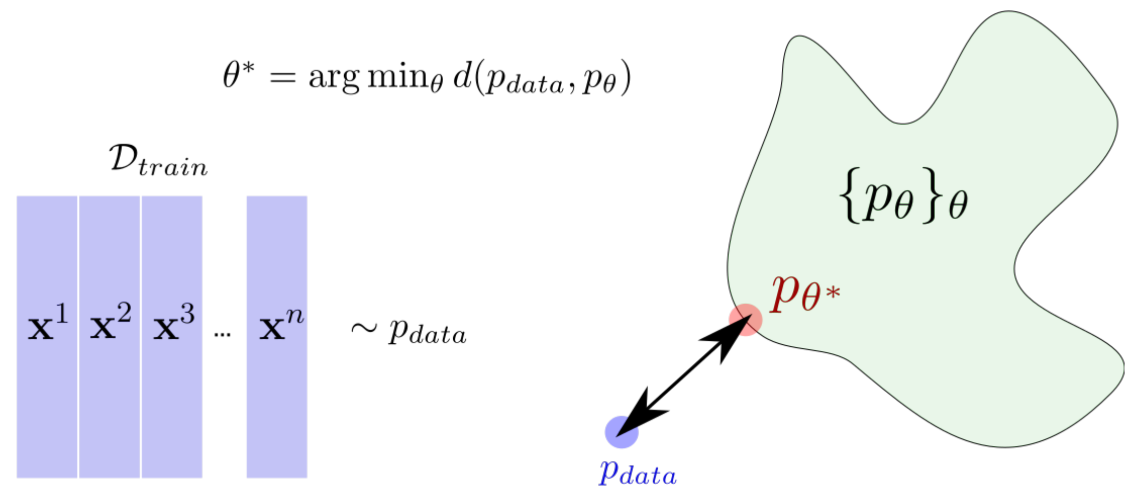
Training DGMs

October 25, 2022



We suppose that our training data come from an **original distribution** p_{data} (which is **not explicitly known**).

Instead, we have access to a set $\mathcal{D}_{train}$ of n **samples** $\mathbf{x}^i$ from p_{data}, which we suppose independent and identically distributed.

On the other side, we consider a family of **parameterized distributions** p_θ, parameterized by θ (for example, the parameters we saw on the auto-regressive models).

We want to determine an "optimal" θ^* such that p_{θ^*} is **as close as possible to** p_{data}.

Note that it is **impossible to get** p_{data} **exactly**: We do not have access to it but only to an approximation through the n samples.

Example

Each image can be seen as a vector $\mathbf{x}^i$ of $28 \times 28 = 784$ binary variables.

The whole space of binary images is huge: $2^{784} \approx 10^{236}$. Of course, the **set of plausible figure images is much smaller but still very large**. So, even 1 million sample images would be little compared to the true support of p_{data}.

0.1 Loss functions for generative models

An important point to state the problem correctly is to evaluate **how far** a candidate p_θ is from p_{data}.

The **Kullback-Leibler (KL) divergence** between two distributions p and q is

$$D_{KL}(p||q) \triangleq \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

- It is **not symmetric**: $D_{KL}(p||q) \neq D(q||p)$.
- It satisfies, for all p, q

$$D_{KL}(p||q) \geq 0,$$

with equality if and only if $p = q$.

To verify this statement, we use the fact that the KL expression can be interpreted as an **expectation** and the fact that $-\log$ is a **convex** function:

$$-\log\left(E_{\mathbf{x}\sim p}\left[\frac{q(\mathbf{x})}{p(\mathbf{x})}\right]\right) \leq E_{\mathbf{x}\sim p}\left[-\log\left(\frac{q(\mathbf{x})}{p(\mathbf{x})}\right)\right].$$

Now, the term on the left is:

