MCMC Sampling (wrap-up), Deep Generative Models

CS772A: Probabilistic Machine Learning

Piyush Rai

Hamiltonian/Hybrid Monte Carlo (HMC)

- HMC (Neal, 1996) is an "auxiliary variable sampler" and incorporates gradient info
- Uses the idea of simulating a Hamiltonian Dynamics of a physical system
- Consider the target posterior $p(\theta|\mathcal{D}) \propto \exp(-U(\theta))$
- Think of θ as "position" then $U(\theta) = -\log p(\mathcal{D}|\theta)p(\theta)$ is like "potential energy"
- Let's introduce an auxiliary variable - the momentum $\boldsymbol{r}$ of the system
- Can now define a joint distribution over the position and momentum as

$$p(\theta, \boldsymbol{r}|\mathcal{D}) \propto \exp\left(-U(\theta) - \frac{1}{2}\boldsymbol{r}^\top M^{-1}\boldsymbol{r}\right) \propto p(\theta|\mathcal{D})p(\boldsymbol{r})$$

- The total energy (potential + kinetic) or the Hamiltonian of the system

Constant w.r.t. time

$$H(\theta, \boldsymbol{r}) = U(\theta) - \frac{1}{2}\boldsymbol{r}^\top M^{-1}\boldsymbol{r} = U(\theta) + K(\boldsymbol{r})$$

- Given a sample $(\theta, \boldsymbol{r})$ from $p(\theta, \boldsymbol{r})$, ignoring $\boldsymbol{r}$, θ will be a sample from $p(\theta|\mathcal{D})$

Generating Samples in HMC

- Given an initial $(\boldsymbol{\theta}, \boldsymbol{r})$, Hamiltonian Dynamics defines how $(\boldsymbol{\theta}, \boldsymbol{r})$ changes w.r.t. time t

$$\frac{\partial \theta}{\partial t} = \frac{\partial H}{\partial \boldsymbol{r}} = \frac{\partial K}{\partial \boldsymbol{r}}$$
$$\frac{\partial \boldsymbol{r}}{\partial t} = -\frac{\partial H}{\partial \theta} = -\frac{\partial U}{\partial \theta}$$

$$\left(H(\theta, \boldsymbol{r}) = U(\theta) + \tfrac{1}{2}\boldsymbol{r}^\top M^{-1}\boldsymbol{r} = U(\theta) + K(\boldsymbol{r}) \right)$$

- We can use these equations to update $(\boldsymbol{\theta}, \boldsymbol{r}) \rightarrow (\boldsymbol{\theta}^*, \boldsymbol{r}^*)$ by discretizing time

Reason: Getting analytical solutions for the above requires integrals which is in general intractable

- For $s = 1{:}S$, sample as follows
  - Initialize $\theta_0 = \theta^{(s-1)}$, $\boldsymbol{r}_* \sim \mathcal{N}(0, \mathbf{I})$ and $\boldsymbol{r}_0 = \boldsymbol{r}_* - \frac{\rho}{2}\frac{\partial U}{\partial \theta}|_{\theta_0}$
  - Do L "leapfrog" steps with learning rates $\rho_\ell = \rho$ for $\ell < L$ and $\rho_L = \rho/2$
    - For $\ell = 1{:}L$

$$\theta_\ell = \theta_{\ell-1} + \rho \frac{\partial K}{\partial \boldsymbol{r}}|_{\boldsymbol{r}_{\ell-1}}$$
$$\boldsymbol{r}_\ell = \boldsymbol{r}_{\ell-1} - \rho_\ell \frac{\partial U}{\partial \theta}|_{\theta_\ell}$$

L usually set to 5 and learning rate tuned to make acceptance rate around 90%

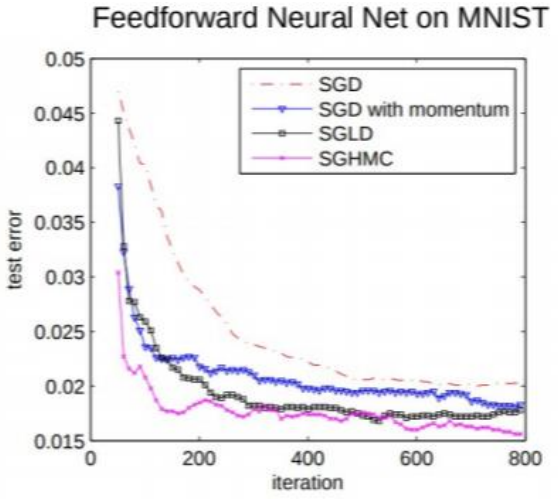
  - Perform MH accept/reject test on $(\boldsymbol{\theta}_L, \boldsymbol{r}_L)$. If accepted $\theta^{(s)} = \theta_L$

