Generative Models

Lecture 2

We saw last time how we can approximate an intractable posterior distribution $p(\mathbf{z}|\mathcal{D})$ where $\mathcal{D}$ is a data set of samples from some unknown distribution p.

$\mathbf{z}$ is a latent variable such that the conditional distribution $p(\mathbf{x}|\mathbf{z})$ is known.

We can put a prior distribution on $\mathbf{z}$, $p_{prior}(\mathbf{z})$.

The likelihood

$$p(\mathcal{D}|\mathbf{z}) = \prod_{\mathbf{x}\in\mathcal{D}} p(\mathbf{x}|\mathbf{z})$$

is known and so by Bayes' formula

$$p(\mathbf{z}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{z})}{p(\mathcal{D})} p_{prior}(\mathbf{z})$$

The probability $p(\mathcal{D})$ is intractable so we can't actually compute the posterior $p(\mathbf{z}|\mathcal{D})$ from Bayes' formula.

The idea of *Variational Inference* is to approximate the intractable posterior distribution with a distribution from some family of parametrized distributions

$$\{q_\phi\}_{\phi \in \Phi}$$

Here *approximate* means finding a ϕ_0 that minimizes the KL-Divergence

$$\phi_0 = \underset{\phi}{\operatorname{argmin}}\ D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z}|\mathcal{D}))$$

We saw last time that this could be reduced to maximizing the $ELBO$

$$\begin{aligned}
&\mathbb{E}_{q_\phi}\left(\log p(\mathcal{D}|\mathbf{z})\right) - D_{KL}(q_\phi(\mathbf{z})||p_{prior}(\mathbf{z})) \\
&= \int \log p(\mathcal{D}|\mathbf{z}) q_\phi(\mathbf{z}) d\mathbf{z} + \int \log \frac{p_{prior}(\mathbf{z})}{q_\phi(\mathbf{x})} q_\phi(\mathbf{z}) d\mathbf{z}
\end{aligned}$$

How do we find minimum of a function $f(\theta)$?

We try to find a stationary point i.e. a point where

$$\nabla_\theta f(\theta) = 0$$

The equation $\nabla_\theta f(\theta) = 0$ is generally not solvable so we have to resort to approximation methods

The method of *gradient descent* is an algorithm to find a sequence converging to a stationary point. This algorithm was already known to Newton

We use the Taylor polynomial of degree 1 to aproximate by a linear function

$$f(\phi + \Delta\phi) \approx f(\phi) + \nabla_\phi f(\phi) \cdot \Delta\phi$$

In which direction should we move from ϕ to get the largest change in f i.e. in which direction should $\Delta\phi$ point in order for

$$||f(\phi + \Delta\phi) - f(\phi)|| \approx |\nabla_\phi f(\phi) \cdot \Delta\phi|$$

to be as large as possible

The *Cauchy-Schwartz Inequality* tells us that for any two vectors $\mathbf{u}$ and $\mathbf{v}$

$$|\mathbf{u} \cdot \mathbf{v}| \leq ||\mathbf{u}|| \, ||\mathbf{v}||$$

with equality if and only if $\mathbf{u}$ and $\mathbf{v}$ are parallel i.e. $\mathbf{v} = \lambda \mathbf{u}$ for some number λ

So in our case we should choose the direction

$$\Delta\phi = \alpha\nabla_\phi f(\phi)$$

Gradient Descent

Start with an arbitray ϕ_{init} and then construct a sequence

$$\phi_1 = \phi_{init} + \alpha\nabla_\phi f(\phi_{init}), \phi_2 = \phi_1 + \alpha\nabla_\phi f(\phi_1), \phi_3 = \phi_2 + \alpha\nabla_\phi f(\phi_2), \ldots$$

If α is sufficiently small this sequence converges to a stationary point

Monte Carlo Estimation

If p is some distribution and f a function.

How can we estimate the integral

$$\int f(\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

This integral is precisely

$$\mathbb{E}_p(f(\mathbf{z}))$$

and we can estimate this expectation by averaging over samples $\{\mathbf{z}_i\}_{1,2,\ldots,N}$ from p.

