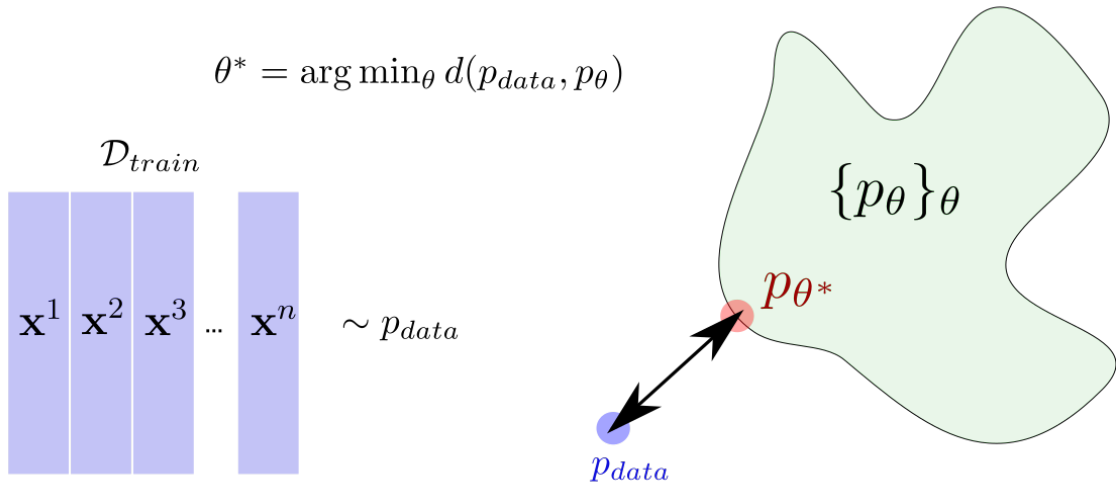
Latent variable models

October 28, 2022



0.1 Motivation for latent variable models

From: https://insidescience.org/

Among the variations observed in the face images $\mathbf{x} \in \mathbb{R}^m$ above, there are a lot that can "explained" **through some variables that are not present explicitly here** (but that, because of our training, we can identify rather precisely):

- Gender.
- Age.
- Ethnicity.
- Hair color.
- Having a beard.
- Pose with respect to the camera.

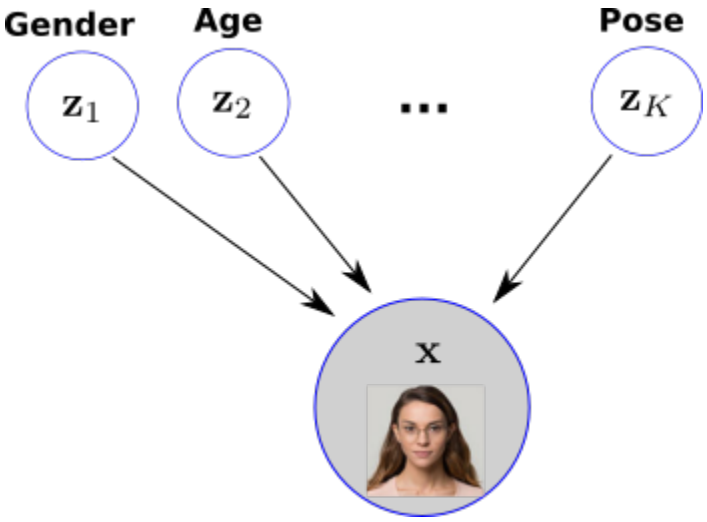
With all of these variables given, a mockup can be generated (cf. police TV series).

An idea is to make these variables that do not come with our original data **appear explicitly**: idea of **latent variable z**.

Note that if i give you the **true values of these variables z**, then you could probably produce a rather precise sketch of the face. Said in another way:

$$p(\mathbf{x}|\mathbf{z})$$

should be much **easier to handle than** $p(\mathbf{x})$ (and would be more informative).