$-\log\left(E_{\mathbf{x}\sim p}\left[\frac{q(\mathbf{x})}{p(\mathbf{x})}\right]\right) = -\log\left(\sum_{\mathbf{x}} p(\mathbf{x})\left[\frac{q(\mathbf{x})}{p(\mathbf{x})}\right]\right) = -\log\left(\sum_{\mathbf{x}} q(\mathbf{x})\right) = -\log 1 = 0.$

Hence:

$$E_{\mathbf{x}\sim p}\left[-\log\left(\frac{q(\mathbf{x})}{p(\mathbf{x})}\right)\right] \geq 0.$$

To come back to our original problem:

$$\begin{aligned}
D_{KL}(p_{data}||p_\theta) &= E_{\mathbf{x}\sim p_{data}}\left[-\log\left(\frac{p_\theta(\mathbf{x})}{p_{data}(\mathbf{x})}\right)\right] && (1)\\
&= -E_{\mathbf{x}\sim p_{data}}\left[\log\left(p_\theta(\mathbf{x})\right)\right] + E_{\mathbf{x}\sim p_{data}}\left[\log\left(p_{data}(\mathbf{x})\right)\right]. && (2)
\end{aligned}$$

Now, observe that the second term does **not depend** on θ. Hence,

$$\theta^* = \arg\min_\theta -E_{\mathbf{x}\sim p_{data}}\left[\log\left(p_\theta(\mathbf{x})\right)\right] = \arg\max_\theta E_{\mathbf{x}\sim p_{data}}\left[\log\left(p_\theta(\mathbf{x})\right)\right].$$

This expression on the right is the **log likelihood**:

- high values when samples from p_{data} are **evaluated with high values on** p_θ (likely samples from p_{data} should be likely on p_θ),
- very low (negative) values if $p_\theta(\mathbf{x})$ is small.

Note that with this formulation:

- we **know** how to make p_θ as close as possible to p_{train},
- we **do not know** how close we will be at the end: there will still be an unknown part $E_{\mathbf{x}\sim p_{data}}\left[\log\left(p_{data}(\mathbf{x})\right)\right]$ that we are not taking into account.

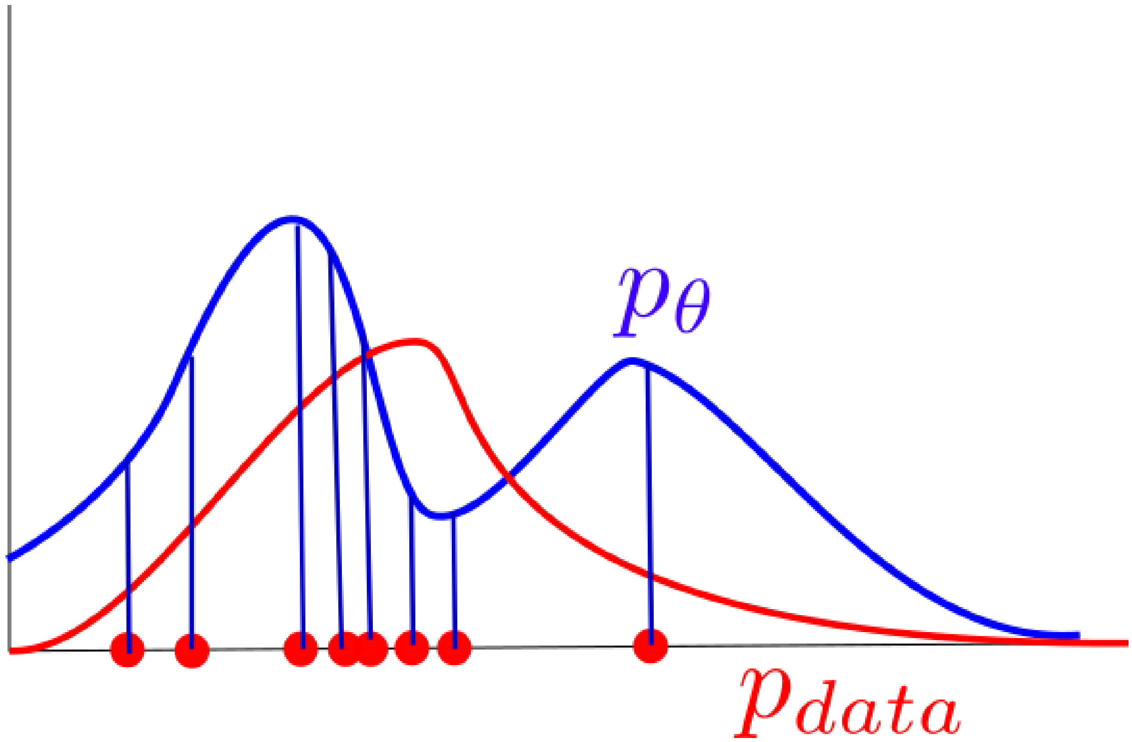
How to compute $E_{\mathbf{x}\sim p_{data}}\left[\log\left(p_\theta(\mathbf{x})\right)\right]$?