$$\mathbb{E}_p(f(\mathbf{z})) \approx \frac{1}{N}\sum_i f(\mathbf{z}_i)$$

We can now approximate the gradient by espressing it as an expectation and then use Monte Carlo

$$\nabla_\phi \Big(\int \log p(\mathcal{D}|\mathbf{z}) q_\phi(\mathbf{z}) d\mathbf{z} + \int \log \frac{p_{prior}(\mathbf{z})}{q_\phi(\mathbf{z})} q_\phi(\mathbf{z}) d\mathbf{z} \Big)$$

$$= \int \log p(\mathcal{D}|\mathbf{z}) \nabla_\phi q_\phi(\mathbf{z}) d\mathbf{z} + \int \log p_{prior}(\mathbf{z}) \nabla_\phi q_\phi(\mathbf{z}) d\mathbf{z} - \int \nabla_\phi q_\phi(\mathbf{z}) d\mathbf{z}$$

$$- \int \log q_\phi \nabla_\phi q_\phi(\mathbf{z}) d\mathbf{z}$$

Remark that $\nabla_\phi \log q_\phi(\mathbf{z}) = \dfrac{\nabla_\phi q_\phi(\mathbf{z})}{q_\phi(\mathbf{z})}$ so $\nabla_\theta q_\phi(\mathbf{z}) = \nabla_\phi \log q_\phi(\mathbf{z}) q_\phi(\mathbf{z})$

If we substitute this the expression becomes

$$\int (\log(p(\mathcal{D}|\mathbf{z})p_{prior}(\mathbf{z})) - \log q_\phi(\mathbf{z}) - 1)\nabla_\phi \log q_\phi(\mathbf{z}) q_\phi(\mathbf{z}) d\mathbf{z}$$

$$= \mathbb{E}_{q_\phi}((\log(p(\mathcal{D}|\mathbf{z})p_{prior}(\mathbf{z})) - \log q_\phi(\mathbf{z}) - 1)\nabla_\phi \log q_\phi(\mathbf{z}))$$

This expectation can now be estimated using Monte Carlo

Example

We consider a *GMM = Gaussian Mixture Model.*

It is given by n Gaussians

$$\mathcal{N}(\mathbf{x}|\mu_i, \Sigma_i), i = 1, 2, \ldots, n$$

where the parameters $\{(\mu_i, \Sigma_i)\}$ are unknown.

We also have an unknown probability distribution π over the set $\{1, 2, \ldots, n\}$

Samples from the model are generated by first sampling a k from π and then sampling a point $\mathbf{x}$ from the Gaussian $\mathcal{N}(\mu_k, \Sigma_k)$

Given a set of samples $\mathcal{D}$, estimate the parameters $\{\{(\mu_i, \Sigma_i)\}, \pi\}$

In this case we know something about the data distribution.

It is a mixture of Gaussians p_θ with parameters $\theta = \{\{(\mu_i, \Sigma_i)\}, \pi\}$ and the ML estimate of the parameters is

$$\theta_0 = \underset{\theta}{\text{argmax}} \ \log p_\theta(\mathcal{D})$$

While we can't do this directly, we can try to maximize a lower bound, the $ELBO = \mathbb{E}_{q_\phi}\left(\log p_\theta(\mathcal{D}|\mathbf{z})\right) - D_{KL}(q_\phi(\mathbf{z})||p_{prior}(\mathbf{z}))$

The latent variables are now the unknown parameters θ so we now view the parameters $\mathbf{z} = \theta$ themselves as random variables.

Thus $q_\phi(\mathbf{z})$ is $q_\phi(\theta)$ and $p(\mathcal{D}|\mathbf{z})$ is $p(\mathcal{D}|\theta)$ and the $ELBO$ is

$$\mathbb{E}_{q_\phi(\theta)} \left(\log p(\mathcal{D}|\theta) - \log \frac{p_{prior}(\theta)}{q_\phi(\theta)} \right)$$

Maximizing this quantity with respect to both the variational parameters ϕ and the Gaussian mixture parameters we need to compute

$$\nabla_{\theta,\phi} \mathbb{E}_{q_\phi} \left(\log p_\theta(\mathcal{D}|\theta) - \log \frac{p_{prior}(\theta)}{q_\phi(\theta)} \right)$$

We shall use the Python package `pyro` which makes it very easy to do variational inference.

```
import os
from collections import defaultdict
from matplotlib import pyplot
import torch
import numpy as np
import scipy.stats
from torch.distributions import constraints
import pandas as pd

import pyro
import pyro.distributions as dist
from pyro import poutine
from pyro.infer.autoguide import AutoDelta,AutoNormal
from pyro.optim import Adam
from pyro.infer import SVI, TraceEnum_ELBO, config_enumerate, infer_discrete

import random

