Deep Generative Models

Chapter 3: Generation by Explicit Distribution Learning

Ali Bereyhi

ali.bereyhi@utoronto.ca

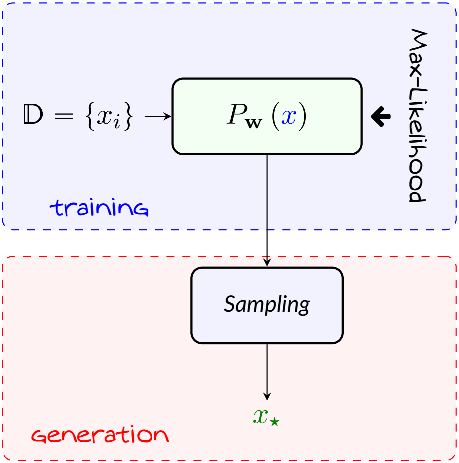
Department of Electrical and Computer Engineering

University of Toronto

Summer 2025

Conventional Data Generation

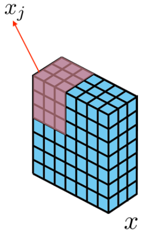
In conventional approaches, *we learn the data distribution and sample it!*



Data Samples: *Arrays of Unit Components*

Let's assume $x = \texttt{DataType}\left\{x_1, \ldots, x_d\right\}$ *for some data-type*

- *This data-type could be any form of array*
  - ↳ *Tensor of pixels or pixel-patches*
  - ↳ *A sequence of scalars, vectors or tensors*



- x_j for $j = 1, \ldots, d$ are unit components
  - ↳ *They could be scalars or fixed-size tensors*
  - ↳ *We assume that the space of* x_j is *small and manageable*

We refer to d *as data dimension*

↳ *This is in practice very large*

Generation by *Sampling*

Assume we trained the model $P_{\mathbf{w}}(x)$: *we intend to generate a new sample*

+ *Well, we simply sample the trained $P_{\mathbf{w}}(x)$! Right?!*
– Sure! But, let's see how simple is sampling on our computer!

Say every x_i is a discrete variable with C possible outcomes, i.e.,

$$x_j \in \mathbb{A} = \{a_1, \ldots, a_C\}$$

The question that we intend to answer is

+ *How complicated is it to sample from $P_{\mathbf{w}}(x)$?*

Recap: *Sampling a Multinomial Distribution*

Example: *Say* $u \sim \mathrm{Unif}\left(0,1\right)$ *is uniformly distributed between* 0 *and* 1. *We pass* u *through the following thresholding function to build* x *as*

$$x = 0 \text{ if } 0 \leqslant u < p \text{ and } x = 1 \text{ if } p \leqslant u \leqslant 1$$

The new variable x *is distributed as* $P\left(x=0\right) = 1 - P\left(x=1\right) = p$

↳ *We can sample a binary variable by thresholding a uniform sample* u*!*

This is how the computer directly samples a multinomial distribution: to sample from the multinomial distribution $\mathbf{p} = [p_1, \ldots, p_C]$ *with* C *outcomes*

↳ *Break the interval* $[0,1]$ *with levels proportional to* $p_1, \ldots, p_C$

↳ *Sample a uniform variable* $u \sim \mathrm{Unif}\left(0,1\right)$

↳ *Threshold* u *with the levels*

Recap: *Sampling a Multinomial Distribution*

```
DirectSample(p = [p_1, ..., p_C]):
1: for c = 1 : C do
2:     Build the intervals at edges sum_{j=1}^{c} p_j
3: end for
4: Sample a standard uniform u                          u ~ Unif(0,1)
5: Threshold u based on the intervals
6: return thresholded outcome
```

$\texttt{DirectSample}(\mathbf{p} = [p_1, \ldots, p_C])$:

1: **for** $c = 1 : C$ **do**
2: *Build the intervals at edges* $\sum_{j=1}^{c} p_j$
3: **end for**
4: *Sample a standard uniform* u — $u \sim \mathrm{Unif}\,(0, 1)$
5: *Threshold* u *based on the intervals*
6: **return** thresholded outcome

Moral of Story

To directly sample a multinomial distribution with C possible outcomes, we need to compute $\mathcal{O}\,(C)$ levels

Sampling Complexity

Back to our problem: *we have*

$$x = \texttt{DataType}\left\{x_1, \ldots, x_d\right\}$$

with each x_j having C possible outcomes

+ *How complicated is it to sample from $P_{\mathbf{w}}\left(x\right)$?*
- *Let's so a simple calculation*

$$\text{\# of outcomes for } x = C^d \rightsquigarrow \text{complexity } = \mathcal{O}\left(C^d\right)$$

Conclusion

Even if we have trained $P_{\mathbf{w}}\left(x\right)$ perfectly, it's exponentially hard to directly sample from the trained model!

