AIML - CS 337

Lecture 21: Generative Models

7th November 2022

Lecturer: Abir De *Scribe: Group 43*

1 Introduction

So far, we have looked at various discriminative algorithms. It is important to understand the difference between Discriminative models and Generative models. Discriminative models draw boundaries in the data space, while generative models try to model how data is placed throughout the space. A generative model focuses on explaining how the data was generated, while a discriminative model focuses on predicting the labels of the data.

2 What to expect from a Generative Model

Let's say we are dealing with images. Given a set of images, $I = \{\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_N}\}$, a Generative model tries to learn the underlying distribution from which the data might have been obtained. However, this is not an easy task. We also expect the Generative model to generate new images similar to a given image $\mathbf{x}$, i.e the model should be able to sample an output image $\mathbf{x}'$ using the conditional distribution given an input image $\mathbf{x}$. A few obvious challenges are: How do we approximate the underlying distribution? and how do we calculate the similarity between an input $\mathbf{x}$ and a generated output $\mathbf{x}'$?

3 Universal approximator of a distribution

Goal To learn a function $P : \mathbb{R}^d \to [0, 1]$ satisfying $\int_{-\infty}^{\infty} P(\mathbf{x})d\mathbf{x} = 1$ and capable of capturing any distribution. For example, we would like $P_\theta(.)$ to capture different distributions by changing the parameter θ.

A possible approach For each $\mathbf{x}$, we can model the distribution as a Normal distribution with mean and covariance matrices taken as a function of θ and $\mathbf{x}$. We can model the mean and covariance as the output of some neural network. We can write

$$P_\theta(\mathbf{x}) \sim \mathcal{N}(\mu_\theta(\mathbf{x}), \Sigma_\theta(\mathbf{x}))$$

The neural network should be able to learn $\mu_\theta(\mathbf{x})$ and $\Sigma_\theta(\mathbf{x})$ to enable the Normal distribution to capture the desired distribution.

How do we train this model? We can use the KL divergence to measure the similarity between two probability distributions. Here we will be comparing the empirically generated PDF and the PDF obtained by the Generative model.

KL divergence For distributions P and Q of a continuous Random variable, the KL divergence is defined to be

$$KL(P||Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

It is a measure of how one probability distribution P is different from a second, reference probability distribution Q.

4 Generative procedure : Latent Variable Model

Let $\mathbf{x}$ denote the variable that represents our data and assume that $\mathbf{x}$ is generated from a latent variable z that is not directly observed. Here, z can be thought of an *encoded* representation of $\mathbf{x}$ and is analogous to a seed for generation. We assume that we can easily sample the latent variables according to some probability density function $P(z)$ defined over some high dimensional space $\mathcal{Z}$. Then, say we have a family of deterministic functions $f(z;\theta)$, parameterized by a vector θ in some space Θ, where $f : \mathcal{Z} \times \Theta$. f is deterministic, but if z is random and θ is fixed, then $f(z;\theta)$ is a random variable in the space $\mathcal{X}$. We wish to optimize θ such that we can sample z from $P(z)$ and, with high probability, $f(z;\theta)$ will be *like* the $\mathbf{x}$'s in our dataset.
To accommodate the notion of likelihood, we replace these deterministic functions with a probability distribution. The total probability of $\mathbf{x}$ being observed can then be expressed as

$$P(\mathbf{x}) = \int P(\mathbf{x}|z;\theta)P(z)dz$$

For example, we can assume that both z and $\mathbf{x}|z$ are distributed normally. Then the genrative process can be summarized as

$$z \sim \mathcal{N}(\mathbf{0}, I)$$
$$\mathbf{x}|z \sim \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z))$$

Here, $\mu_\theta(z)$ and $\Sigma_\theta(z)$ can be thought of as the output of some Neural Network which takes in z as an input.
The model should learn to increase $P(\mathbf{x})$ given some training data, i.e., the generative model should learn to make the training data more likely.

Training Let's say we have some training data $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}$. As mentioned above, we focus on maximizing the **Likelihood** of the training data. The likelihood of training data for a generative model can be written as

$$\begin{aligned} P(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}) &= \prod_{i=1}^{N} P(\mathbf{x_i}) \\ &= \prod_{i=1}^{N} \int P_\theta(\mathbf{x_i}|z) P(z) dz \end{aligned}$$

So, we can write the log likelihood as

$$LL(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}) = \sum_{i=1}^{N} \log \int P_\theta(\mathbf{x_i}|z) P(z) dz$$

Using *Jensen's* Inequality,

$$\sum_{i=1}^{N} \log \mathbb{E}_z[P_\theta(\mathbf{x_i}|z)] \geq \sum_{i=1}^{N} \mathbb{E}_z[\log P_\theta(\mathbf{x_i}|z)]$$

So, we can finally write

$$\sum_{i=1}^{N} \log \int P_\theta(\mathbf{x_i}|z) P(z) dz \geq \sum_{i=1}^{N} \int [\log P_\theta(\mathbf{x_i}|z)] P(z) dz = \sum_{i=1}^{N} \mathbb{E}_z[\log P_\theta(\mathbf{x_i}|z)]$$