```

Generating the Dataset

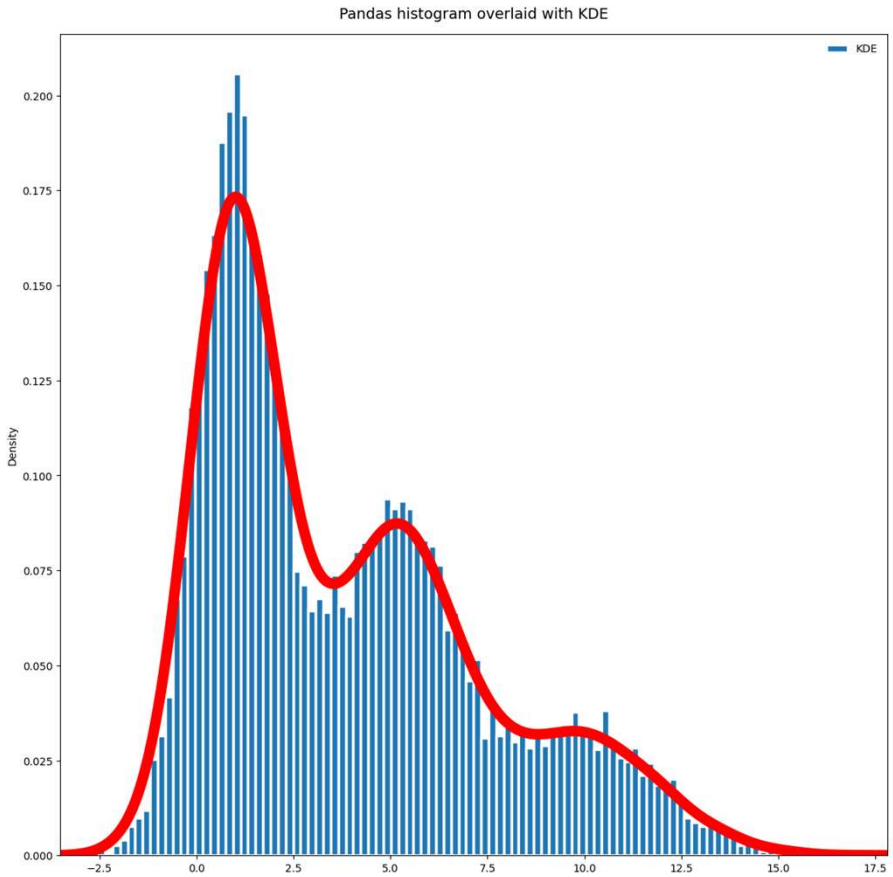
These are the hidden (latent) variables θ

We generate 10000 samples from the distribution p_θ

```
pi = [1/2,1/3,1/6]
mu = [1.0,5.0,10.0]
sigma2 = [1.0,1.5,2.0]
```

```
n_samples = 10000
data = []

#first we generate n_samples samples from the distribution pi i.e. a sequece of the form {1,3,
cluster_assignments = np.random.choice(3,n_samples,p=pi)


# for each of the indices in cluster_assignments we sample from the Normal(mu[idx],sigma2[idx]
for assignment in cluster_assignments:
    mean = mu[assignment]
    var = sigma2[assignment]
    sample = np.random.normal(mean,var)
    data.append(sample)
```

Histogram of samples



In `pyro` the `model` is the data generating function consisting of the prior $p_{prior}(\theta)$ and the conditional distribution $p(\mathbf{x}|\theta)$ (the likelihood)

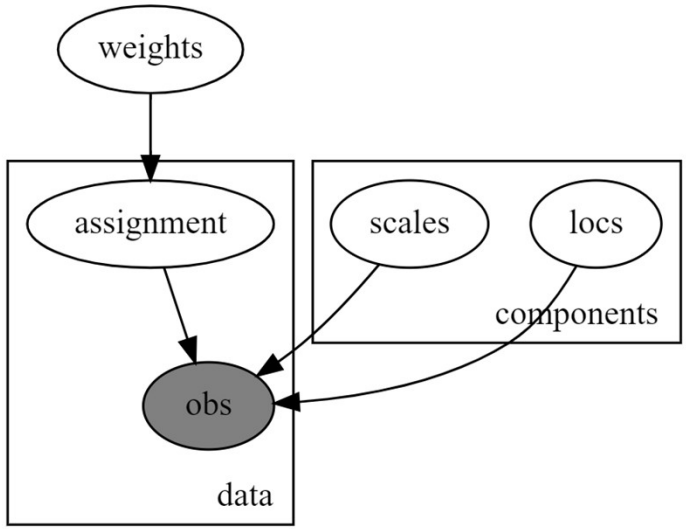
The prior on π is the Dirichlet distribution $Dir(0.5, 0.5, 0.5)$

The prior on each of the 3 means (called locations in pyro) is $\mathcal{N}(0, 10)$ and the prior on each of the 3 variances (called scales in pyro) is $\exp\mathcal{N}(0, 2)$