In our situation of having to train a model to produce new images $\mathbf{x}$,

- $\mathbf{x}$ is **observed** during training $(\mathbf{x}^1, ..., \mathbf{x}^n)$
- $\mathbf{z}$ is not observed: **unsupervised representation learning**, we will try to "discover" these $\mathbf{z}$.

For example, we may use a neural network to model the parameters of $p(\mathbf{x}|\mathbf{z})$ and constraint $\mathbf{z} \in \mathbb{R}^K$ to some particular distribution:

- $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I}_K)$,
- $\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mu_\theta(\mathbf{z}), \mathbf{\Sigma}_\theta(\mathbf{z}))$.

We expect the features $\mathbf{z}$ to be meaningful, interpretable (but do not have **guarantee** about it).

Now, because we do not know anything about $\mathbf{z}$, the problem is ill-posed: to learn $p(\mathbf{x}|\mathbf{z})$, we need to know something about how these features are present in $\mathbf{x}$, i.e. $p(\mathbf{z}|\mathbf{x})$, but we don't have it! Chicken and egg...

0.2 Mixtures of Gaussians

An example of latent variable model: The **mixture of Gaussians**.

Assume we use K Gaussians to represent a distribution on some data $\mathbf{x}$:

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(\mathbf{z} = k)p(\mathbf{x}|\mathbf{z} = k)$$

- $\mathbf{z}$ is (in that case) a **categorical random variable** and takes values in $1...K$,
- $p(\mathbf{x}|\mathbf{z} = k)$ is a Gaussian distribution: μ_k, Σ_k in dimension m.

Parameters of this model:

- for each k, $\pi_k \triangleq p(\mathbf{z} = k)$ ($K - 1$ parameters),
- for each k, the parameters of the Gaussian conditional distribution μ_k, Σ_k ($mK + \frac{1}{2}m(m+1)K$ parameters).

Once you have trained this model, **sampling** is trivial:

1. Sample $\hat{\mathbf{z}}$ from the categorical distribution.
2. Sample $\hat{\mathbf{x}}$ from the Gaussian given by $\hat{\mathbf{z}}$: $\mu_{\hat{\mathbf{z}}}, \Sigma_{\hat{\mathbf{z}}}$.

What such a model does is to **cluster** the data $\mathbf{x}$ into different Gaussian components.

How to solve this problem (and train the model)?

Expectation-Maximization!

- Start with some **initialization** (e.g. $k-$means)
- Then iterate two phases, until **convergence**
  - For each data $\mathbf{x}^i$, compute the $p(\mathbf{z}^i = k|\mathbf{x}^i)$, with the Gaussian parameters and weights (priors) $p(\mathbf{z}^i = k|\mathbf{x}^i)$ fixed (kind of **classification of the data**),

$$p(\mathbf{z}^i = k|\mathbf{x}^i) \propto p(\mathbf{x}^i|\mathbf{z}^i = k; \mu_k, \mathbf{\Sigma}_k)p(\mathbf{z}^i = k).$$

