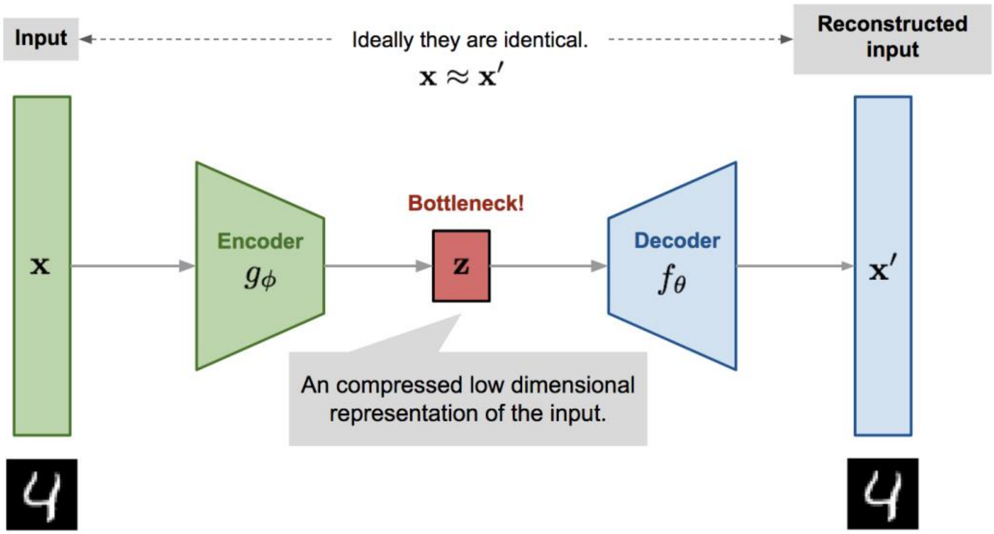
Deep Generative Models (VAE and GAN)

CS772A: Probabilistic Machine Learning

Piyush Rai

A Deep Generative Model: Variational Auto-encoder (VAE)

- VAE* is a probabilistic extension of autoencoders (AE). An AE is shown below



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

- 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

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

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

Maximized to find the optimal θ and ϕ

q_ϕ should be such that data x is reconstructed 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

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)

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

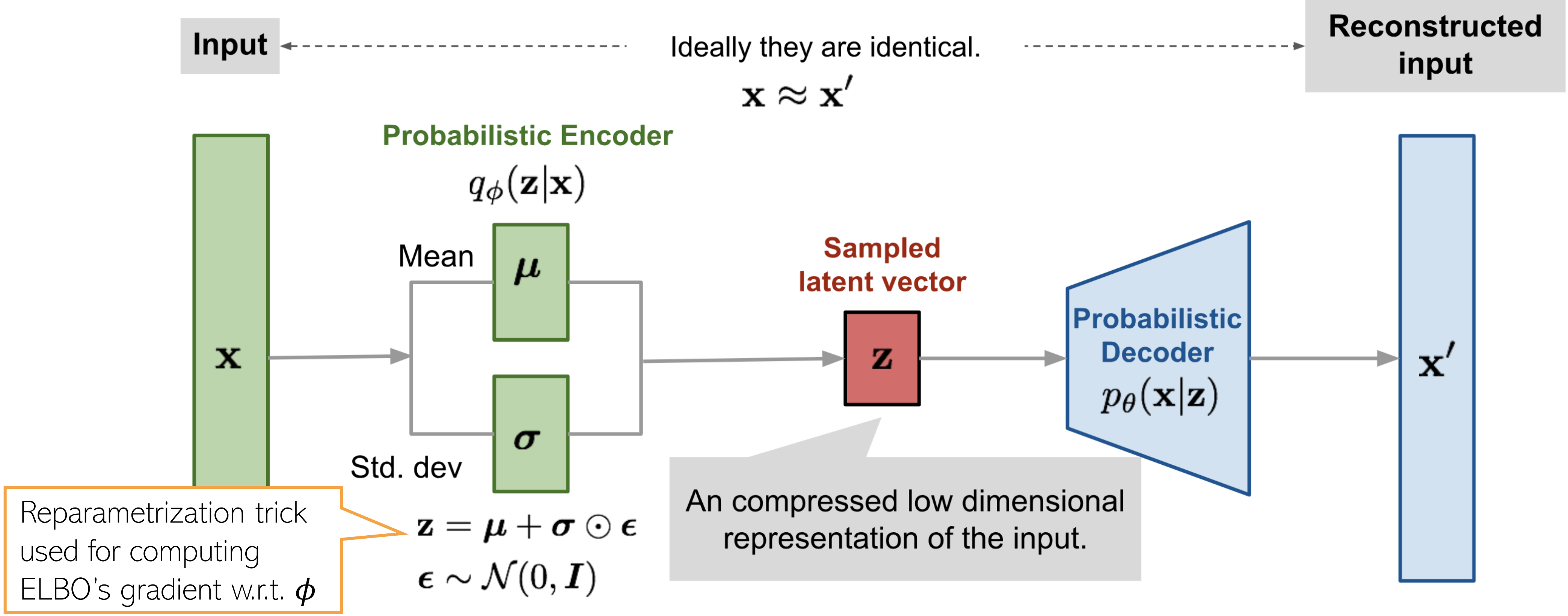
If $\boldsymbol{q}$ is Gaussian then the NN will output a mean and a variance