The `assignment` is a sample from the distribution π, which in turn is sampled from $Dir(0.5, 0.5, 0.5)$ This selects one of the Gaussians and we sample from that. This is then compared to the actual data (`obs`)

```
def model(data):
    samples = []
    # the distribution pi is sampled from a Dirichlet distribution. The Dirichlet is a distribu
    # probability simplex i.e. all vectors $x_1,x_2,...,x_n$ with x_i > 0 and summing up to 1
    # each of the parameters are stored in the pyro.param_store as items in a dict, as we can s
    # In pyro distributions are basically characterized by their samples
    weights=pyro.sample('weights',dist.Dirichlet(0.5*torch.ones(K)))

    #the pyro.plate specifies an array of independent distributions. Here the distribution for
    #Gaussian with mean 0 and variance 10, the variances have a LogNormal distribution since th
    # these distributions are stored in an array in the param_store named 'components'
    with pyro.plate('components',K):
        #the prior distriburion of the means
        locs = pyro.sample('locs',dist.Normal(0.,10.))
        # the prior distribution of the variances
        scales = pyro.sample('scales',dist.LogNormal(0.,2.))

    #here is the likelihood, for each data point an assignment to a Gaussian and then it is a s
    # first a sample from the distribution pi, which in turn is a sample from the Dirichlet dis
    # then a sample from the corresponding Gaussian with mean and variance sampled from the app
    with pyro.plate('data',len(data)):
        assignment = pyro.sample('assignment',dist.Categorical(weights))
        pyro.sample('obs',dist.Normal(locs[assignment],scales[assignment]),obs=data)
```

Graphical Representation of the Model



The *Dirichlet Distribution* of dimension n is a distribution over the probability simplex in $\mathbb{R}^n$

$$\{(x_1, x_2, \ldots, x_n) \,|\, x_i > 0 \text{ and } x_1 + x_2 + \ldots x_n = 1\}$$

It has n positive parameters $\alpha_1, \alpha_2, \ldots \alpha_n$ and the density is

$$Dir(\alpha_1, \alpha_2, \ldots, \alpha_n)(\mathbf{x}) = B(\alpha_1, \alpha_2, \ldots, \alpha_n) \prod_i x_i^{\alpha_i - 1}$$

The normalizing constant (to make the integral of the density function = 1) is

$$B(\alpha_1, \alpha_2, \ldots, \alpha_n) = \frac{\prod_i \Gamma(\alpha_i)}{\Gamma(\sum_i \alpha_i)}$$

Γ is the Gamma function

$$\Gamma(\alpha) = \int_0^\infty t^{\alpha - 1} \exp(-t) dt$$

One problem with this model is that the π distribution is discrete so we can't differentiate it.

For each point $\mathbf{x} \in \mathcal{D}$ we sample an $i \in \{0, 1, 2\}$ from π and the likelihood is $\mathcal{N}(\mathbf{x}|\mu_i, \sigma_i)\pi(i)$ so to marginalize out π we would have to sum over 3 possible assignments for each $\mathbf{x}$ in $\mathcal{D}$. In this case $3^{10,000}$ terms.

The likelihood function with π marginalized out is

$$p(\mathbf{x}|\{\mu_i, \sigma_i\}_{i=1,2,3}, \pi) = \sum_{i=0,1,2} \mathcal{N}(\mathbf{x}|\mu_i, \sigma_i)\pi(i)$$

so

$$p(\mathcal{D}|\{\mu_i, \sigma_i\}_{i=0,1,2}, \pi) = \prod_{\mathbf{x}\in\mathcal{D}} \sum_{i=0,1,2} \mathcal{N}(\mathbf{x}|\mu_i, \sigma_i)\pi(i)$$

This is a sum with $3^{len(dataset)}$ terms which is obviously intractable

One of the great features of `pyro` is that it can deal with this problem through a technique called *variable elimination* (which is beyond the scope of the course)

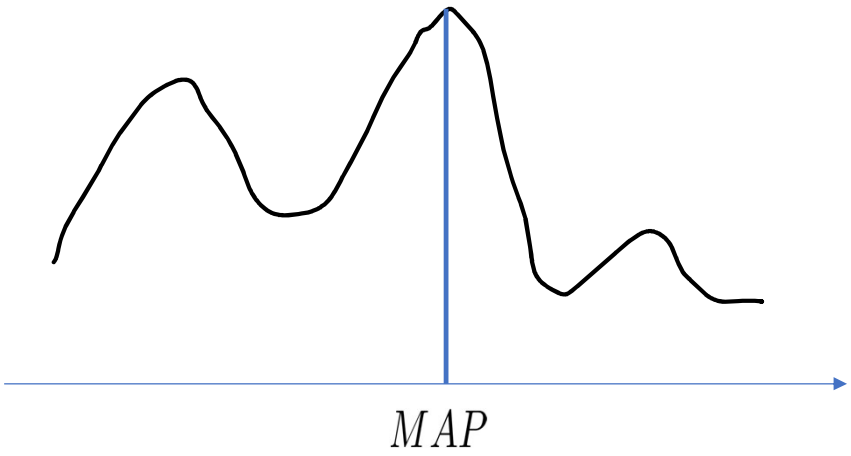
So using a special loss function `TraceEnum_ELBO` the program will automatically marginalize out the discrete variables during the training.