  - Re-estimate the parameters of the GMM with the $p(\mathbf{z}^i = k)$ fixed,

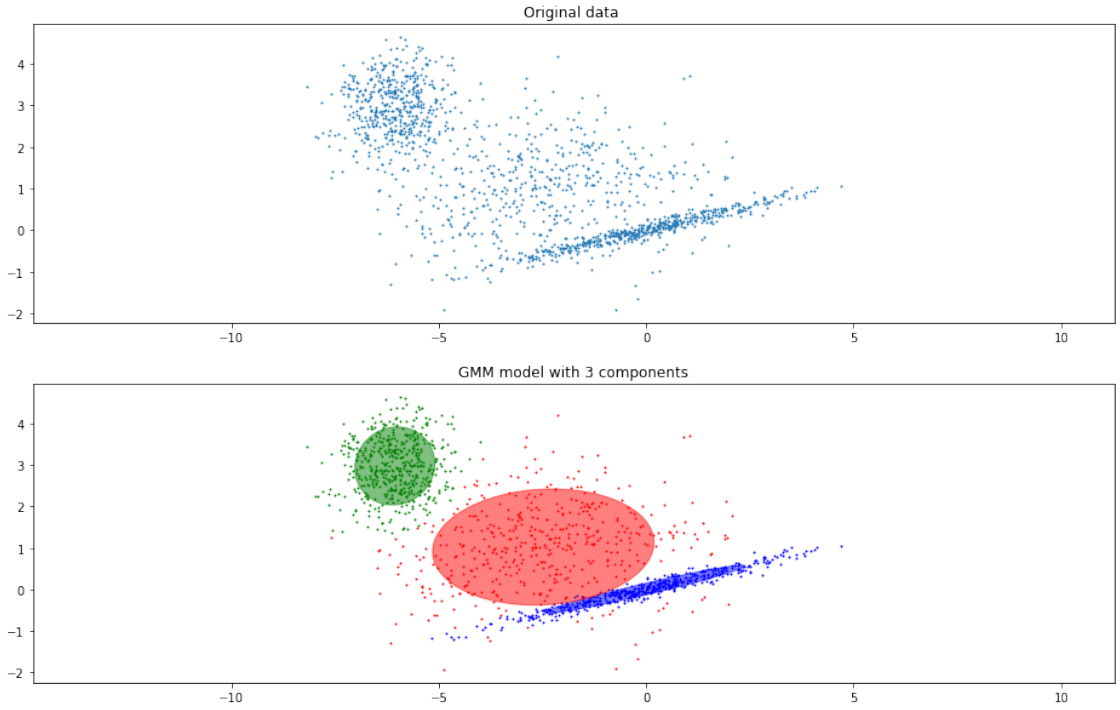
$$\mu_k = \frac{\sum_{i=1}^{n} p(\mathbf{z}^i = k|\mathbf{x}^i)\mathbf{x}^i}{\sum_{i=1}^{n} p(\mathbf{z}^i = k|\mathbf{x}^i)}, \tag{1}$$

$$\mathbf{\Sigma}_k = \frac{\sum_{i=1}^{n} p(\mathbf{z}^i = k|\mathbf{x}^i)(\mathbf{x}^i - \mu_k)(\mathbf{x}^i - \mu_k)^T}{\sum_{i=1}^{n} p(\mathbf{z}^i = k|\mathbf{x}^i)}, \tag{2}$$

$$p(\mathbf{z} = k) = \frac{1}{n}\sum_{i=1}^{n} p(\mathbf{z}^i = k|\mathbf{x}^i). \tag{3}$$

[1]:
```
import numpy as np
import itertools

from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl

from sklearn import mixture

# Number of samples per true component
n_samples = 500

# Generate data from a two-component model
np.random.seed(0)
C1 = np.array([[0.0, -0.1], [1.7, 0.4]])
C2 = np.array([[0.0, 1.0], [2.0, 0.1]])
X = np.vstack([
    np.dot(np.random.randn(n_samples, 2), C1),
    np.dot(np.random.randn(n_samples, 2), C2) + np.array([-2.5,1]),
    0.7 * np.random.randn(n_samples, 2) + np.array([-6, 3])])
n_components = 3


# Fit a Gaussian mixture with EM
gmm = mixture.GaussianMixture(n_components=n_components, covariance_type="full")
gmm.fit(X)

color_iter = itertools.cycle(["blue", "green", "red"])
bars = []

# Plot the BIC scores
plt.figure(figsize=(16, 10))

# Plot the data
ax = plt.subplot(2, 1, 1)
ax.scatter(X[:,0],X[:, 1], 0.8)
ax.axis('equal')
plt.title(
    f"Original data "
)

# Plot the GMM
ax = plt.subplot(2, 1, 2)
Z_ = gmm.predict(X)
for i, (mean, cov, color) in enumerate(zip(gmm.means_, gmm.covariances_,␣
 ↪color_iter)):
    v, w = linalg.eigh(cov)
```

```
    if not np.any(Z_ == i):
        continue
    plt.scatter(X[Z_ == i, 0], X[Z_ == i, 1], 0.8, color=color)

    # Plot an ellipse to show the Gaussian component
    angle = np.arctan2(w[0][1], w[0][0])
    angle = 180.0 * angle / np.pi  # convert to degrees
    v = 2.0 * np.sqrt(2.0) * np.sqrt(v)
    ell = mpl.patches.Ellipse(mean, v[0], v[1], 180.0 + angle, color=color)
    ell.set_clip_box(ax.bbox)
    ell.set_alpha(0.5)
    ax.add_artist(ell)

plt.title(
    f"GMM model with "
    f"{gmm.n_components} components"
)
ax.axis('equal')
plt.show()
```



