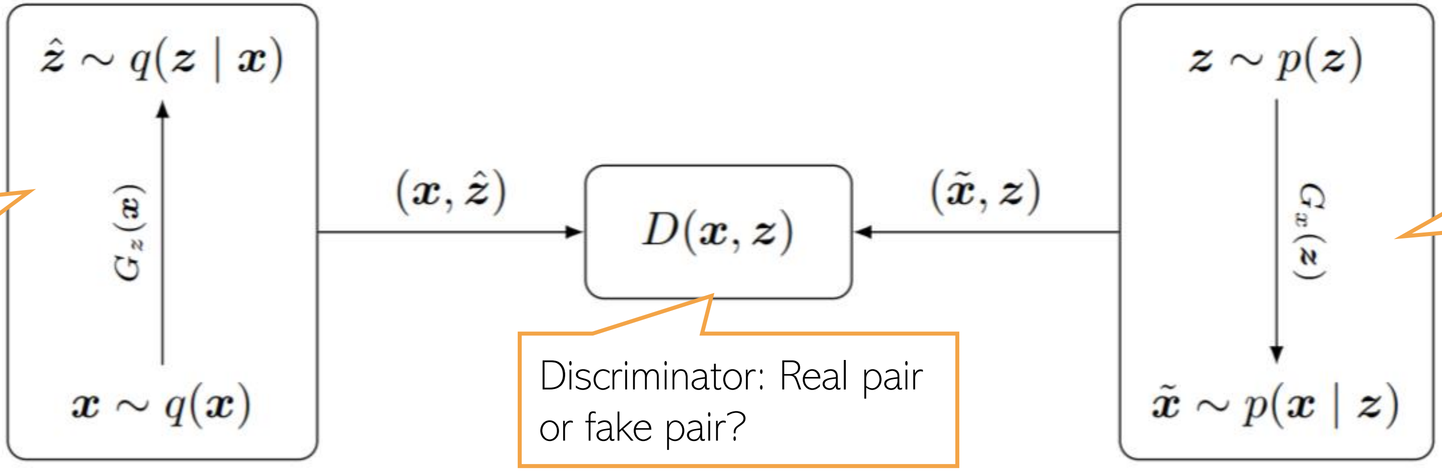
Real pair/fake pair?

$$\min_{G,E} \max_{D} V(D,E,G)$$

$$V(D,E,G) := \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{X}}}\Big[\underbrace{\mathbb{E}_{\mathbf{z}\sim p_E(\cdot|\mathbf{x})}\left[\log D(\mathbf{x},\mathbf{z})\right]}_{\log D(\mathbf{x},E(\mathbf{x}))}\Big] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{Z}}}\Big[\underbrace{\mathbb{E}_{\mathbf{x}\sim p_G(\cdot|\mathbf{z})}\left[\log\left(1-D(\mathbf{x},\mathbf{z})\right)\right]}_{\log(1-D(G(\mathbf{z}),\mathbf{z}))}\Big]$$

- Adversarially Learned Inference# (ALI) is another variant that can learn representations

Encoder with joint $q(x,z) = q(x)q(z|x)$



Decoder/generator with joint $p(x,z) = p(z)p(x|z)$

Discriminator: Real pair or fake pair?

*Adversarial Feature Learning (Donahue et a Dumoulin l, 2017) #Adversarially Learned Inference (Dumoulin et al, 2017)

Evaluating GANs

High IS and low FID is desirable

Both IS and FID measure how realistic the generated data is

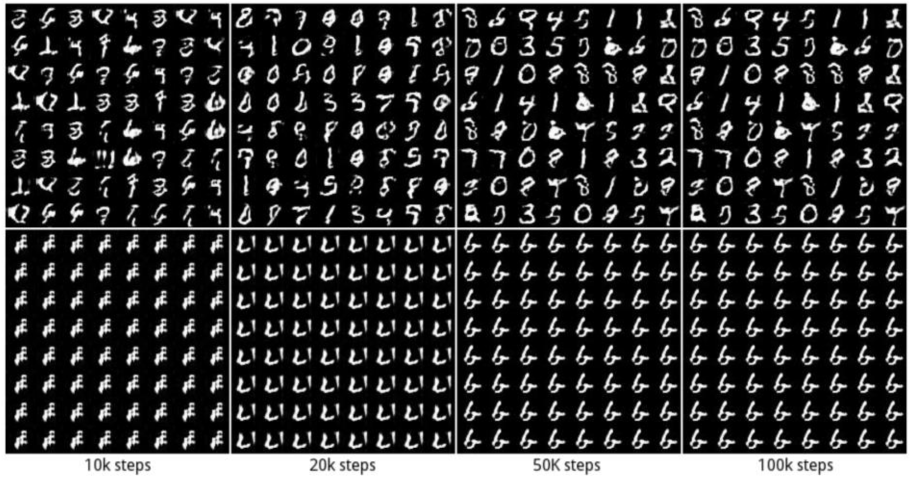
- Two measures that are commonly used to evaluate GANs
  - Inception score (IS): Evaluates the distribution of generated data
  - Frechet inception distance (FID): Compared the distribution of real data and generated data
- Inception Score defined as $\exp(\mathbb{E}_{x\sim p_g}[\text{KL}(p(y|x)||p(y))])$ will be high if
  - Very few high-probability classes in each sample x: Low entropy for $p(y|x)$
  - We have diverse classes across samples: Marginal $p(y)$ is close to uniform (high entropy)
- FID uses extracted features (using a deep neural net) of real and generated data
  - Usually from the layers closer to the output layer
- These features are used to estimate two Gaussian distributions