- 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

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



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

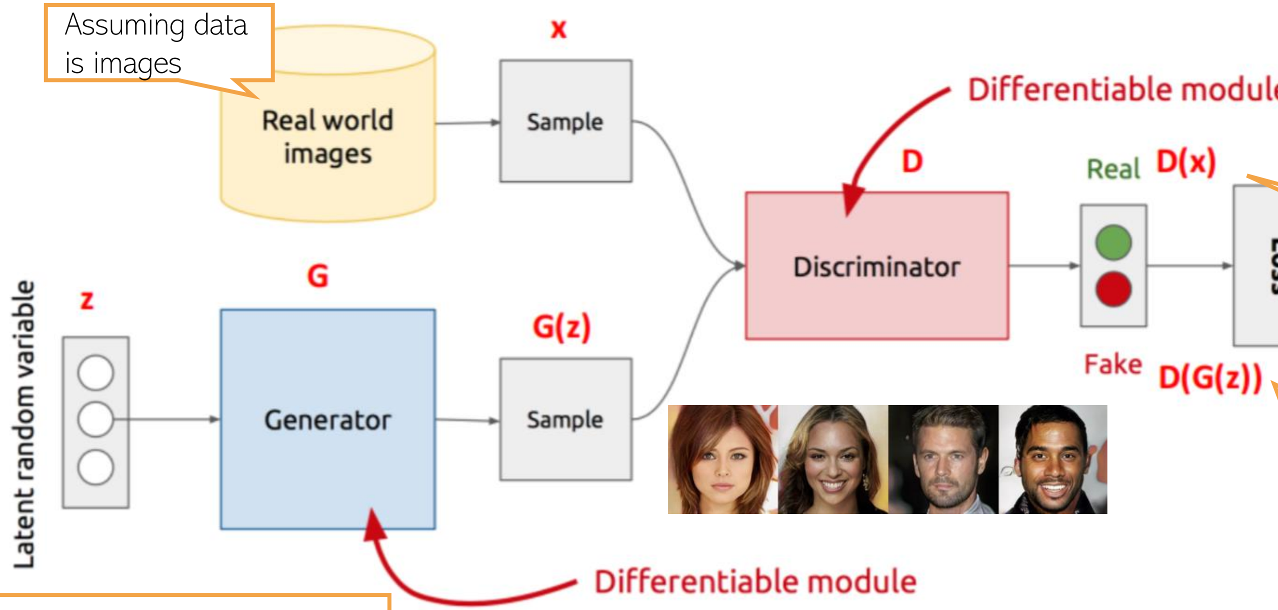
Generative Adversarial Network (GAN)

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 trained 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 G fixed, the optimal D (exercise)

Distribution of real data

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

Distribution of synthetic data

- Given the optimal D, The optimal generator G is found by minimizing

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

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

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

1 Initialize $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_d$;
2 **for** *each training iteration* **do**
3   **for** K *steps* **do**
4   Sample minibatch of M noise vectors $\mathbf{z}_m \sim q_z(\mathbf{z})$;
5   Sample minibatch of M examples $\mathbf{x}_m \sim p_D$;
6   Update the discriminator by performing stochastic gradient *ascent* using this gradient:
$\nabla_{\boldsymbol{\theta}_d} \frac{1}{M}\sum_{m=1}^{M} [\log D(\mathbf{x}_m) + \log(1 - D(G(\mathbf{z}_m)))].$;
7   Sample minibatch of M noise vectors $\mathbf{z}_m \sim q_z(\mathbf{z})$;
8   Update the generator by performing stochastic gradient *descent* using this gradient:
$\nabla_{\boldsymbol{\theta}_g} \frac{1}{M}\sum_{m=1}^{M} \log(1 - D(G(\mathbf{z}_m))).$;
9 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