Since we only have samples $\mathbf{x} \in \mathcal{D}_{train}$ from p_{data}, we will approximate the true expectation by an **empirical expectation**

$$E_{\mathbf{x}\sim p_{data}}\left[\log\left(p_\theta(\mathbf{x})\right)\right] \approx E_{\mathbf{x}\in\mathcal{D}_{train}}\left[\log\left(p_\theta(\mathbf{x})\right)\right].$$

Hence, training will be done by **maximizing the log-likelihood**:

$$\theta^* = \arg\max_\theta \frac{1}{|\mathcal{D}_{train}|}\sum_{\mathbf{x}\in\mathcal{D}_{train}} \log\left(p_\theta(\mathbf{x})\right) = \arg\max_\theta \sum_{\mathbf{x}\in\mathcal{D}_{train}} \log\left(p_\theta(\mathbf{x})\right).$$



Note that this is equivalent to **maximizing the likelihood of the independent data within** $\mathcal{D}_{train}$:

$$\theta^* = \arg\max_\theta \prod_{\mathbf{x} \in \mathcal{D}_{train}} p_\theta(\mathbf{x}).$$

We will call:

$$L(\theta, \mathcal{D}_{train}) \triangleq \prod_{\mathbf{x} \in \mathcal{D}_{train}} p_\theta(\mathbf{x}).$$

The approximation above is fundamentally a **Monte-Carlo estimation process**!

Idea: **approximate an expected value** (here, $E_{\mathbf{x}\sim p_{data}}[\log(p_\theta(\mathbf{x}))]$) through an **empirical expectation on samples** from the same distribution.

$$l = E_{\mathbf{x}\sim p_{data}}[\log(p_\theta(\mathbf{x}))] \approx \hat{l} = \frac{1}{|\mathcal{D}_{train}|} \sum_{\mathbf{x} \in \mathcal{D}_{train}} \log(p_\theta(\mathbf{x})) = \frac{1}{|\mathcal{D}_{train}|} \log L(\theta, \mathcal{D}_{train}).$$

Note that:

- l is a **deterministic** value (the value of a sum or integral).
- $\hat{l}$ is a **random variable** (it is a weighted sum of r.v.).

We have some useful properties for this estimate $\hat{l}$:

- **Unbiased** estimate: $E_{p_{data}}[\hat{l}] = l$.
- **Convergence**: $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \log\left(p_\theta(\mathbf{x}^i)\right) = l$.

- **Variance** proportional to the inverse of the number of samples $\frac{1}{n}$:

$$var\left[\hat{l}\right] = \frac{1}{n} E_{\mathbf{x}\sim p_{data}}\left[(\log\left(p_\theta(\mathbf{x})\right) - l)^2\right].$$

Example

A **biased coin** with head (H) and tail (T).

$$\mathcal{D}_{train} = \{H, H, H, T, H, T, H, H\},$$

p_θ would be a **Bernoulli distribution** with parameter θ (probability of having an H). How to choose θ according to what we have seen?