Thinking in the generative capacity, note that we can see:

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(\mathbf{z} = k) p(\mathbf{x} | \mathbf{z} = k)$$

i.e. the **full model for our data** as a mixture of much simpler models (the Gaussians).

Now imagine a mixture of an **infinite number of Gaussians**, i.e. $\mathbf{z}$ being **continous and multivariate**:

- $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I}_K)$,
- $\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mu_\theta(\mathbf{z}), \mathbf{\Sigma}_\theta(\mathbf{z}))$

We will **learn** the means and covariances as **functions** of $\mathbf{z}$.

These functions will be implemented **as neural networks**:

$$\mu_\theta(\mathbf{z}) = h(\mathbf{A}\mathbf{z} + \mathbf{b}), \tag{4}$$
$$\mathbf{\Sigma}_\theta(\mathbf{z}) = diag(\exp(h'(\mathbf{C}\mathbf{z} + \mathbf{d}))), \tag{5}$$

with h an activation function and:

$$\theta \triangleq (\mathbf{A}, \mathbf{B}, \mathbf{c}, \mathbf{d}).$$

As in the case of the mixture of Gaussians, even though each $p(\mathbf{x}|\mathbf{z})$ is rather **simple** (a Gaussian), the resulting marginal $p(\mathbf{x})$ may be very complex/flexible.

0.3 Variational inference

Denote the **joint distribution of x and z** as $p(\mathbf{x}, \mathbf{z}; \theta) = p(\mathbf{x}|\mathbf{z}; \theta)p(\mathbf{z})$.

Assume we have a dataset $\mathcal{D}$ made by data points $\mathbf{x}^i$, for $i = 1...n$.

Training the model can be set up as **maximizing the likelihood of our data**:

$$\theta^* = \arg\max_\theta p(\mathbf{x}^1, ..., \mathbf{x}^n; \theta) = \arg\max_\theta \prod_i p(\mathbf{x}^i; \theta).$$

The optimization problem described above can be re-written as:

$$\theta^* = \arg\max_\theta \log \prod_i p(\mathbf{x}^i; \theta), \tag{6}$$
$$= \arg\max_\theta \sum_i \log p(\mathbf{x}^i; \theta), \tag{7}$$
$$= \arg\max_\theta \sum_i \log \int_{\mathbf{z}} p(\mathbf{x}^i, \mathbf{z}; \theta) d\mathbf{z}. \tag{8}$$

The term $\int_{\mathbf{z}} p(\mathbf{x}^i, \mathbf{z}; \theta) d\mathbf{z}$ is also called **marginal likelihood**.

Evaluating $\log \int_{\mathbf{z}} p(\mathbf{x}^i, \mathbf{z}; \theta) d\mathbf{z} = \log \int_{\mathbf{z}} p(\mathbf{x}^i|\mathbf{z}; \theta)p(\mathbf{z}) d\mathbf{z}$ is in general **very, very difficult** (no closed form)!

Imagine even the simpler case of **discrete latent features**,

$$\mathbf{z} \in [1, ..., L]^K,$$

and evaluating the integral would be as a sum with L^K terms.