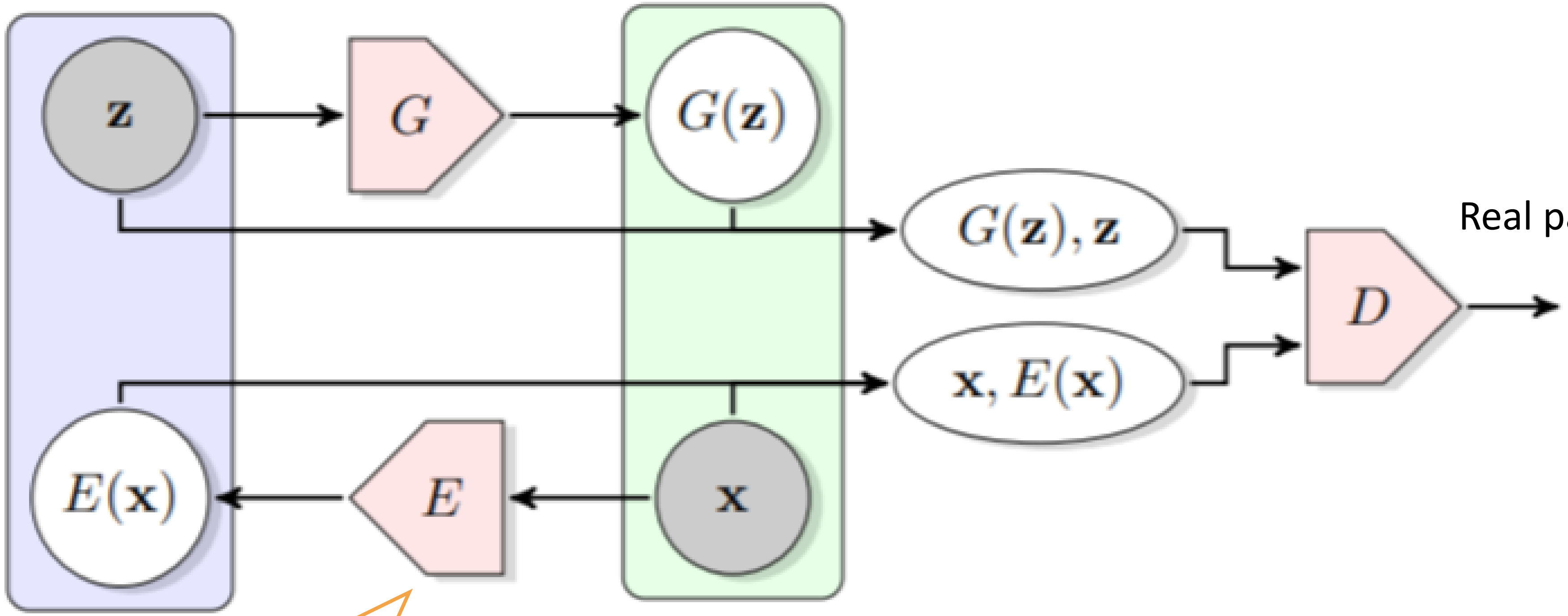
In practice, in this step, instead of minimizing $\log(1 - D(G(z))$, we maximize $\log\left(D(G(z))\right)$

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



Real pair/fake pair?

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

*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

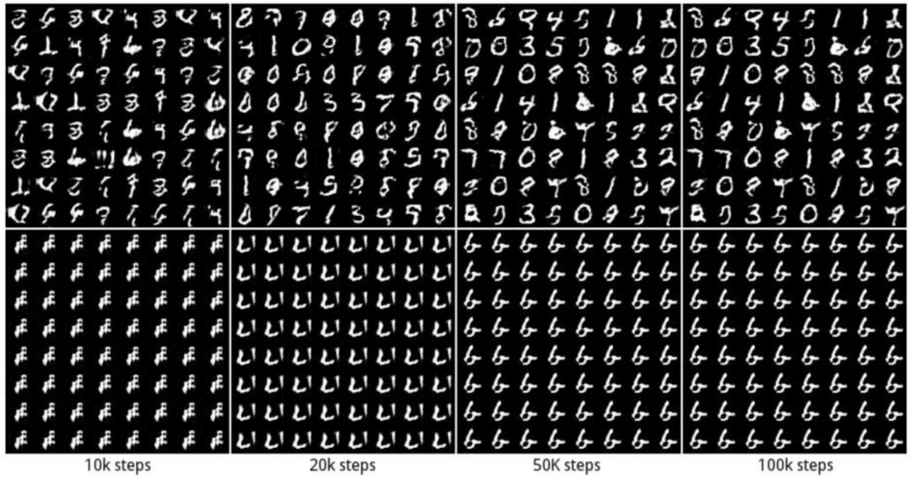
- Two measures that are commonly used to evaluate GANs
  - Inception score (IS): Evaluates the distribution of generated data
  - Frechet inception distance (FID): Compared the distribution of real data and generated data
- Inception Score defined as $\exp(\mathbb{E}_{x \sim p_g}[\mathrm{KL}(p(y|x)||p(y))])$ 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 $\mathcal{N}(\mu_R, \Sigma_R)$ $\mathcal{N}(\mu_G, \Sigma_G)$ Using generated data

- FID is then defined as $\mathbf{FID} = |\mu_G - \mu_R|^2 + \mathbf{trace}(\Sigma_G + \Sigma_R - (\Sigma_G \Sigma_R)^{1/2})$
- These measures can also be used for evaluating other deep gen models like VAE

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)