We have to tell `pyro` that we want to this. so we *decorate* the model definition with a `@config_enumerate`|

```
# we first have to define the pyro mode
# the distribution over {0,1,2}, the me
# the prior is a product of the distrib
K=3

@config_enumerate
def model(data):
    samples = []
    # the distribution pi is sampled fr
    # probability simplex i.e. all vect
```

We want the means, the variances and the weights that fit the data as well as possible. We may not really be interested in the distribution $p(\{\mu_i, \sigma_i^2\}, \pi)$ but only in the values that maximize the probability.

This is known as the *MAP Maximum Apostiori Probability.*



This means that we can take our variational distribution to be a δ-function i.e. all it's probability is concentrated in one point.

`pyro` can automatically construct a variational distribution that is a δ-function using the `AutoDelta` class

In `pyro` all the parameters have names and the values are stored in a `dict` called the `pyro.param_store`. To see named parameters we can call `pyro.get_param_store().named_parameters()`

The training works by doing *gradient ascent* to maximize the $ELBO$

The gradients

$$\nabla_{\theta,\phi}\mathbb{E}_{q_\phi}(\log p_\theta(\mathcal{D},\theta) - \log q_\phi(\theta))$$

can be computed in two steps

$$\nabla_\theta \mathbb{E}_{q_\phi}(\log p_\theta(\mathcal{D},\theta) - \log q_\phi(\theta)) = \mathbb{E}_{q_\phi}\left(\nabla_\theta \log p_\theta(\mathcal{D},\theta) - \frac{\nabla_\theta q_\phi(\theta)}{q_\phi(\theta)}\right)$$

Differentiating with respect to θ is straightforward since we always have

$$\nabla_\theta \mathbb{E}_{q_\phi}(f(\theta)) = \mathbb{E}_{q_\phi}(\nabla_\theta f(\theta))$$

Differentiating with respect to ϕ is slightly more complicated.

We can use Monte Carlo to estimate the gradients of the form

$$\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}}(f_\phi(\mathbf{z})) = \nabla_\phi \int f_\phi(\mathbf{z}) q_\phi(\mathbf{z}) d\mathbf{z}$$

as we did before but in certain cases there is an easier way. This is the socalled *reparametrization trick*

This trick works in particular if q_ϕ is Gaussian (but also for most exponential family distributions)

The Reparametrization Trick (for Gaussians)

If

$$q_\theta = \mathcal{N}(\mu(\phi), \Sigma(\phi))$$

then

$$q_\phi(\mathbf{z}) = \frac{1}{(2\pi)^{n/2} \det \Sigma(\phi)^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{z} - \mu(\phi)) \cdot \Sigma(\phi)^{-1} \cdot (\mathbf{z} - \mu(\phi))^T\right)$$

We can write

$$\Sigma(\phi) = \sqrt{\Sigma(\phi)}^T \cdot \sqrt{\Sigma(\phi)} \text{ (Cholesky decomposition)}$$

Now put

$$\mathbf{u} = (\mathbf{z} - \mu(\phi)) \cdot \sqrt{\Sigma(\phi)}^{-1}$$

Then

$$\mathbf{z} = \mu(\phi) + \mathbf{u} \cdot \sqrt{\Sigma(\phi)}$$

Changing variables in the integral we get

$$\int f_\phi(\mathbf{z}) q_\phi(\mathbf{z}) d\mathbf{z} = \int f(\mu(\phi) + \mathbf{u} \cdot \sqrt{\Sigma(\phi)}) q_\phi(\mu(\phi) + \mathbf{u} \cdot \sqrt{\Sigma(\phi)}) \det \sqrt{\Sigma(\phi)} d\mathbf{u}$$

But

$$q_\phi(\mu(\phi) + \mathbf{u} \cdot \sqrt{\Sigma(\phi)}) \det \sqrt{\Sigma(\phi)} = \frac{1}{(2\pi)^{n/2} \sqrt{\det \Sigma(\phi)}} \exp(-\frac{1}{2} \mathbf{u} \cdot \mathbf{u}^T) \det \sqrt{\Sigma(\phi)}$$

$$= \frac{1}{(2\pi)^{n/2}} \exp(-\frac{1}{2} \mathbf{u} \cdot \mathbf{u}^T)$$

$$= \mathcal{N}(\mathbf{u}|0, I)$$

So the integral is now

$$\int f(\mu(\phi) + \mathbf{u} \cdot \sqrt{\Sigma(\phi)})\mathcal{N}(\mathbf{u}|0, I)d\mathbf{u} = \mathbb{E}_{\mathcal{N}(0,I)}(f(\mu(\phi) + \mathbf{u} \cdot \sqrt{\Sigma(\phi)})$$