A single sample generated by taking L steps

- The momentum forces exploring different regions instead of getting driven to regions where the MAP solution is

HMC in Practice

- HMC typically has very low rejection rate (that too, primarily due to discretization error)
- Performance can be sensitive to L (no. of leapfrog steps) and step-sizes, tuning hard
- A lot of renewed interest in HMC (you may check out NUTS - No U-turn Sampler – doesn't require setting L)
  - Prob. Prog. packages e.g., Tensorflow Prob., Stan, etc, contain implementations of HMC
- Can also do HMC on minibatches (Stochastic Gradient HMC - Chen et al, 2014)
- An illustration: SGHMC vs other methods on MNIST classification (Bayesian neural net)



(Figure: Stochastic Gradient Hamiltonian Monte Carlo (Chen et al, 2014))

Parallel/Distributed MCMC

- Suppose our goal is to compute the posterior of $\theta \in \mathbb{R}^D$ (assuming N is very large)

$$p(\theta|\mathbf{X}) \propto p(\theta)p(\mathbf{X}|\theta) = p(\theta)\prod_{n=1}^{N} p(\boldsymbol{x}_n|\theta)$$

- Suppose we have J machines with data partitioned as $\mathbf{X} = \{\mathbf{X}^{(j)}\}_{j=1}^{J}$
- Let's assume that the posterior $p(\theta|\mathbf{X})$ factorizes as

$$p(\theta|\mathbf{X}) = \prod_{i=1}^{J} p^{(j)}(\theta|\mathbf{X}^{(j)})$$

- Here $p^{(j)}(\theta|\mathbf{X}^{(j)}) \propto p(\theta)^{1/J}\prod_{\boldsymbol{x}_n \in \mathbf{X}^{(j)}} p(\boldsymbol{x}_n|\theta)$ is known as the "subset posterior"
- Assume the j^{th} machine generates T MCMC samples $\{\theta_{j,t}\}_{t=1}^{T}$
- We need a way to combine these subset posteriors using a "consensus"

$$\hat{\theta}_1, \ldots, \hat{\theta}_T = \texttt{CONSENSUSSAMPLES}(\{\theta_{j,1}, \ldots, \theta_{j,T}\}_{j=1}^{J})$$

Parallel/Distributed MCMC

- Many ways to compute the consensus samples. Let's look at two of them
- Approach 1: Weighted Average: $\hat{\theta}_t = \sum_{j=1}^{J} W_j \theta_{j,t}$ where W_j can be learned as follows
  - Assuming Gaussian likelihood and Gaussian prior on θ

$$\begin{aligned} \bar{\Sigma}_j &= \text{sample covariance of } \{\theta_{j,1}, \ldots, \theta_{j,T}\} \\ \Sigma &= (\Sigma_0^{-1} + \sum_{j=1}^{J} \bar{\Sigma}_j^{-1})^{-1} \quad (\Sigma_0 \text{ is the prior's covariance}) \\ W_j &= \Sigma(\Sigma_0^{-1}/J + \bar{\Sigma}_j^{-1}) \end{aligned}$$

These approaches can also be used to make VI parallel/distributed

- Approach 2: Fit J Gaussians, one for each $\{\theta_{j,t}\}_{t=1}^{T}$ and take their product

$$\begin{aligned} \bar{\mu}_j &= \text{sample mean of } \{\theta_{j,1}, \ldots, \theta_{j,T}\}, \quad \bar{\Sigma}_j = \text{sample covariance of } \{\theta_{j,1}, \ldots, \theta_{j,T}\} \\ \hat{\Sigma}_J &= (\sum_{j=1}^{J} \bar{\Sigma}_j^{-1})^{-1}, \quad \hat{\mu}_J = \hat{\Sigma}_J(\sum_{j=1}^{J} \bar{\Sigma}_j^{-1}\bar{\mu}_j) \quad \text{(cov and mean of prod. of Gaussians)} \\ \hat{\theta}_t &\sim \mathcal{N}(\hat{\mu}_J, \hat{\Sigma}_J), t = 1, \ldots, T \quad \text{(the final consensus samples)} \end{aligned}$$

