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 $z_n$ to $D$-dim $x_n$

$$p(z) \xrightarrow{Z_n} p(x|NN(z; W))$$

- If $z_n$ 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 $z$ from prior, pass it through the model to generate $x$

$$x_n \sim \mathcal{N}(\text{NN}(z_n; W), \sigma^2 I_D)$$

$$x_n \sim \text{ExpFam}(\text{NN}(z_n; W))$$

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

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)

  - The basic difference is that VAE assumes a prior $p(z)$ on the latent code $z$
    - This enables it to not just compress the data but also generate synthetic data
    - How: Sample $z$ 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)
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” $q_{\phi}(z|x)$

- VAE is learned by maximizing the ELBO

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

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

- The Reparametrization Trick is commonly used to optimize the ELBO
- Posterior is inferred only over z, and usually only point estimate on $\theta$ and $\phi$
Amortized Inference

- Latent variable models need to infer the posterior $p(z_n|x_n)$ for each observation $x_n$

- This can be slow if we have lots of observations because
  1. We need to iterate over each $p(z_n|x_n)$
  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)

\[
p(z_n|x_n) \approx q(z_n|\phi_n) = q(z_n|\text{NN}(x_n; W))
\]

  - Thus no need to learn $\phi_n$’s (one per data point) but just a single NN with params $W$
    - This will be our “encoder network” for learning $z_n$
    - Also very efficient to get $p(z_\ast|x_\ast)$ for a new data point $x_\ast$

  If $q$ is Gaussian then the NN will output a mean and a variance
Variational Autoencoder: The Complete Pipeline

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

\[
\mathcal{L}(\theta, \phi|x) = \mathbb{E}_{q\phi(z|x)} \left[ \log p\theta(x, z) - \log q\phi(z|x) \right]
\]

\[
= \mathbb{E}_{q\phi(z|x)} \left[ \log p\theta(x|z) \right] - \text{KL} (q\phi(z|x) || p\theta(z))
\]

---

VAE and Posterior Collapse

- VAEs may suffer from **posterior collapse**
  \[ \mathcal{L}(\theta, \phi | x) = \mathbb{E}_{q_\phi(z|x)} \left[ \log p_\theta(x|z) \right] - \text{KL} (q_\phi(z|x) \| p_\theta(z)) \]

- Thus, due to posterior collapse, reconstruction will still be good but the code \( z \) may be garbage (not useful as a representation for \( x \))

- Several ways to prevent posterior collapse, e.g.,
  - Use KL annealing
    \[ \mathcal{L}(\theta, \phi | x) = \mathbb{E}_{q_\phi(z|x)} \left[ \log p_\theta(x|z) \right] - \beta \text{KL} (q_\phi(z|x) \| p_\theta(z)) \]
  - Avoid KL from becoming 0 using some \( q \) that doesn’t collapse to the prior
  - More tightly couple \( z \) with \( x \) using skip-connections (Skip-VAE)

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

A carefully tuned value between 0 and 1

For example, keep the variance of \( q \) as fixed

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”)
  - 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)

- 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)

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

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

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

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

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

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

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

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

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

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.
The GAN training procedure can be summarized as

1. Initialize $\theta_g, \theta_d$;
2. for each training iteration do
   3. for $K$ steps do
      4. Sample minibatch of $M$ noise vectors $z_m \sim q_z(z)$;
      5. Sample minibatch of $M$ examples $x_m \sim p_D$;
      6. Update the discriminator by performing stochastic gradient ascent using this gradient:
         $$\nabla_{\theta_d} \frac{1}{M} \sum_{m=1}^{M} \left[ \log D(x_m) + \log(1 - D(G(z_m))) \right].$$
      7. Sample minibatch of $M$ noise vectors $z_m \sim q_z(z)$;
      8. Update the generator by performing stochastic gradient descent using this gradient:
         $$\nabla_{\theta_g} \frac{1}{M} \sum_{m=1}^{M} \log(1 - D(G(z_m))).$$
3. Return $\theta_g, \theta_d$

In practice, $\theta_g$ and $\theta_d$ denote the params of the deep neural nets defining the generator and discriminator, respectively.

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

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, instead of minimizing $\log(1 - D(G(z)))$, we maximize $\log \left( D(G(z)) \right)$.
GANs that also learn latent representations

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

\[ \min_{G} \max_{E} V(D, E, G) = \mathbb{E}_{x \sim p_x} \left[ \mathbb{E}_{z \sim p_{E}(\cdot|x)} [\log D(x, z)] + \mathbb{E}_{z \sim p_z} \left[ \mathbb{E}_{x \sim p_{G}(\cdot|z)} [\log (1 - D(G(z), z))] \right] \right] \]

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

\[ \hat{z} \sim q(z \mid x) \]
\[ \hat{x} \sim p(x \mid z) \]

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

- 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

$$\mathcal{N}(\mu_R, \Sigma_R) \quad \mathcal{N}(\mu_G, \Sigma_G)$$

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

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

- 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 $q(x_t|x_{t-1}) = \mathcal{N}(x_t|\sqrt{1-\beta_t}x_{t-1}, \beta_t I)$
- 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 $\mu_\theta$ and $\Sigma_\theta$ are neural nets
Denoising Diffusion Models: Training

- The model is trained by minimizing the following objective

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

\[
p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) := p(x_T) \prod_{t=1}^{T} \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))
\]

\[
q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) := \prod_{t=1}^{T} \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathbf{I})
\]

\[
\mathcal{L} = L_0 + L_1 + L_2 + \ldots + L_{T-1} + L_T
\]

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

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

\[
L_T = D_{KL}(q(x_T|x_0) \parallel p(x_T))
\]

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

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

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

This is also a Gaussian
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

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

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

An example: Neural Autoregressive Density Estimator (NADE)

An auto-regressive model