Now we are differentiating an expectation over a distribution that does not depend on ϕ so we can just move the ∇_ϕ into the expectation

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z})}(f_\phi(\mathbf{z})) &= \nabla_\phi \mathbb{E}_{\mathcal{N}(\mathbf{u}|0,I)}\left(f(\mu(\phi) + \mathbf{u} \cdot \sqrt{\Sigma(\phi)})\right) \\ &= \mathbb{E}_{\mathcal{N}(\mathbf{u}|0,I)}\left(\left(\nabla_\phi(f(\mu(\phi) + \mathbf{u} \cdot \sqrt{\Sigma(\phi)})\right)\right)\end{aligned}$$

`pyro` will figure out whether it can do the reparametrization and approximate the gradients using Monte Carlo

This is done by the `SVI` (Stochastic Variational Inference) class

```
svi = SVI(model,guide,optim,loss=elbo)
```

The `SVI` object will take care of computing the gradients using Monte Carlo and do the gradient ascent step and compute the new $ELBO$

```
for i in range(700):
    loss = svi.step(data)
```

So we will use `AutoDelta` to make a `guide` from the family of δ-distributions over the parameters. The parameters ϕ for a δ-distribution q_ϕ is just a particular point $\{\mu_i, \sigma_i^2\}_{i=0,1,2}, \pi$ in the parameter space

To start the gradient ascent we start at some random point in the parameter. We are going to start with a particular parameter set:

$\mu =$ tensor consisting of K randomly chosen data points, $\sigma^2 =$ tensor$[1, 1, 1], \pi = \{1/3, 1/3, 1/3\}$

The starting point is an input to the `AutoDelta` class

The `AutoDelta` defines sites in the `param_store`
`AutoDelta.weights, AutoDelta.locs,AutoDelta.scales`
The `init_loc_fn function, set the values to the values above`

```
def init_loc_fn(site):
    if site['name'] == 'weights':
        return torch.ones(K)/K
    if site['name'] == 'scales':
        return torch.ones(K)
    if site['name'] == 'locs':
        return data[torch.randint(0,10000,size=(K,))]
    raise ValueError(site['name'])
```

This function takes in an integer `seed` which is used to initialize the random number generator.

If a `seed` is defined the random number generator is deterministic

```
def initialize(seed):

    pyro.set_rng_seed(seed)

    pyro.clear_param_store()

    guide = AutoDelta(poutine.block(model,expose=['weights','locs','scales']),
                                init_loc_fn =  init_loc_fn )
    svi = SVI(model,guide,optim,loss=elbo)
    return guide, svi, svi.loss(model,guide,data)
```

It initializes the `pyro` random number generator with the seed.

It initializes the `guide` and specifies that it exposes the parameters and it sets the initial values using the `init_loc_fn` function

It instantiates the `SVI`object with the `model`, the `guide`,the optimizer `optim` and the loss function `elbo=TraceEnum_ELBO()`

Once these objects have been instantiated we can run the *training loop* which computes the Monte Carlo approximate gradients and does the gradient ascent step. All automatic using the `svi.step` method

The first 3 lines inserts some code that computes the norm of the gradients with the respect to the three sets of variables. If the training converges, these norms should $\to 0$

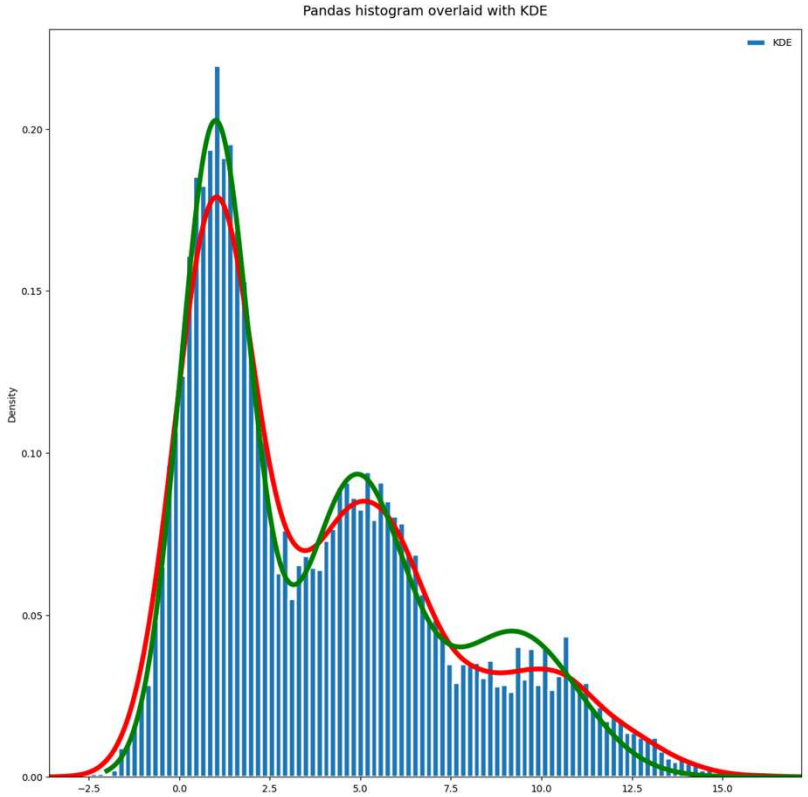
```
gradient_norms = defaultdict(list)
for name,value in pyro.get_param_store().named_parameters():
    value.register_hook(lambda g,name=name: gradient_norms[name].append(g.norm().item()))

losses = []
for i in range(700):
    loss = svi.step(data)
    losses.append(loss)
    print('.' if i % 100 else '\n', end='')
```