- For detailed proofs and other approaches, may refer to the reference below

Patterns of Scalable Bayesian Inference (Angelino et al, 2016)

Approximate Inference: VI vs Sampling

- VI approximates a posterior distribution $p(\boldsymbol{Z}|\boldsymbol{X})$ by another distribution $q(\boldsymbol{Z}|\boldsymbol{\phi})$
- Sampling uses $\boldsymbol{S}$ samples $\boldsymbol{Z}^{(1)}, \boldsymbol{Z}^{(2)}, \ldots, \boldsymbol{Z}^{(S)}$ to approximate $p(\boldsymbol{Z}|\boldsymbol{X})$
- Sampling can be used within VI (ELBO approx using Monte-Carlo)
- In terms of "comparison" between VI and sampling, a few things to be noted
  - Convergence: VI only has local convergence, sampling (in theory) can give exact posterior
  - Storage: Sampling based approx needs to storage all samples, VI only needs var. params $\boldsymbol{\phi}$
  - Prediction Cost: Sampling always requires Monte-Carlo avging for posterior predictive; with VI, sometimes we can get closed form posterior predictive

PPD if using sampling: $p(x_*|X) = \int p(x_*|Z)p(Z|X)dZ \approx \frac{1}{S}\sum_{s=1}^{S} p\left(x_*|Z^{(s)}\right)$

Closed form if integral is tractable (otherwise Monte Carlo avg still needed for PPD)

PPD if using VI: $p(x_*|X) = \int p(x_*|Z)p(Z|X)dZ \approx \int p(x_*|Z)q(Z|\phi)dZ$

Compressing the S samples into something more compact

- There is some work on "compressing" sampling-based approximations*

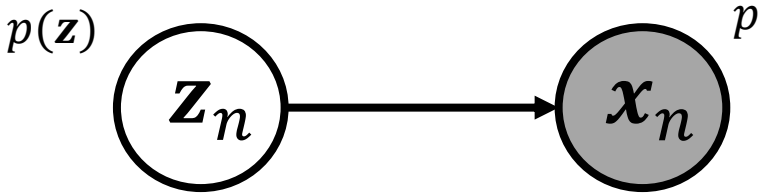
*"Compact approximations to Bayesian predictive distributions" by Snelson and Ghaharamani, 2005; and "Bayesian Dark Knowledge" by Korattikara et al, 2015

Inference Methods: Summary

- MLE/MAP: Straightforward for differentiable models (can even use automatic diff.)
- Conjugate models with one "main" parameter: Straightforward posterior updates
- MLE-II/MAP-II: Often useful for estimating the hyperparameters
- EM: If we want to do MLE/MAP for models with latent variables
  - Very general algorithm, can also be made online
  - Used when we want point estimates for some unknowns and posterior over others
  - Can use it for hyperparameter estimation as well
  - Often better than using direct gradient methods
- VI and sampling methods can be used to get full posterior for complex models
  - Quite easy if we have local conjugacy (VI has closed form updates, Gibbs sampler is easy to derive)
  - In other cases, we have general VI with Monte-Carlo gradients, MH sampling
  - MCMC can also make use of gradient info (LD/SGLD)
- For large-scale problems, online/distributed VI/MCMC, or SGD based posterior approx

Latent Variable Models for Generation Tasks

- Assume a K-dim latent variable $\boldsymbol{z}_n$ is transformed to generate to D-dim observation $\boldsymbol{x}_n$



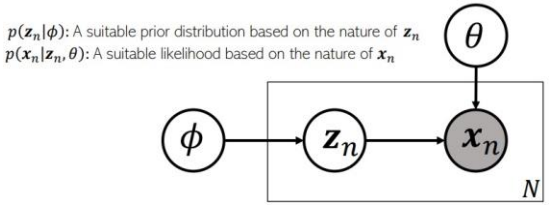
Also possible to use a GP to model f

f can be some linear or nonlinear model which defines the parameters of the distribution of $p(x|z)$ and W denotes the parameters of this model