In any case, training and optimizing θ will involve computing the gradients:

$$\nabla_\theta \log \int_{\mathbf{z}} p(\mathbf{x}^i, \mathbf{z}; \theta) d\mathbf{z}.$$

Let us focus on these **marginal likelihoods**:

$$p(\mathbf{x}^i; \theta) = \int_{\mathbf{z} \in \mathcal{Z}} p(\mathbf{x}^i, \mathbf{z}; \theta) d\mathbf{z}.$$

Note that this **marginal likelihood** can be seen as an **expectation** value. Suppose for example that the support of $\mathbf{z}$ is finite:

$$p(\mathbf{x}^i; \theta) = |\mathcal{Z}| \int_{\mathbf{z} \in \mathcal{Z}} \frac{1}{|\mathcal{Z}|} p(\mathbf{x}^i, \mathbf{z}; \theta) d\mathbf{z} = |\mathcal{Z}| E_{\mathbf{z}}[p(\mathbf{x}^i, \mathbf{z}; \theta)].$$

We could use **Monte-Carlo** to estimate the term on the right:

1. Sample S samples $\mathbf{z}^{(1)}, ...\mathbf{z}^{(S)}$ uniformly at random.
2. Approximate the expectation through the average value of the function over the samples:

$$p(\mathbf{x}^i; \theta) = |\mathcal{Z}| \frac{1}{S} \sum_{s=1}^{S} p(\mathbf{x}^i, \mathbf{z}^{(s)}; \theta).$$

But, in practice, it will **not work**!

It is very improbable to sample (by chance) a $\mathbf{z}^{(s)}$ such that $p(\mathbf{x}^i, \mathbf{z}^{(s)}; \theta)$ is **high**; most of the samples will give **very low values** and the estimation of the expected value will be bad.

We need a **more informative distribution** for $\mathbf{z}$.

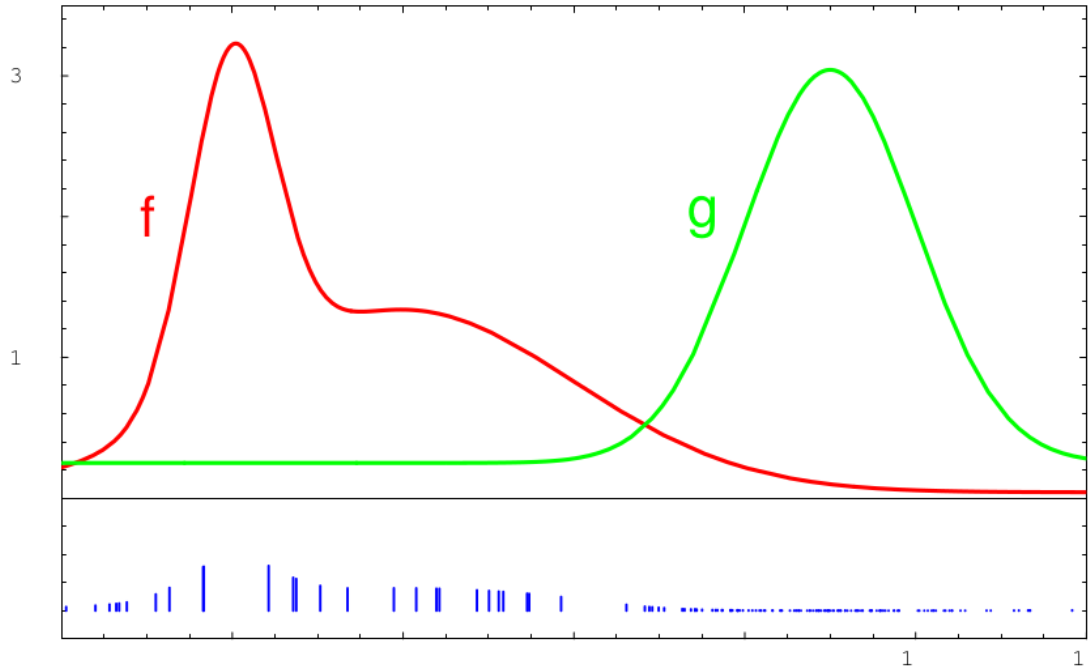
One common trick in these situations is to use **importance sampling**. Importance sampling involves sampling from **another distribution** $q(\mathbf{z})$ (instead of the uniform distribution), from which sampling is easy. Then we can re-write the expectation as