Sampling Complexity

+ *Is it the same story if x_j is continuous?*
– Yes!

To directly sample[1] a target density function, we

1. *sample a Gaussian distribution*
2. *pass it through a transform computed using C density values*

Similar to discrete case

$$\text{\# of density values required for } x = C^d \rightsquigarrow \text{complexity } = \mathcal{O}\left(C^d\right)$$

[1] *Note that this is direct sampling. For high dimensional distributions, the computers use more efficient approaches like Gibbs sampling which are of polynomial order. We talk about them in later sections briefly.*

Training: *Primary Assumptions*

To train the model, we use maximum likelihood estimation (MLE)

Likelihood *(formal)*

Likelihood of model $P_{\mathbf{w}}$ computed on $\mathbb{D} = \left\{ x^i \text{ for } i = 1, \ldots, n \right\}$ is defined as

$$\mathcal{L}(\mathbf{w}) = P_{\mathbf{w}}(\mathbb{D})$$

Note that we need joint distribution; however, we assume i.i.d. samples

Basic Assumption: *i.i.d. Dataset*

We assume that x^j are independently sampled from data distribution; thus,

$$\mathcal{L}(\mathbf{w}) = \prod_j P_{\mathbf{w}}\left(x^j\right)$$

Training *via MLE*

To train the model, we maximize the log-likelihood function

$$\mathbf{w}^\star = \operatorname*{argmax}_{\mathbf{w}} \log \mathcal{L}(\mathbf{w}) = \operatorname*{argmax}_{\mathbf{w}} \frac{1}{n} \sum_j \log P_{\mathbf{w}}\left(x^j\right)$$

+ *But, why is MLE a good approach?*
– There are two ways to see it: *intuitive* and *through divergence*

Intuitive Justification of MLE

We know that the data-space is huge as compared to the number of samples

- *the collected samples are the most-likely ones*
- *the probability of them happening together should be very high*

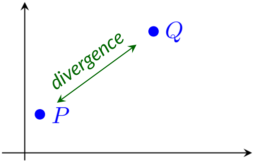
To illustrate the second view: *we need to recap the notion of divergence*

Recap: *Kullback-Leibler Divergence*

As seen in Assignment 1: *two distributions can be compared via divergence*

↳ *If they are the same* $\rightsquigarrow$ *the divergence is zero*

↳ *The more different they are* $\rightsquigarrow$ *the larger the divergence will be*



Kullback-Leibler Divergence

KL divergence between two distributions $P(x)$ and $Q(x)$ is defined as

$$D_{\mathrm{KL}}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} = \mathbb{E}_{x\sim P}\left\{\log \frac{P(x)}{Q(x)}\right\}$$

Properties of KL Divergence

1. *Each distribution is in divergence zero from itself*

$$D_{\mathrm{KL}}\left(P \| P\right) = \mathbb{E}_{x\sim P}\left\{\log \frac{P\left(x\right)}{P\left(x\right)}\right\} = \mathbb{E}_{x\sim P}\left\{\log 1\right\} = 0$$

2. *KL divergence is always non-negative* $\rightsquigarrow$ *Gibbs' Inequality*

$$D_{\mathrm{KL}}\left(P \| Q\right) = \mathbb{E}_{x\sim P}\left\{\log \frac{P\left(x\right)}{Q\left(x\right)}\right\} \underset{\text{Jensen Inequality}}{\geqslant} -\log \mathbb{E}_{x\sim P}\left\{\frac{Q\left(x\right)}{P\left(x\right)}\right\} = 0$$

Attention: *KL divergence is not symmetric*

$$D_{\mathrm{KL}}\left(P \| Q\right) = \mathbb{E}_{x\sim P}\left\{\log \frac{P\left(x\right)}{Q\left(x\right)}\right\} \neq \mathbb{E}_{x\sim Q}\left\{\log \frac{Q\left(x\right)}{P\left(x\right)}\right\} = D_{\mathrm{KL}}\left(Q \| P\right)$$

Estimating KL Divergence

Note that KL divergence is given as an expectation: *we can use the law of large numbers to estimate it from a large number of samples*

Law of Large Numbers (LLN) *(informal)*

Assume $\{x^j \text{ for } j = 1, \dots n\}$ *are i.i.d. samples from* $P(x)$*; then,*

$$\frac{1}{n} \sum_j F\left(x^j\right) \to \mathbb{E}_{x \sim P} \left\{F\left(x\right)\right\}$$