- It is common to use a Gaussian prior for $\boldsymbol{z}_n$ (though other priors can be used)
- If we use a neural net or GP, such models 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 latent variable model for generation

Factor Analysis and Probabilistic PCA



- FA and PPCA assume f to be a linear model
- In FA/PPCA, latent variables $\boldsymbol{z}_n \in \mathbb{R}^K$ typically assumed to have a Gaussian prior
  - If we want sparse latent variabled, can use Laplace or spike-and-slab prior on $\boldsymbol{z}_n$
  - More complex extensions of FA/PPCA use a mixture of Gaussians prior on $\boldsymbol{z}_n$
- Assumption: Observations $\boldsymbol{x}_n \in \mathbb{R}^D$ typically assumed to have a Gaussian likelihood
  - Other likelihood models (e.g., exp-family) can also be used if data not real-valued
- Relationship between $\boldsymbol{z}_n$ and $\boldsymbol{x}_n$ modeled by a noisy linear mapping

$$\boldsymbol{x}_n = \boldsymbol{W}\boldsymbol{z}_n + \epsilon_n = \sum_{k=1}^{K} \boldsymbol{w}_k z_{nk} + \epsilon_n$$

Zero-mean and diagonal or spherical Gaussian noise

Linear combination of the columns of $\boldsymbol{W}$

$$p(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n|0,\mathbf{I})$$
$$p(\mathbf{x}_n|\mathbf{z}_n) = \mathcal{N}(\boldsymbol{x}_n|\mathbf{W}\mathbf{z}_n,\Psi)$$

Diagonal for FA, spherical for PPCA

- Linear Gaussian Model. $\boldsymbol{W}$, $\boldsymbol{z}_n$'s, and Ψ can be learned (e.g, using EM, VI, MCMC)

Some Variants of FA/PPCA

- Gamma-Poisson latent factor model
  - Assumes K-dim non-negative latent variable $\mathbf{z_n}$ and D-dim count-valued observations $\mathbf{x_n}$
  - An example: Each $\mathbf{x_n}$ is the word-count vector representing a document

$$p(\mathbf{z_n}) = \prod_{k=1}^{K} \text{Gamma}(z_{nk}|a_k, b_k))$$
$$p(\mathbf{x_n}|\mathbf{z_n}) = \prod_{d=1}^{D} \text{Poisson}(x_{nd}|f(\mathbf{w_d}, \mathbf{z_n}))$$

Popular for modeling count-valued data (in text analysis, recommender systems, etc)

Non-negative priors often give a nice interpretability to such latent variable models (will see some more examples of such models shortly)

This is the rate of the Poisson. It should be non-negative, $\exp(\mathbf{w}_d^\top \mathbf{z}_n)$, or simply $\mathbf{w}_d^\top \mathbf{z}_n$ if $\boldsymbol{w}_d$ is also non-negative (e.g., using a gamma/Dirichlet prior on it)

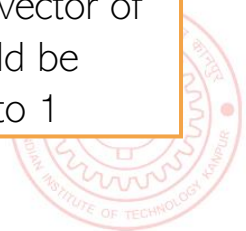
  - This can be thought of as a probabilistic non-negative matrix factorization model

- Dirichlet-Multinomial/Multinoulli PCA
  - Assumes K-dim non-negative latent variable $\mathbf{z_n}$ and D categorical obs $\mathbf{x_n} = \{\boldsymbol{x}_{nd}\}_{d=1}^{D}$
  - An example: Each $\mathbf{x_n}$ is a document with D words in it (each word is a categorical value)