The MAP estimates are

```
weights =[0.49195504 0.1915361  0.3165089 ]
locs=[0.9845338 9.28559   4.87013  ]
scales = [0.9769526 1.720114  1.3770542]
```

Plotting the estimated density function

```
def estimated_density(x):
    return np.sum([scipy.stats.norm(loc=mu,scale=sigma2).pdf(x)*pi\
                   for  mu,sigma2,pi in zip(locs.detach().numpy(),\
                                            scales.detach().numpy(),
                                             weights.detach().numpy())])
```



Using sklearn

sklearn has a class **mixture.GaussianMixture** that will find the aprroximate parameters from the data

```
from sklearn.mixture import GaussianMixture
```

```
GMM = GaussianMixture(n_components=3,
                      covariance_type='full',
                      tol=0.001,
                      reg_covar=1e-06,
                      max_iter=700,
                      n_init=1,
                      init_params='kmeans')
```

```
data_np = data.numpy().reshape(-1,1)
```

```
1 GMM.fit(data_np)
```

```
GaussianMixture(max_iter=700, n_components=3)
```

```
1 GMM.means_
```

```
array([[ 1.06627283],
       [ 5.23887506],
       [10.18118554]])
```

```
1 GMM.weights_
```

```
array([0.52372087, 0.32188703, 0.1543921 ])
```

```
1 GMM.covariances_
```

```
array([[[1.13644822]],

       [[2.02028996]],

       [[3.49414859]]])
```

This variance is way off

The results are pretty much the same as what we get with **pyro**, except the last variance which should be 1.5

The EM-algorithm

`sklearn.mixture.GaussianMixture` uses the *EM Algoritm* (Expectation-Maximization)

Let θ denote the parameters $\{\mu_i, \sigma_i^2\}_{i=0,1,2}$ and π the weight distribution. Then we can write

$$\log p(\mathcal{D}) = \log \int p(\mathcal{D}, \theta, \pi) d\theta d\pi$$

Let $q(\theta, \pi)$ be any distribution over θ and π and write

$$\log p(\mathcal{D}) = \log \int q(\theta, \pi) \frac{p(\mathcal{D}, \theta, \pi)}{q(\theta, \pi)} d\theta d\pi = \log \mathbb{E}_{q(\theta,\pi)} \left(\frac{p(\mathcal{D}, \theta, \pi)}{q(\theta, \pi)} \right)$$

Using Jensen's inequality

$$\log \mathbb{E}_{q(\theta,\pi)} \left(\frac{p(\mathcal{D}, \theta, \pi)}{q(\theta, \pi)} \right) \geq \mathbb{E}_{q(\theta,\pi)} \left(\log \frac{p(\mathcal{D}, \theta, \pi)}{q(\theta, \pi)} \right)$$

We want to find a distribution $q(\theta, \pi)$ that maximizes the expectation

The expectation is in fact precisely the ELBO, since

$$\begin{aligned}
\mathbb{E}_{q(\theta,\pi)}\left(\log\frac{p(\mathcal{D},\theta,\pi)}{q(\theta,\pi)}\right) &= \mathbb{E}_{q(\theta,\pi)}\left(\log\frac{p(\mathcal{D}|\theta,\pi)p_{prior}(\theta,\pi)}{q(\theta,\pi))}\right) \\
&= \mathbb{E}_{q(\theta,\pi)}\left(\log\frac{p(\mathcal{D}|\theta,\pi)}{q(\theta,\pi)}\right) + \mathbb{E}_{q(\theta,\pi)}\left(\log p_{prior}(\theta,\pi)\right) \\
&= \mathbb{E}_{q(\theta,\pi)}\left(\log p(\mathcal{D}|\theta,\pi)\right) - D_{KL}(q(\theta,\pi)||p_{prior}(\theta,\pi))
\end{aligned}$$