We will have:

$$p_\theta(\mathbf{x} = H) = \theta, \tag{3}$$
$$p_\theta(\mathbf{x} = T) = 1 - \theta, \tag{4}$$

and we can evaluate the likelihood of $\mathcal{D}_{train}$.

$$L(\theta, \mathcal{D}_{train}) = \theta^6(1-\theta)^2,$$

and

$$\log L(\theta, \mathcal{D}_{train}) = 6\log\theta + 2\log(1-\theta).$$

More generally, log-likelihood function

$$\log L(\theta, \mathcal{D}_{train}) = \#heads \log\theta + \#tails \log(1-\theta).$$

Differentiating the expression above:

$$\frac{\partial \log L}{\partial \theta} = \frac{\#heads(1-\theta) - \#tails\theta}{\theta(1-\theta)} = \frac{\#heads - \theta(\#heads + \#tails)}{\theta(1-\theta)},$$

that cancels at $\theta^* = \frac{\#heads}{\#heads+\#tails}$. Note that

$$\frac{\partial^2 \log L}{\partial \theta^2} = -\frac{\#heads}{\theta^2} - \frac{\#tails}{(1-\theta)^2} < 0,$$

hence we have a maximum. So, in the example $\theta^* = \frac{3}{4}$.

Now, in general, things are not that easy!

$$\theta^* = \arg\max_{\theta} \log L(\theta, \mathcal{D}_{train}).$$

with

$$\log L(\theta, \mathcal{D}_{train}) = \sum_{\mathbf{x}\in\mathcal{D}_{train}} \log\left(p_\theta(\mathbf{x})\right),$$

but in most cases:

- it does not have a **closed form solution**;
- it is not even convex (so **its optimization is not trivial at all**).

The most common option is to use **gradient ascent methods**:

- Initialize $\theta^{(0)}$ at random.
- Compute $\nabla_\theta \log L(\theta, \mathcal{D}_{train})$ (e.g., by **back-propagation**).
- Apply gradient ascent:

$$\theta^{(k+1)} = \theta^{(k)} + \alpha_k \nabla_\theta \log L(\theta, \mathcal{D}_{train}).$$

*

In general, gives **decent results**.

What if $\mathcal{D}_{train}$ is very large? $\log L(\theta, \mathcal{D}_{train})$ may be very expensive to compute:

$$\log L(\theta, \mathcal{D}_{train}) = n \sum_{\mathbf{x}\in\mathcal{D}_{train}} \frac{1}{n} \log\left(p_\theta(\mathbf{x})\right) = n E_{\mathbf{x}\in\mathcal{D}_{train}}\left[\log\left(p_\theta(\mathbf{x})\right)\right].$$

Hence we can use again Monte-Carlo:

- with one sample (**stochastic gradient**):

$$\log L(\theta, \mathcal{D}_{train}) \approx n \log\left(p_\theta(\mathbf{x})\right) \text{ for } \mathbf{x} \sim \mathcal{D}_{train},$$

- with a subset of $\mathcal{D}_{train}$ (**mini-batch stochastic gradient**)

$$\log L(\theta, \mathcal{D}_{train}) \approx \sum_{\mathbf{x}\sim\mathcal{D}_{train}} \log\left(p_\theta(\mathbf{x})\right).$$

If the space for the parameterized p.d.f p_θ is limited, then you may have a lot of training data from p_{train}, but the **limitations of the model itself** p_θ make that you cannot fit well p_{data}: **bias**.

On the opposite, if your p_θ are very complex and expressive, then to fit them well you need an **infinite amount of data** in $\mathcal{D}_{train}$, which you **do not have**. Hence, given the **finiteness** of your training data, the fitting problem is an **ill-posed problem**: you may have lots of θ that fit well $\mathcal{D}_{train}$: **variance**.

