Notes on Generative Models

Kumar Krishna Agrawal

This handout is an introductory overview of generative models. It is intended to be a quick reference for those who are already familiar with the basic concepts of probability theory and machine learning. It is not intended to be a comprehensive review of the field.

Motivation

One view of machine learning is that it is the study of algorithms that make inference about probability distributions. In this view, the dataset is viewed as a sample from some underlying unknown probability distribution. That is for some unknown probability distribution $(P, \mathcal{X})$, defined on a set $\mathcal{X}$, we have a dataset $\mathcal{D} = \{x_1, \ldots, x_n\}$, where each $x_i \sim P$.

Given samples from the unknown distribution, we can define different inference tasks that aim to reason about the underlying distribution. Some examples include:

- **Density estimation**: Given a dataset $\mathcal{D}$, estimate the probability density function $p(x)$ of the underlying distribution P.
- **Data generation**: Given a dataset $\mathcal{D}$, generate new samples from the underlying distribution P.
- **Representation Learning**: Given a dataset $\mathcal{D}$, learn a representation of the underlying distribution P. This is particularly useful for high-dimensional support we have limited annotated data.

Generative Models

We focus on three classes of generative models:

- **Likelihood-based models**: Given a family of parametric models $\mathcal{M}$, we aim to learn a model p_θ from the family $\mathcal{M}$ that is close to the true distribution p_{data}. This is done by maximizing the likelihood of the observed data under the model, i.e. $\max_\theta \mathbb{E}_{x \sim p_{\text{data}}} \Big[\log p_\theta(x) \Big]$
- **Implicit models** : An alternative approach to representing a probability distribution, is to model the generative process. In other words, instead of explicitly modeling the density, we learn how to directly sample new points $\hat{x} \sim p_{\text{data}}$. Note that with enough samples, we can estimate the density of the distribution by computing the empirical distribution $\hat{p}_{\text{data}}$.

- **Diffusion models** : An emerging class of generative models is *score-based* or *diffusion* models. Note that in likelihood-based models we care about $\nabla_\theta \log p_\theta(x)$ that informs us about the direction of descent in the parameter space. In diffusion models, we instead care about the *score* $\nabla_x \log p_\theta(x)$ that informs us about how the inputs themselves should be perturbed to increase the probability of the output. Additionally, diffusion models are typically *iterative* in samping, where the inputs are perturbed multiple times to generate an output.

Unsupervised Representation Learning

Consider the setting where you have access to large amounts of unlabelled images from the internet, $\mathcal{D} = \{x_1, \ldots, x_n\}$, where each $x_i \in \mathbb{R}^d$. How do we learn meaningful representations of the data?

Contractive Autoencoder

One approach is performing non-linear dimensionality reduction using neural networks. For example, with a *contractive autoencoder*, we learn an encoder $f_\theta(x) : \mathbb{R}^d \to \mathbb{R}^k$ and a decoder $g_\phi(z) : \mathbb{R}^k \to \mathbb{R}^d$, such that $g_\phi(f_\theta(x)) \approx x$.

Denoising Autoencoder

As we've seen in lecture, data augmentations present a powerful tool for improving the generalization of neural networks. An instantiation of this in the autoencoder setting is the *denoising autoencoder* [1]. In this setting, we corrupt the input x with noise ϵ and learn an encoder $f_\theta(x) : \mathbb{R}^d \to \mathbb{R}^k$ and a decoder $g_\phi(z) : \mathbb{R}^k \to \mathbb{R}^d$, such that $g_\phi(f_\theta(x)) \approx x$.

[1] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008

Comparing Distributions

One of the core objectives of generative models is to learn p_θ that is close to p_{data}. This requires defining a notion of *closeness* between two distributions. Consider the setting where we have two distributions $P(x)$ and $Q(x)$, that share the same support $\mathcal{X}$.

Metrics that are commonly used to compare two distributions include:

- **Kullback-Leibler Divergence**: $\text{KL}(P||Q) = \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right]$
- **Jensen-Shannon Divergence**: $\text{JS}(P||Q) = \frac{1}{2}\text{KL}(P||M) + \frac{1}{2}\text{KL}(Q||M)$, where $M(x) = \frac{P(x)+Q(x)}{2}$

Note that the KL divergence is generally asymmetric, i.e. $\mathrm{KL}(P||Q) \neq \mathrm{KL}(Q||P)$.