Now assume $q(\theta, \pi) = q(\theta)q(\pi)$

Then we can write the integral

$$\int q(\theta, \pi) \log \frac{p(\mathcal{D}, \theta|\pi)p_{prior}(\pi)}{q(\theta, \pi))} d\theta d\pi = \int q(\theta) \left(\int q(\pi) \log \frac{p(\mathcal{D}, \theta|\pi)p_{prior}(\pi)}{q(\theta)q(\pi)} d\pi \right) d\theta$$

We are using an iterative proces to find $q(\theta)$ and $q(\pi)$

Assume $q^{(t)}(\theta)$ and $q^{(t)}(\pi)$ have been computed

we then want to find $q^{(t+1)}(\theta)$ which maximizes

$$\int q^{(t+1)}(\theta) \left(\int q^{(t)}(\pi) \log \frac{p(\mathcal{D}, \theta|\pi) p_{prior}(\pi)}{q^{(t+1)}(\theta) q^{(t)}(\pi)} d\pi \right) d\theta$$

$$= \mathbb{E}_{q^{(t+1)}(\theta)} \left(\mathbb{E}_{q^{(t)}(\pi)} \left(\log \frac{p(\mathcal{D}, \theta|\pi) p_{prior}(\pi)}{q^{(t+1)}(\theta) q^{(t)}(\pi)} \right) \right)$$

Rewrite the inner expectation

$$\begin{aligned}
&\mathbb{E}_{q^{(t)}(\pi)}\left(\log \frac{p(\mathcal{D},\theta|\pi)p_{prior}(\pi)}{q^{(t+1)}(\theta)q^{(t)}(\pi)}\right) \\
&= \mathbb{E}_{q^{(t)}(\pi)}\left(\log p(\mathcal{D},\theta|\pi)\right) + \mathbb{E}_{q^{(t)}(\pi)}(\log p_{prior}(\pi)) \\
&- \mathbb{E}_{q^{(t)}(\pi)}\left(\log q^{(t+1)}(\theta)\right) - \mathbb{E}_{q^{(t)}(\pi)}\left(\log q^{(t)}(\pi)\right)
\end{aligned}$$

Now $\log q(\theta)$ does not depend on π so

$$\mathbb{E}_{q^{(t)}(\pi)}\left(\log q(\theta)\right) = \log q(\theta)$$

The term,

$$\mathbb{E}_{q^{(t)}(\pi)}(\log p_{prior}(\pi)) - \mathbb{E}_{q^{(t)}(\pi)}\left(\log q^{(t)}(\pi)\right)$$

does not depend on θ and so for the maximization can be viewed as a constant.

Thus we can write

$$\begin{aligned}
&\mathbb{E}_{q^{(t+1)}(\theta)}\left(\mathbb{E}_{q^{(t)}(\pi)}\left(\log\frac{p(\mathcal{D},z|\pi)p(\theta)}{q^{(t+1)}(\theta)q^{(t)}(\pi)}\right)\right)\\
&=\mathbb{E}_{q^{(t+1)}(\theta)}\left(\log\exp\left(\mathbb{E}_{q^{(t)}(\pi)}\left(\log p(\mathcal{D},\theta|\pi)\right)\right)\right)-\mathbb{E}_{q^{(t+1)}(\theta)}\left(\log q_z(z)\right)\\
&+const\\
&=\mathbb{E}_{q^{(t+1)}(\theta)}\left(\log\frac{\exp\left(\mathbb{E}_{q^{(t)}(\pi)}\left(\log p(\mathcal{D},\theta|\pi)\right)\right)}{q^{(t+1)}(\theta)}\right)+const
\end{aligned}$$

We recognize the expectation as the negative KL-divergence

$$-D_{KL}(q^{(t+1)}(\theta)||\exp(\mathbb{E}_{q^{(t)}(\pi)}(p(\mathcal{D},\theta|\pi))$$

The maximum value is 0 and so we get the maximum is when

$$q^{(t+1)}(\theta) = \exp(\mathbb{E}_{q^{(t)}(\pi)}(p(\mathcal{D},\theta|\pi))$$

A similar computation then gives us the maximal distribution

$$q^{(t+1)}(\pi) = p_{prior}(\pi) \exp\left(\mathbb{E}_{q^{(t+1)}(\theta)}(p(\mathcal{D}, \theta|\pi))\right)$$

One can show that this process converges and the limit distribution is the optimal variational distribution from the family of distributions where θ and π are independent