Based on the above inequality, we can instead aim to maximise $\sum_{i=1}^{N} \mathbb{E}_z[\log P_\theta(\mathbf{x_i}|z)]$.
We can compute $\mathbb{E}_z[\log P_\theta(\mathbf{x_i}|z)]$ approximately. For a given i, first obtain a large number of samples $s_{i1}, s_{i2}, ..., s_{1S}$ for z and compute $\frac{1}{S} \sum_{j=1}^{S} \log P_\theta(\mathbf{x_i}|s_{ij})$. Thus we finally try to maximize

$$\sum_{i=1}^{N} \sum_{j=1}^{S} \frac{\log P_\theta(\mathbf{x_i}|s_{ij})}{S}$$

5 Conditional generation

So far we have seen the generation of a sample based on the training data without any conditions. Suppose, we now modify the problem : Given a sample $\mathbf{x}^*$, generate a sample $\mathbf{x}$ which is most similar to $\mathbf{x}^*$. In other words, we would like to maximize

$$P(\mathbf{x}|\mathbf{x}^*) = \int P(\mathbf{x}|z) P(z|\mathbf{x}^*) dz$$

The first quantity $P(\mathbf{x}|z)$ within the integral is known. The second part can be calculated as

$$\begin{aligned} P(z|\mathbf{x}^*) &= \frac{P(\mathbf{x}^*|z) P(z)}{P(\mathbf{x}^*)} \\ &= \frac{P(\mathbf{x}^*|z) P(z)}{\int_{\mathcal{Z}} P(\mathbf{x}^*|s) P(s) ds} \end{aligned}$$

Thus, the required probability becomes

$$P(\mathbf{x}|\mathbf{x}^*) = \int P(\mathbf{x}|z)\frac{P(\mathbf{x}^*|z)P(z)}{\int_{\mathcal{Z}} P(\mathbf{x}^*|s)P(s)ds}dz$$

All quantities in the above equation can be computed using known information. However, it is expensive to compute it in general.
Alternatively, we can model $P(z|\mathbf{x}^*)$ separately. It means that we need a new function $Q_\mathbf{x}(z)$ which can take a value of $\mathbf{x}$ and give us a distribution over z values that are most likely to produce $\mathbf{x}$. The space of z values that are likely under Q should be much smaller than the space of all z values which were all previously equally likely under the prior $P(z)$.

Training $Q_\mathbf{x}(z)$ A separate neural network can be used to train $Q_\mathbf{x}(z)$. Our objective can be to minimise the KL divergence between $Q_\mathbf{x}(z)$ and $P(z|\mathbf{x})$

$$\begin{aligned}
\mathrm{KL}(Q_\mathbf{x}(z), P(z|\mathbf{x})) &= \mathbb{E}_{z\sim Q_\mathbf{x}} \log Q_\mathbf{x}(z) - \mathbb{E}_{z\sim Q_\mathbf{x}} \left(\log \frac{P(\mathbf{x}|z)P(z)}{P(\mathbf{x})}\right) \\
&= \mathbb{E}_{z\sim Q_\mathbf{x}} \log Q_\mathbf{x}(z) - \mathbb{E}_{z\sim Q_\mathbf{x}} \log P(\mathbf{x}|z) - \mathbb{E}_{z\sim Q_\mathbf{x}} \log P(z) + \mathbb{E}_{z\sim Q_\mathbf{x}} \log P(\mathbf{x}) \\
&= \mathrm{KL}(Q_\mathbf{x}(z), P(z)) - \mathbb{E}_{z\sim Q_\mathbf{x}} \log P(\mathbf{x}|z) + \mathbb{E}_{z\sim Q_\mathbf{x}} P(\mathbf{x})
\end{aligned}$$

Here, our goal is to minimise the KL divergence between $Q_\mathbf{x}(z)$ and $P(z|\mathbf{x})$. Alternatively, we aim to maximize $\mathbb{E}_{z\sim Q_\mathbf{x}} \log P(\mathbf{x}|z) - \mathrm{KL}(Q_\mathbf{x}(z), P(z))$. The first part in this quantity aims to maximize the likelihood, whereas the second part tries to minimize the KL divergence between $Q_\mathbf{x}(z)$ and $P(z|\mathbf{x})$. The expressed we obtained here is also referred to as the Evidence Lower Bound(**ELBO**) function [1]. Thus, our objective function will be $\sum_{i=1}^{N} \mathbb{E}_{z\sim Q_{\mathbf{x_i}}} \log P(\mathbf{x_i}|z) - \mathrm{KL}(Q_{\mathbf{x_i}}(z), P(z))$. We aim to maximize ELBO and identify the underlying parameters which give us the maximum.

Please refer to [2] and [3] for more info on Variational Auto encoders

References

[1] Understanding variational autoencoders. https://en.wikipedia.org/wiki/Evidence_lower_bound.

[2] C. Doersch. Tutorial on variational autoencoders. https://arxiv.org/pdf/1606.05908.pdf, 2016.

[3] J. Rocca. Understanding variational autoencoders. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73, 2019.