Using real data $\mathcal{N}(\mu_R, \Sigma_\text{R})$ $\quad\quad$ $\mathcal{N}(\mu_G, \Sigma_\text{G})$ Using generated data

- FID is then defined as $\textbf{FID} = |\mu_G - \mu_R|^2 + \ \textbf{trace}(\Sigma_G + \Sigma_R - (\Sigma_G\Sigma_R)^{1/2})$

GAN: Some Issues/Comments

- GAN training can be hard and the basic GAN suffers from several issues
- Instability of training procedure
- Mode Collapse problem: Lack of diversity in generated samples
  - Generator may find some data that can easily fool the discriminator
  - It will stuck at that mode of the data distribution and keep generating data like that



GAN 1: No mode collapse (all 10 modes captured in generation)

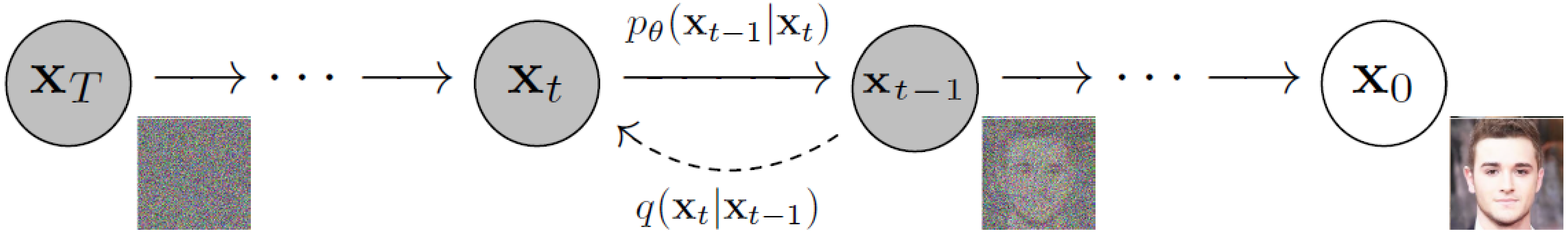
GAN 2: Mode collapse (stuck on one of the modes)

- Some work on addressing these issues (e.g., Wasserstein GAN, Least Squares GAN, etc)

Denoising Diffusion Models

After learning the model, can use the reverse process to generate data from random noise

- Based on a forward (adding noise) process and a reverse (denoising) process



- Steps of the forward process are defined by a fixed Gaussian $q(x_t|x_{t-1})$
  - The f.p. starts with the clean image x_0 and adds zero-mean Gaussian noise at each step
  - The f.p. distribution is defined as $q(x_t|x_{t-1}) = \mathcal{N}(x_t|\sqrt{(1-\beta_t)}x_{t-1}, \beta_t I)$ — $\beta_t \in (0,1)$
  - Eventually as $T \to \infty$, we get x_T which is isotropic Gaussian noise
  - Can show: $q(x_t|x_0) = \mathcal{N}(x_t|\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$ where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$
- Steps of the reverse process are defined by a learnable Gaussian $p_\theta(x_{t-1}|x_t)$
  - $p_\theta(x_{t-1}|x_t)$ is an approximation of the reverse diffusion $q(x_{t-1}|x_t)$
  - $p_\theta(x_{t-1}|x_t)$ modeled as $\mathcal{N}(x_{t-1}|\mu_\theta(x_t), \Sigma_\theta(x_t))$ where μ_θ and Σ_θ are neural nets

Denoising Diffusion Models: Training

- The model is trained by minimizing the following objective

Upper bound on the negative log-likelihood (negative of the ELBO)

$$\mathbb{E}[-\log p_\theta(x_0)] \leq \mathbb{E}\left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] := \mathcal{L}$$

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := p(\mathbf{x}_T)\prod_{t=1}^{T} \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \prod_{t=1}^{T} \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

$$\mathcal{L} = L_0 + L_1 + L_2 + \cdots + L_{T-1} + L_T$$

$$L_0 = -\log p_\theta(x_0|x_1)$$

This is also a Gaussian

Overall loss is just a sum of several KL divergences between Gaussians, and thus available in closed form

$$L_{t-1} = D_{KL}(q(x_{t-1}|x_t, x_0) \,||\, p_\theta(x_{t-1}|x_t))$$

$$L_T = D_{KL}(q(x_T|x_0) \,||\, p(x_T))$$

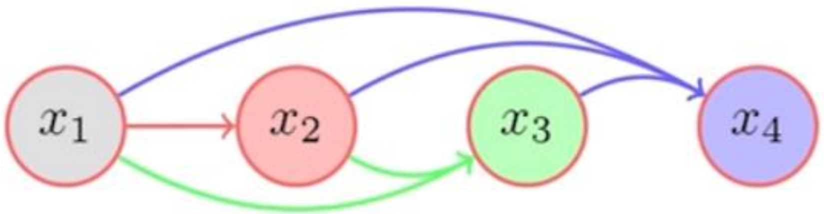
- In some ways, denoising diffusion models are similar to VAEs

Summary

- Looked at various methods for generative modeling for unsupervised learning
  - Classical methods (FA, PPCA, other latent factor models, topic models, etc)
  - Deep generative models (VAE, GAN, Denoising Diffusion Models)
- Many of these methods can also be extended to model data other than images
- There are also generative models that do not use latent variables
  - Can still be used to generate data and learn the underlying data distribution

Assuming each observation is n-dimensional

$$\prod_{i=1}^{n} p(x_i | \mathbf{x}_{<k})$$



An auto-regressive model

Can use a neural network to learn (parameters of) each of these distributions

An example: Neural Autoregressive Density Estimator (NADE)