Also sums to 1

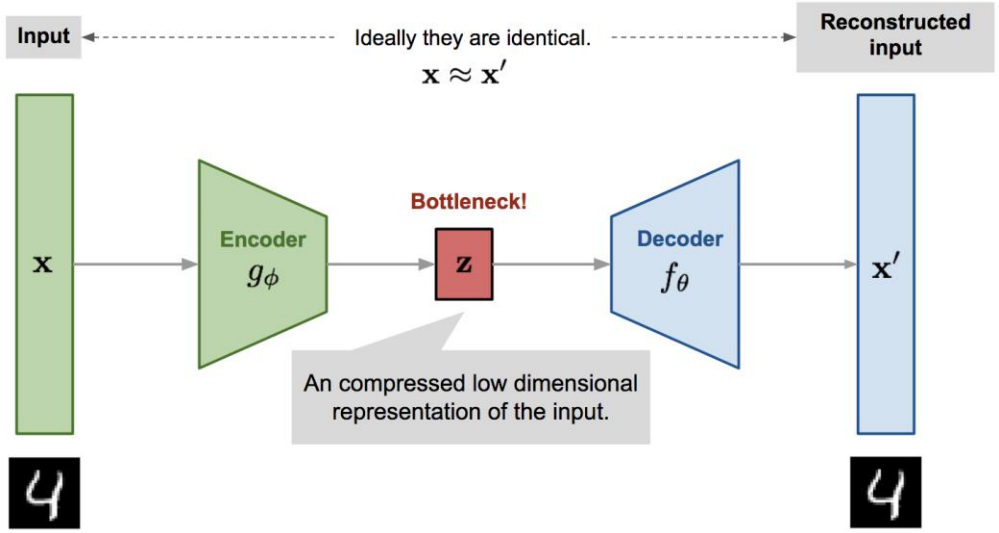
$$p(\mathbf{z_n}) = \text{Dirichlet}(\mathbf{z_n}|\boldsymbol{\alpha})$$
$$p(\mathbf{x_n}|\mathbf{z_n}) = \prod_{d=1}^{D} \text{Multinoulli}(x_{nd}|f(\mathbf{w_d}, \mathbf{z_n}))$$

This should give the probability vector of the multinoulli over x_{nd}. It should be non-negative and should sums to 1



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

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



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

- 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

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

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

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)

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

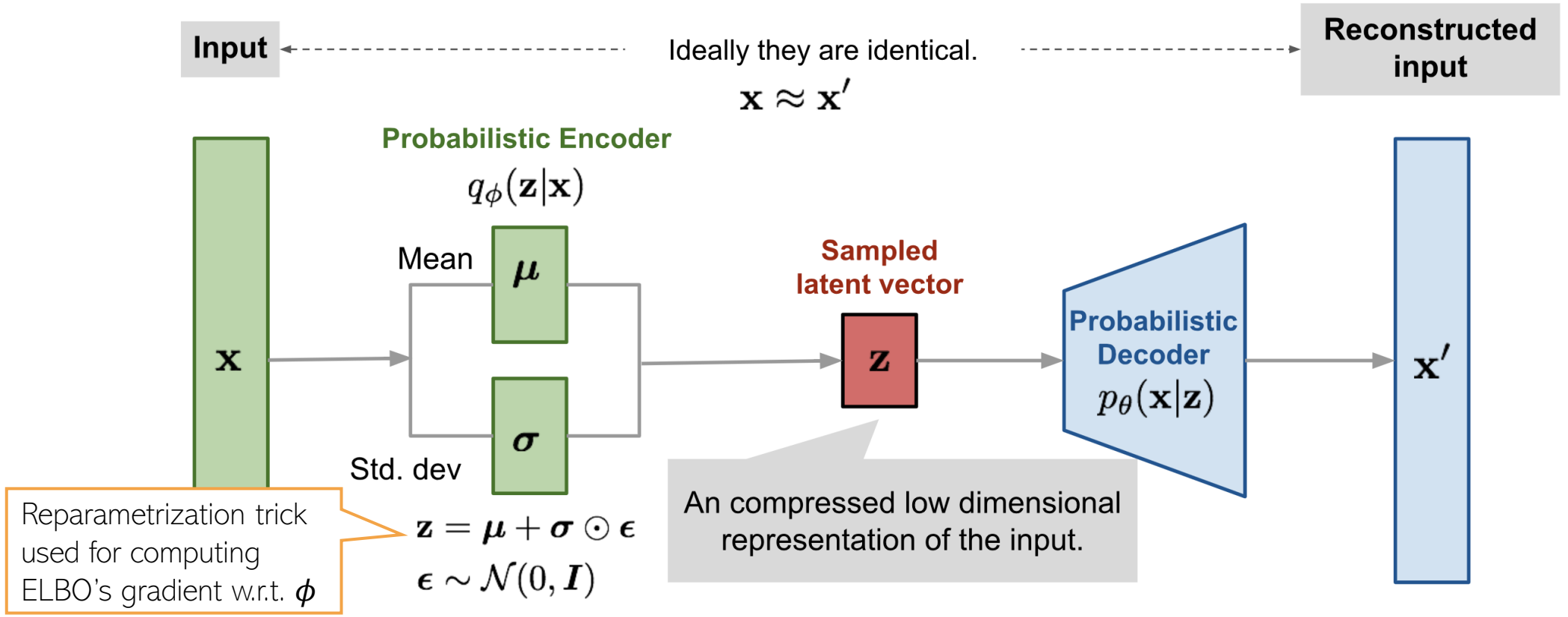
If q is Gaussian then the NN will output a mean and a variance