In choosing one class of model (i.e. the design of p_θ), you may have to balance the two aspects:

- Distribution space too small: your model will **underfit** (strong bias).
- Distribution space too large: your model will **overfit** (strong variance).

To avoid overfitting:

- propose **simpler models**! (smaller networks, parameter sharing between parts of the networks);
- **regularization**:

$$\theta^* = \arg\max_\theta \log L(\theta, \mathcal{D}_{train}) + R(\theta);$$

- monitor the overfitting with a **validation dataset**.

0.2 Application to fitting the parameters of an auto-regressive models

In the case of auto-regressive models,

$$p(\mathbf{x};\theta) = \prod_{i=1}^{m} p(\mathbf{x}_i|\mathbf{x}_{1:i-1};\theta_i).$$

Then

$$\begin{aligned} L(\theta, \mathcal{D}_{train}) &= \prod_{\mathbf{x}\sim\mathcal{D}_{train}} (p_\theta(\mathbf{x})) && (5) \\ &= \prod_{j=1}^{n}\prod_{i=1}^{m} p(\mathbf{x}_i^j|\mathbf{x}_{1:i-1}^j;\theta_i). && (6) \end{aligned}$$

and

$$\log L(\theta, \mathcal{D}_{train}) = \sum_{j=1}^{n}\sum_{i=1}^{m} \log p(\mathbf{x}_i^j|\mathbf{x}_{1:i-1}^j;\theta_i).$$

$$\nabla_{\theta_i} \log L(\theta, \mathcal{D}_{train}) = \sum_{j=1}^{n} \nabla_{\theta_i} \log p(\mathbf{x}_i^j|\mathbf{x}_{1:i-1}^j;\theta_i)$$

Depending on the implementation (shared parameters vs. non-shared parameters) you may estimate $\nabla_{\theta_i} \log L(\theta, \mathcal{D}_{train})$ independently.

Last, as we saw above, the stochastic gradient is used when the amount of training data is very large

$$\nabla_{\theta_i} \log L(\theta, \mathcal{D}_{train}) \approx nE_{\mathbf{x}\sim\mathcal{D}_{train}}[\nabla_{\theta_i} \log p(\mathbf{x}_i|\mathbf{x}_{1:i-1};\theta_i)].$$

0.3 Conditional generative models

In some situations, we are interested in generating some $\mathbf{x}$ **given the values of another variable** $\mathbf{y}$.

For example:

- generation of images "like the ones of ImageNet", conditionally to the class;
- generation of images given a caption.

We could do it by modeling:

$$p(\mathbf{x}, \mathbf{y}; \theta),$$

but in fact it won't be ncessary since the value of $\mathbf{y}$ will be fixed.

Instead:

$$p(\mathbf{x}|\mathbf{y}; \theta).$$

Hence we maximize:

$$L(\theta, \mathcal{D}_{train}) = \prod_{\{\mathbf{x},\mathbf{y}\}\sim\mathcal{D}_{train}} \log\left(p_\theta(\mathbf{x}|\mathbf{y})\right). \tag{7}$$

and we can follow eexactly the same process as above, except that all our components (giving the parameters of the conditional distributions) will now be depending on $\mathbf{y}$ (i.e., $\mathbf{y}$ is an "input" of the neural networks).

Conditional Image Generation with PixelCNN Decoders, A. Van Den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, K. Kavukcuoglu. Proceedings of the NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems.

Conditional PixelCNN:

- Overall, same principle as PixelCNN.
- Modified version of the masked CNN
- Written as a function of $\mathbf{y}$.

Each layer has the form

$$\mathbf{z} = \tanh(\mathbf{W}_{k,f} \circledast \mathbf{x} + \mathbf{V}_{k,f}\mathbf{y}) \odot \sigma(\mathbf{W}_{k,g} \circledast \mathbf{x} + \mathbf{V}_{k,g}\mathbf{y}).$$

In the case of classes, $\mathbf{y}$ can be encoded as a one-hot vector.

[From original paper]

[From original paper]