$$p(\mathbf{x}^i; \theta) = |\mathcal{Z}| \int_{\mathbf{z} \in \mathcal{Z}} q(\mathbf{z}) \frac{1}{|\mathcal{Z}| q(\mathbf{z})} p(\mathbf{x}^i, \mathbf{z}; \theta) d\mathbf{z} = E_{\mathbf{z} \sim q}[\frac{1}{q(\mathbf{z})} p(\mathbf{x}^i, \mathbf{z}; \theta)]$$

Monte Carlo, again: 1. Sample S samples $\mathbf{z}^{(1)}, ...\mathbf{z}^{(S)}$ from $q(\mathbf{z})$. 2. Approximate expectation with **weighted** sample average

$$p(\mathbf{x}^i; \theta) = \frac{1}{S} \sum_{s=1}^{S} \frac{p(\mathbf{x}^i, \mathbf{z}^{(s)}; \theta)}{q(\mathbf{z}^{(s)})}.$$

Intuitively, how do we need to choose q so that we limit the effect commented above on "normal" Monte-Carlo?



$$E_{\mathbf{z}\sim f}\left[h(\mathbf{z})\right] = \int_{\mathbf{z}} h(\mathbf{z}) f(\mathbf{z}) d\mathbf{z} = \int_{\mathbf{z}} h(\mathbf{z}) \frac{f(\mathbf{z})}{g(\mathbf{z})} g(\mathbf{z}) d\mathbf{z}.$$

Ideally:

$$q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x}^i),$$

but we **do not** have access to this posterior on $\mathbf{z}$.

Then,

$$\log p(\mathbf{x}^i;\theta) = \log \int_{\mathbf{z}\in\mathcal{Z}} q(\mathbf{z}) \frac{p(\mathbf{x}^i,\mathbf{z};\theta)}{q(\mathbf{z})} d\mathbf{z} = \log\left(E_{\mathbf{z}\sim q}\left[\frac{p(\mathbf{x}^i,\mathbf{z};\theta)}{q(\mathbf{z})}\right]\right)$$

can be approximated by Monte Carlo as

$$\log p(\mathbf{x}^i;\theta) \approx \log\left(\frac{1}{S}\sum_{s=1}^{S} \frac{p(\mathbf{x}^i,\mathbf{z}^{(s)};\theta)}{q(\mathbf{z}^{(s)})}\right).$$

In most implementations, $S = 1$ (to be efficient). Hence, with just one sample:

$$\log p(\mathbf{x}^i;\theta) \approx \log\left(\frac{p(\mathbf{x}^i,\mathbf{z}^{(1)};\theta)}{q(\mathbf{z}^{(1)})}\right).$$

But

$$\log\left(E_{\mathbf{z}^{(1)}\sim q}\left[\frac{p(\mathbf{x}^i,\mathbf{z}^{(1)};\theta)}{q(\mathbf{z}^{(1)})}\right]\right) \neq E_{\mathbf{z}^{(1)}\sim q}\left[\log\left(\frac{p(\mathbf{x}^i,\mathbf{z}^{(1)};\theta)}{q(\mathbf{z}^{(1)})}\right)\right].$$

We want the **first one**.

Jensen Inequality: since log is concave, then for any **positive** function $f(\mathbf{z})$

$$\log\left(E_{\mathbf{z}\sim q}\left[f(\mathbf{z})\right]\right) = \log(\int_{\mathbf{z}} q(\mathbf{z})f(\mathbf{z})d\mathbf{z}) \geq \int_{\mathbf{z}} q(\mathbf{z})\log(f(\mathbf{z}))d\mathbf{z}.$$

Choosing $f(\mathbf{z}) = \frac{p(\mathbf{x}^i,\mathbf{z})}{q(\mathbf{z})}$,

$$\log\left(E_{\mathbf{z}\sim q}\left[\frac{p(\mathbf{x}^i,\mathbf{z};\theta)}{q(\mathbf{z})}\right]\right) \geq E_{\mathbf{z}\sim q}\left[\log\left(\frac{p(\mathbf{x}^i,\mathbf{z};\theta)}{q(\mathbf{z})}\right)\right]$$

The term on the right is called **Evidence Lower Bound (ELBO)**. Remind that this holds **for any** $q(\mathbf{z})$.