- 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

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



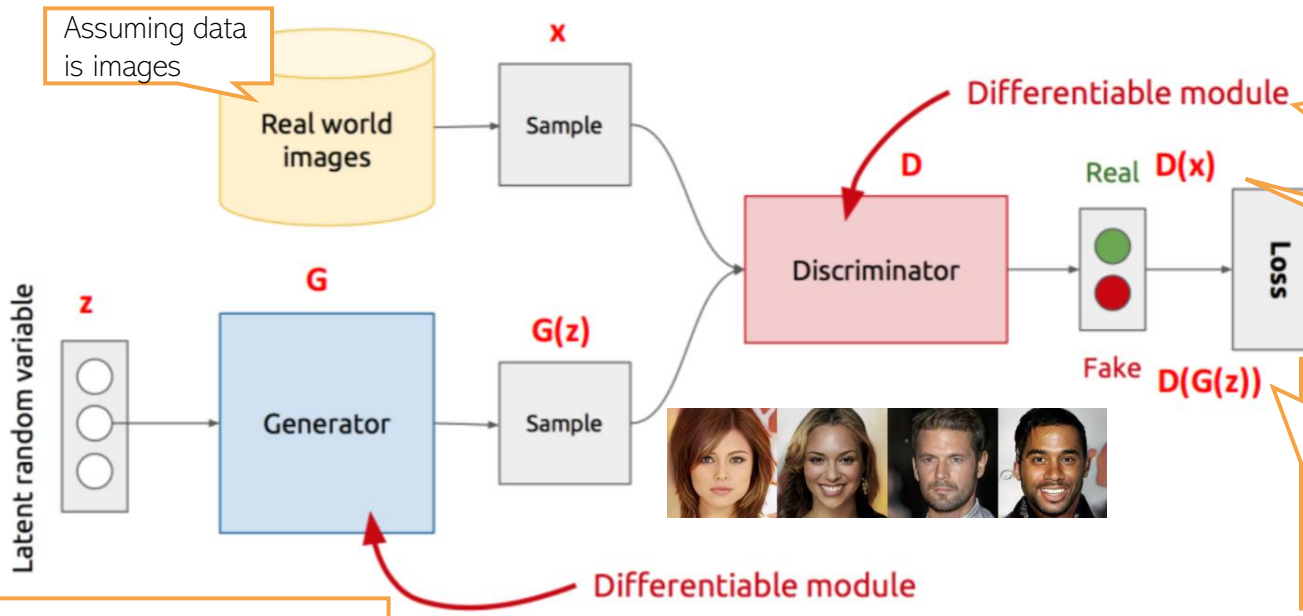
Pic source: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

Generative Adversarial Network (GAN)

- 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)

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

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



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

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

- With G fixed, the optimal 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 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

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

- 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 $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_d$