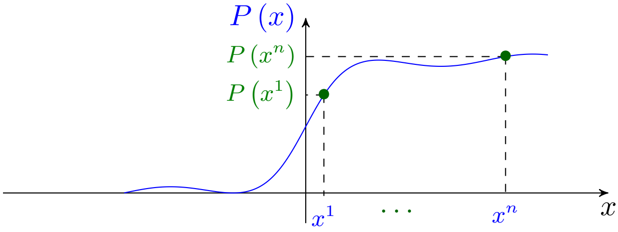
for a function $F(\cdot)$ *as* $n \to \infty$

So, we can estimate the KL divergence from dataset $\mathbb{D}$ *as*

$$\hat{D}_{\rm KL}(P \| Q) = \frac{1}{n} \sum_j \log \frac{P\left(x^j\right)}{Q\left(x^j\right)}$$

Another View: *MLE as Divergence Minimization*

+ *But, what kind of estimate is it?! We still need P!*
– Well! We only need to *evaluate its value at sample points!*



+ *Ready for the alternative viewpoint?!*
– Yes! *An alternative viewpoint on MLE is that*

MLE minimizes KL divergence between model and data distribution

Another View: *MLE as Divergence Minimization*

Genie-aided Power

A genie can compute data distribution at an arbitrary point for us, i.e., if a point x^j from the data-space is given $\rightsquigarrow$ the genie can give us $P(x^j)$

With the help of the genie, we can use dataset $\mathbb{D}$ to estimate the KL divergence between data distribution $P(x)$ and generative model $P_{\mathbf{w}}(x)$

$$\Delta(\mathbf{w}) = \hat{D}_{\mathrm{KL}}(P \| P_{\mathbf{w}}) = \frac{1}{n}\sum_j \log \frac{P(x^j)}{P_{\mathbf{w}}(x^j)}$$

Looking at this estimate, we can say

- *This estimate depends on the model parameter $\mathbf{w}$*
- *The right way to train the model $P_{\mathbf{w}}(\cdot)$ is to minimize this estimate*
  - ↳ *As the divergence goes to zero $\rightsquigarrow$ $\boxed{P_{\mathbf{w}} \to P}$*

Another View: *MLE as Divergence Minimization*

So, the model with minimal KL divergence from data distribution is learned as

$$
\begin{aligned}
\mathbf{w}^\star = \underset{\mathbf{w}}{\mathrm{argmin}} \Delta\left(\mathbf{w}\right) &= \underset{\mathbf{w}}{\mathrm{argmin}} \hat{D}_{\mathrm{KL}}\left(P \| P_{\mathbf{w}}\right) \\
&= \underset{\mathbf{w}}{\mathrm{argmin}} \frac{1}{n} \sum_j \log \frac{P\left(x^j\right)}{P_{\mathbf{w}}\left(x^j\right)} \\
&= \underset{\mathbf{w}}{\mathrm{argmin}} \left[\frac{1}{n} \sum_j \log P\left(x^j\right) - \frac{1}{n} \sum_j \log P_{\mathbf{w}}\left(x^j\right) \right] \\
&= \underset{\mathbf{w}}{\mathrm{argmax}} \frac{1}{n} \sum_j \log P_{\mathbf{w}}\left(x^j\right)
\end{aligned}
$$

Bingo! This is indeed MLE!

↳ *and we do not really need the genie* ☺

MLE: *Final Notes*

MLE $\equiv$ KL Divergence Minimization

Maximum likelihood learning make sense, since it minimizes the KL divergence between the model and data distribution

- ↳ *We can only estimate KL divergence using the dataset*
- ↳ *Clearly, there is some estimation error* $\rightarrow$ *that's what it is!*

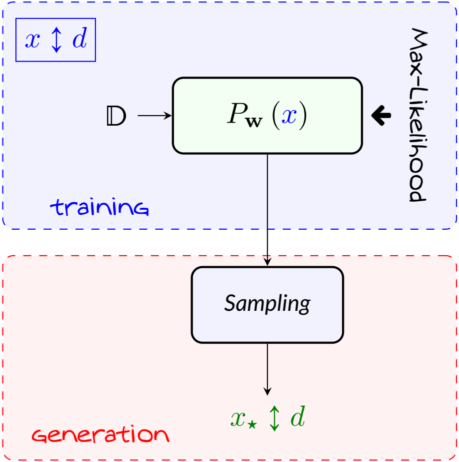
Attention

Note that even in perfect training log-likelihood does not get to zero necessarily!

$$\hat{D}_{\mathrm{KL}}\left(P \| P_{\mathbf{w}^\star}\right) = 0 \rightsquigarrow \frac{1}{n}\sum_j \log P_{\mathbf{w}^\star}\left(x^j\right) = \frac{1}{n}\sum_j \log P\left(x^j\right) = -\hat{H}\left(P\right)$$

where $\hat{H}\left(P\right)$ *is the estimate of data entropy*

Summary of *Conventional Approaches*



minimizes $\hat{D}_{\mathrm{KL}}(P|P_{\mathbf{w}})$

complex in general