Summing up

$$\begin{aligned}
\log p(\mathbf{x}^i;\theta) \geq & \quad E_{\mathbf{z}\sim q}\left[\log\left(\frac{p(\mathbf{x}^i,\mathbf{z};\theta)}{q(\mathbf{z})}\right)\right], && (9)\\
= & \quad \int_{\mathbf{z}} q(\mathbf{z})\log\left(\frac{p(\mathbf{x}^i,\mathbf{z};\theta)}{q(\mathbf{z})}\right)d\mathbf{z}, && (10)\\
= & \quad \int_{\mathbf{z}} q(\mathbf{z})\log\left(p(\mathbf{x}^i,\mathbf{z};\theta)\right)d\mathbf{z} - \int_{\mathbf{z}} q(\mathbf{z})\log\left(q(\mathbf{z})\right)d\mathbf{z}, && (11)\\
= & \quad \int_{\mathbf{z}} q(\mathbf{z})\log\left(p(\mathbf{x}^i,\mathbf{z};\theta)\right)d\mathbf{z} + H(q). && (12)
\end{aligned}$$

Will denote it as $\mathcal{E}(\mathbf{x}^i;\theta,q)$.

Interpretation as an optimization problem:

- The first term represents an energy that encourages q to **concentrate the probability mass where the model parametrized by θ has high evidence** $p(\mathbf{x}^i,\mathbf{z};\theta)$.
- The second term is the **entropy of** q. By making it higher we encourage the **spreading of the probability mass**. This is important because through the first term alone could concentrate the mass to one location (the mode).

Now: note that to get **equality** instead of an **inequality**, we should have:

$$q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x}).$$

You can check that in that case the equality holds. This corresponds to the intuition given above: The $\mathbf{z}$ should be sampled as likely $\mathbf{z}$ given $\mathbf{x}$.

A general result that shows the same:

$$\log p(\mathbf{x}^i;\theta) = \mathcal{E}(\mathbf{x}^i;\theta,q) + D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x}^i;\theta)).$$

When $q(\mathbf{z})$ is **close** to $p(\mathbf{z}|\mathbf{x};\theta)$, **the ELBO $\mathcal{E}(\mathbf{x}^i;\theta,q)$ gets close to the true log-likelihood**.

Now, imagine that we choose $q(\mathbf{z};\phi)$ as a distribution **parameterized through parameters ϕ** (example: from a second **neural network**).

$$\log p(\mathbf{x}^i;\theta) \geq \int_{\mathbf{z}\sim q} q(\mathbf{z};\phi)\log p(\mathbf{x}^i,\mathbf{z};\theta)d\mathbf{z} + H(q(\mathbf{z};\phi)) \triangleq \mathcal{E}(\mathbf{x}^i;\theta,\phi)$$

(this is the ELBO) and

$$\mathcal{L}(\mathbf{x}^i;\theta) \triangleq \log p(\mathbf{x}^i;\theta) = \mathcal{E}(\mathbf{x}^i;\theta,\phi) + D_{KL}(q(\mathbf{z};\phi)\|p(\mathbf{z}|\mathbf{x}^i;\theta))$$

As $q(\mathbf{z};\phi)$ approximate well the posterior $p(\mathbf{z}|\mathbf{x}^i;\theta)$, the bound given ELBO will be close to $\log p(\mathbf{x}^i;\theta)$, which means that **maximizing the ELBO will be very similar to maximizing the marginal liklihood**. We will use both ϕ and θ to maximize the ELBO.

This process is called **variational inference**.