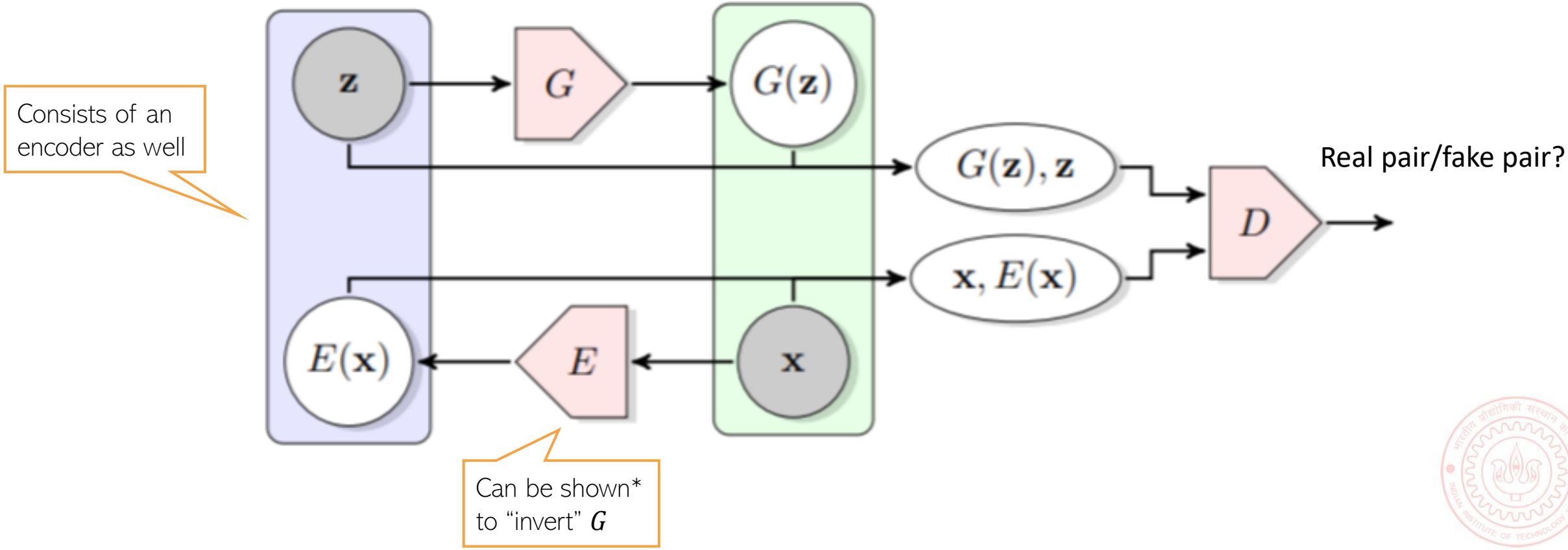
In practice, for stable training, we run $K > 1$ steps of optimizing w.r.t. D and 1 step of optimizing w.r.t. G

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



*Adversarial Feature Learning (Donahue et a Dumoulin l, 2017)

#Adversarially Learned Inference (Dumoulin et al, 2017)

Evaluating GANs

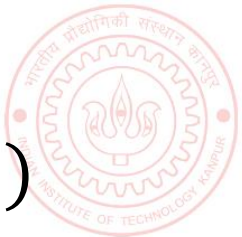
High IS and low FID is desirable

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

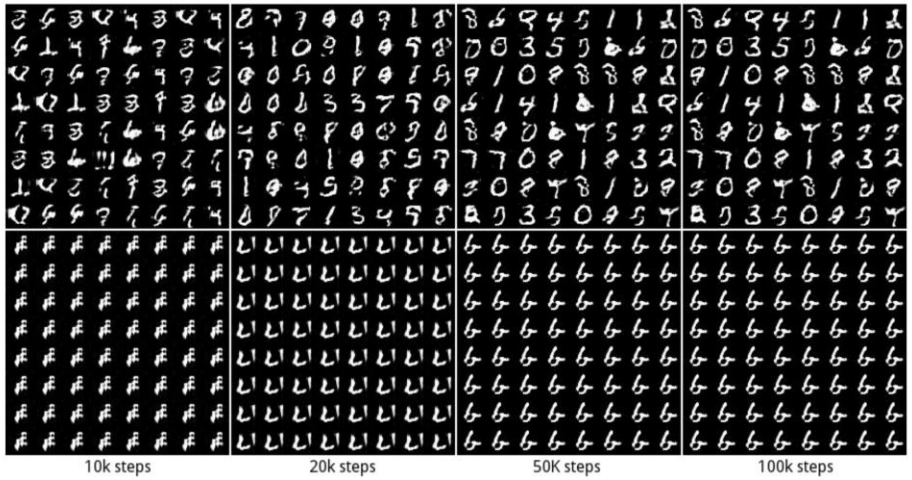
Using real data $\mathcal{N}(\mu_R, \Sigma_R)$ $\mathcal{N}(\mu_G, \Sigma_G)$ Using generated data

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