Variational Autoencoders

VAEs fall in the class of likelihood-based models. The goal is to learn a model $p_\theta(x)$ that is close to the true distribution p_{data}. This is done by maximizing the likelihood of the observed data under the model, i.e. $\max_\theta \mathbb{E}_{x\sim p_{\text{data}}}\Big[\log p_\theta(x)\Big]$.

Evidence Lower Bound

Given a latent-variable model, $z \to x$, we have an alternative objective to maximize the likelihood of the data, i.e. $\max_\theta \mathbb{E}_{x\sim p_{\text{data}}}\Big[\log p_\theta(x)\Big]$. In particular, consider the expression

$$\begin{aligned}\log p_\theta(x) &= \log \int_z p_\theta(x,z)\mathrm{d}z \\ &= \log \int_z p_\theta(x|z)p(z)\mathrm{d}z\end{aligned}$$

The above formulation, is equivalent to sampling *latent-variables* z from the prior $p(z)$ and then computing the likelihood of the data given the latent variables. However, when the latent-variables are continuous, estimating the above expression requires integrating over the entire latent space. As this is intractable in practice, we could instead sample a finite number of latent variables $z_1, \ldots, z_n$ from the prior $p(z)$ and then compute the likelihood of the data given the latent variables. This is known as the *Monte Carlo approximation*, where we approximate the integral with a sum:

$$\log p_\theta(x) \approx \log \sum_{i=1}^{n} p_\theta(x|z_i)p(z_i)$$

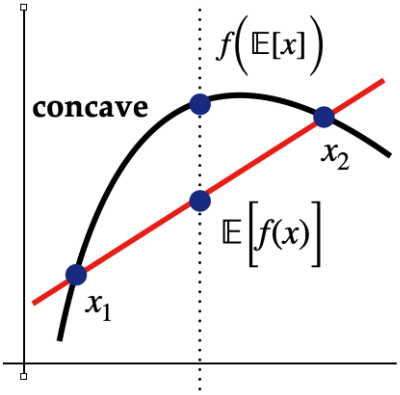
Note that typically, for $x \in \mathbb{R}^d, z \in \mathbb{R}^k$ with $(k < d)$. Estimating the likelihood of the higher-dimensional data x conditioned on the lower-dimensional latent-variables z is unstable to train, due to high variance gradients.

Instead we consider an alternative formulation, where we additionally want to find a paramteric approximation $q_\phi(z|x)$ of the posterior distribution $p(z|x)$ of latents z given observed datapoints x, This is known as the *variational inference* framework. In particular, we have the following expression:

$$\begin{aligned}\log p_\theta(x) &= \log \int_z p_\theta(x|z)p(z)\mathrm{d}z \\ &= \log \int_z \frac{q_\phi(z|x)}{q_\phi(z|x)} p_\theta(x|z)p(z)\mathrm{d}z \\ &= \log \left[\mathbb{E}_{z\sim q_\phi(z|x)}\left\{\frac{p(z)}{q_\phi(z|x)} p_\theta(x|z)\right\}\right]\mathrm{d}z\end{aligned}$$

Note that the above expression is valid for all approximations of the posterior $q_\phi(z|x)$ as long as the support of $q_\phi(z|x)$ is a superset of the support of $p(z)$. Optimizing the above expression is still difficult; instead we optimize a *lower bound* of the above expression, derived with help of Jensen's inequality.

Figure 1: Jensen's inequality



In particular, for any concave function (log in this case), we have the following inequality:

$$f(\mathbb{E}[x]) \geq \mathbb{E}[f(x)]$$

Plugging this inequality into the variational expression, we have the following lower-bound for true likelihood:

$$\begin{aligned}\log p_\theta(x) &\geq \mathbb{E}_{z\sim q_\phi(z|x)}\left\{\log \frac{p(z)}{q_\phi(z|x)} p_\theta(x|z)\right\} \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left\{\log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p(z)}\right\} \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p(z)}\right] \\ &= \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right]}_{\text{Reconstruction Likelihood}} - \underbrace{\mathrm{KL}(q_\phi(z|x)||p(z))}_{\text{Matching Posteriors}}\end{aligned}$$

The above approximation is called the Evidence Lower BOund (ELBO) and is easier to optimizer in practice[2]. One way to approximate the above expression is that we want to increase the likelihood of decoding the data given the latent variables, while also matching the posterior distribution of the latent variables to a distribution from which we can sample efficiently (e.g. Gaussian).

[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013

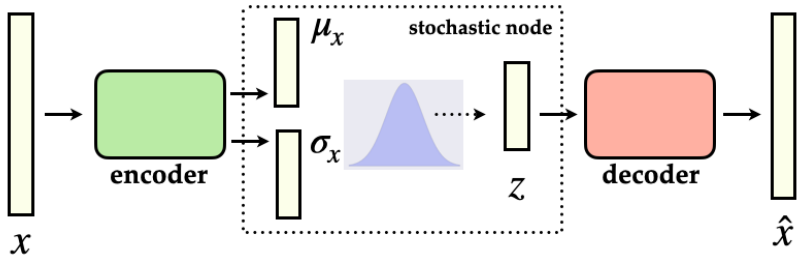


Figure 2: Variational Autoencoder

Reparameterization trick

Gradient based optimization of the above formulation is still difficult, since taking gradient through a stochastic node (e.g. sampling from a distribution) is not differentiable. To overcome this, we use the *reparameterization trick* to sample from the posterior distribution $q_\phi(z|x)$, by sampling from a standard normal distribution and then transforming it to the posterior distribution.

In particular, consider sampling from a Gaussian distribution $N(\mu, \sigma^2)$, where μ and σ are parameters of the distribution. Sampling from the original distribution can be *reparameterized* as follows:

$$\epsilon \sim N(0, 1)$$
$$z = \mu + \sigma \odot \epsilon$$

This resolves the trainability issue, since the above expression provides a modified computation graph:

Generative Adversarial Networks

As an implicit model, GANs present a different perspective to modelling the data distribution. In particular, GANs are composed of two networks:

- **Generator**: $G_\theta(z) : \mathbb{R}^k \to \mathbb{R}^d$, where $z \sim p(z)$
- **Discriminator**: $D_\phi(x) : \mathbb{R}^d \to \mathbb{R}$, where $x \sim p_{\text{data}}(x)$

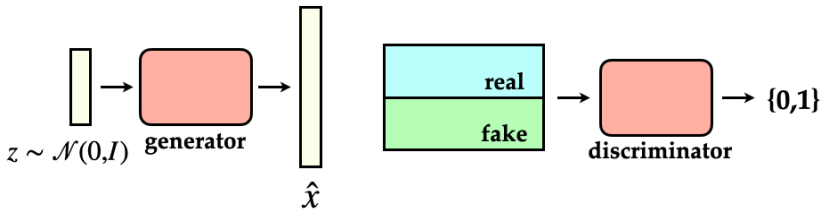


Figure 3: Generative Adversarial Network

Training

The high level intuition behind training GANs is that we want to train the generator to create samples that are indistinguishable from the real data. At the same time, we want to train the discriminator to be able to distinguish between real and fake data. This presents the following *minmax* optimization problem:

$$\min_{G_\theta} \max_{D_\phi} \underbrace{\mathbb{E}_{x \sim p_{\text{data}}(x)} \Big[\log D_\phi(x) \Big]}_{\text{Real Samples}} + \underbrace{\mathbb{E}_{z \sim p(z)} \Big[\log(1 - D_\phi(G_\theta(z))) \Big]}_{\text{Synthetic Samples}}$$

Algorithm 1: Training a GAN

```
# N: number of iterations
Initialize G_θ and D_φ
for i = 1 to N
    Sample z ~ p(z)
    Sample x ~ p_data(x)

    # compute discriminator loss
    discriminator_loss (L_D)
      = E_{x~p_data(x)}[log D_φ(x)] + E_{z~p(z)}[log(1 − D_φ(G_θ(z)))]

    # update discriminator
    φ ← φ + η∇_φ L_D

    # compute generator loss
    generator_loss (L_G)
      = E_{z~p(z)}[log(1 − D_φ(G_θ(z)))]

    # update generator
    θ ← θ − β∇_φ L_G
```

Sampling

Sampling from GAN is done by sampling latents from the prior distribution $p(z)$ and then passing it through the generator network.

In particular, we have the following generative process:

$$z \sim p(z)$$
$$x = G_\theta(z)$$

Diffusion Models

The algorithms and models we've considered directly learn to either estimate density (VAEs) or generate samples (GANs). Notably, these algorithms are single-step, meaning that there is a single step of probabilistic inference (variational or implicit).

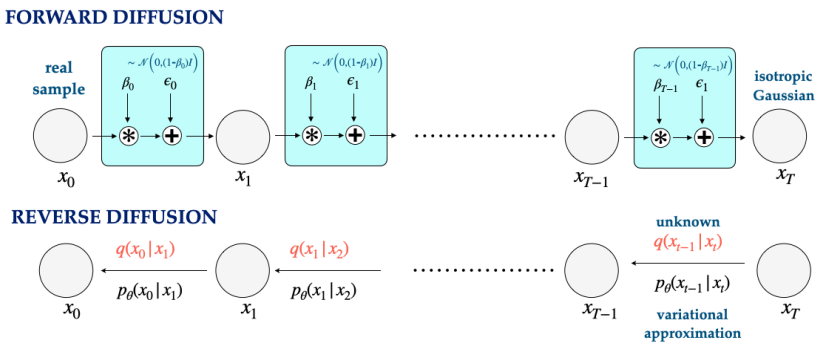


Figure 4: Diffusion Model iteratively adds noise to the input skewing the distribution towards well-understood distributions.

Diffusion models are a class of generative models that are *multi-step*, meaning that they perform multiple steps (say **T**) of inference. In particular, diffusion models are composed of two phases:

- **Forward**: In this phase, we start from samples from the real distribution and iteratively add noise to the samples. In particular, starting from $x_0 \sim p_{\text{data}}$ we have:

  $$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{\beta_{t-1}}\mathbf{x}_{t-1}, (1-\beta_{t-1})\mathbb{I})$$

  Notably, the variance of the distribution is a function of the step t, and the variance increases as we move forward in time. A typical example [3] of this would be linearly increasing from $\beta_0 = 10^{-4}$ to $\beta_T = 0.02$.

- **Reverse**: Sampling from the diffusion model is done by starting from a sample from our target distribution (e.g. isotropic Gaussian) and iteratively denoising the noisy inputs. In the Markov chain illustrated in fig. 4 this corresponds to starting at x_T and moving backwards in time, such that x_0 is the denoised sample.

  Performing this denoising step, however requires us to estimate $q(x_{t-1}|x_t)$ which is not tractable. Instead, we perform a variational

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33: 6840–6851, 2020

inference to estimate this posterior distribution, and maximize the ELBO (as in VAEs) for each time-step. A key insight that enables such inference, is that one can recover a closed form expression for $q(x_{t-1}|x_t, x_0)$.

For more details on the diffusion model, we refer the reader to these wonderful blog posts [4].

[4] Yang Song. Generative modeling by estimating gradients of the data distribution. May 2021; and Lilian Weng. What are diffusion models? Jul 2021

Training

Algorithm 2: Training Diffusion Models

```
# T: number of denoising steps

while not converged:
    x_0 ~ q(x_0)
    t ~ Uniform({1, ..., T})

    # sample noise
    ε ~ N(0, 1)

    # compute sample
    x_t = sqrt(ᾱ_t) x_0 + sqrt(1 − ᾱ_t) ε

    # compute the gradient
    grad = ∇_θ ||ε − ε_θ(x_t, t)||_2^2

    # update the parameters
    θ ← θ − α grad

return x_0
```

Sampling

With a trained estimator for the score function, we can reverse the diffusion process to sample from the data-distribution.

Algorithm 3: Sampling from DM

```
# T: number of denoising steps

x_T ~ N(0, I)
for t in range(T, 1, -1):
    if t > 1:
        z ~ N(0, I)
    else:
        z = 0

    # generate sample at timestep t-1
    x_{t-1} = (1/sqrt(α_t)) (x_t − ((1−α_t)/sqrt(1−ᾱ_t)) ε_θ(x_t, t)) + σ_t z

return x_0
```

References

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Yang Song. Generative modeling by estimating gradients of the data distribution. May 2021.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

Lilian Weng. What are diffusion models? Jul 2021.