Variational Inference

In the general Bayesian situation we have a random variable $\mathbf{X}$, depending on a *latent variable* (meaning hidden) $\mathbf{Z}$ with density $p_{\mathbf{Z}}(\mathbf{z})$ where we know $p_{\mathbf{X}|\mathbf{Z}}$ and $p_{\mathbf{Z}}$. The joint density $p_{\mathbf{X},\mathbf{Z})}(\mathbf{x},\mathbf{z}) = p_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{z})p_{\mathbf{Z}}(\mathbf{z})$. The posterior distribution

$$p_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x}) = \frac{p_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{z})}{p_{\mathbf{X}}(\mathbf{x})} p_{\mathbf{Z}}(\mathbf{z})$$

is intractable because of the intractibility of

$$p_{\mathbf{X}}(\mathbf{x}) = \int p_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{z})p_{\mathbf{Z}}(\mathbf{z})d\mathbf{z}$$

The idea of Variational Inference is to approximate the posterior distribution with a parametrized distribution $q_\phi(\mathbf{z})$ from a family of distributions e.g. Gaussians, which are parametrized by a parameter vector ϕ (q_ϕ can also be conditional on $\mathbf{X}$ so instead of $q_\phi(\mathbf{z})$ we are looking for a conditional density $q_\phi(\mathbf{z}|\mathbf{x})$ We want q_ϕ to be as close to $p(\mathbf{z}|\mathbf{x})$ as possible in the sense that $D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z}||\mathbf{x}))$ (or $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$) is minimized.

The problem of finding q now becomes a problem of finding ϕ_0

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

Of course this seems to be equally hard because to compute the KL-divergence it seems that we need to know the distribution $p(\mathbf{z}|\mathbf{x})$ which we are trying to approximate.

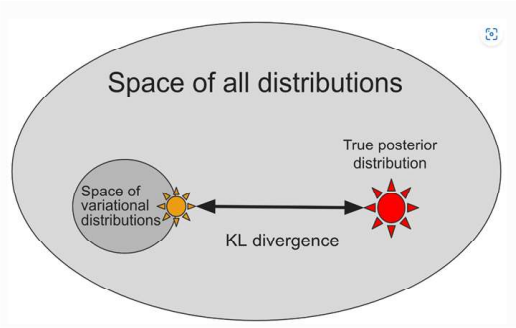

Figure 1:

We can rewrite the KL-divergence

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

Now

$$\mathbb{E}_{q_\phi}(\log p(\mathbf{x})) = \int q_\phi(\mathbf{z}) \log p(\mathbf{x}) d\mathbf{z} = \log p(\mathbf{x}) \int q_\phi(\mathbf{z}) d\mathbf{z} = \log p(\mathbf{x})$$

since $\log p(\mathbf{x})$ does not depend on $\mathbf{z}$ and since $\int q_\phi(\mathbf{z})d\mathbf{z} = 1$ (q_ϕ is a density)

The term $\mathbb{E}_{q_\phi}(\log q_\phi) - \mathbb{E}_{q_\phi}(\log p(\mathbf{z})) = D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z}))$ so we get

$$D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z}|\mathbf{x}) = D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z})) - \mathbb{E}_{q_\phi}(\log p(\mathbf{x}|\mathbf{z})) + \log p(\mathbf{x})$$

Since $\log p(\mathbf{x})$ is constant i.e. does not depend on q_ϕ we are reduced to minimizing the expression

$$D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z})) - \mathbb{E}_{q_\phi}(\log p(\mathbf{x}|\mathbf{z}))$$

Rearranging the terms we get

$$\log p(\mathbf{x}) = \mathbb{E}_{q_\phi}(\log p(\mathbf{x}|\mathbf{z})) - D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z})) + D_{KL}(q_\phi(\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$$

Since KL-divergence is always ≥ 0 we get

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi}(\log p(\mathbf{x}|\mathbf{z})) - D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z}))$$

The log probability, $\log p(\mathbf{x})$, is called the *evidence* and so

$$\mathbb{E}_{q_\phi}(\log p(\mathbf{x}|\mathbf{z})) - D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z}))$$

is a lower bound for the evidence. This expression is called the *ELBO = Evidence Lower BOund* and our problem is then reduced to *maximizing* the ELBO.

In order to solve this maximization i.e. to find

$$\underset{\phi}{\text{argmax}}\ \mathbb{E}_{q_\phi}(\log p(\mathbf{x}|\mathbf{z})) - D_{KL}(q_\phi(\mathbf{z})||p(\mathbf{z}))$$

we can use gradient ascent so we need to compute

$$\nabla_\phi ELBO(\phi) = \nabla_\phi \mathbb{E}_{q_\phi}\left(\log p(\mathbf{x}|\mathbf{z}) + \log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}))}\right)$$

In the real world we have observations given as a dataset $\mathcal{D}$, the latent variable is of the form $\mathbf{z}, \theta$ where θ are parameters for the distribution $p = p_\theta$. We could also write this as

$$p_\theta(\mathbf{x}|\mathbf{z}) = p(\mathbf{x}|\mathbf{z}, \theta)$$

i.e. the parameters are also latent variables.

The Maximum Likelihood principle would be to compute the argmax of the log-likelihood

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

Again this is in most cases intractable, so we instead maximize the $ELBO$.

The $ELBO$ now depends on two sets of parameters so we need gradients with respect to both

$$\nabla_{\theta,\phi} \mathbb{E}_{q_\phi} \left(\log p_\theta(\mathcal{D}|\mathbf{z}) + \log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z})} \right)$$

Example Assume we have a *Gaussian Mixture Model (GMM)* with three components with unknown parameters consisting of means μ_1, μ_2, μ_3, standard deviations $\sigma_1, \sigma_2, \sigma_3$ and weights $\pi(1), \pi(2), \pi(3)$. We let θ denote the set of parameters $\theta = (\mu_1, \mu_2, \mu_3, \sigma_1, \sigma_2, \sigma_3, \pi)$

Let $p(\mathbf{x})$ be the density function and let $p(\mathbf{z})$ be the probability mass function of π (since the distribution is discrete, there is no density function)

$p(\mathbf{x})$ can be quite complicated but the conditional density

$$p(\mathbf{x}|z = i) = \mathcal{N}(\mathbf{x}|\mu_i, sigms_i^2)$$

We are given a set of samples $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ from the GMM. The density function is of the form

$$p(\mathcal{D}|\theta) = \prod_{\mathbf{x} \in \mathcal{D}} \pi(1)\mathcal{N}(\mathbf{x}; \mu_1, \sigma_1^2) + \pi(2)\mathcal{N}(\mathbf{x}; \mu_1, \sigma_2^2) + \pi(3)\mathcal{N}(\mathbf{x}; \mu_3, \sigma_3^2)$$

We sample from this distribution by first sample an $i = 1, 2, 3$ from the distribution π and then sampling from the Gaussian $\mathcal{N}(\mu_i, \sigma_i^2)$

We want to construct samples from the posterior distribution

$$p_{post}(\theta|\mathcal{D})$$

Choosing a prior $p_{prior}(\theta)$ the posterior is

$$p_{post}(\theta|\mathcal{D}) = \frac{\prod_{\mathbf{x}\in\mathcal{D}} p(\mathbf{x}|\theta)}{p(\mathcal{D})} p_{prior}(\theta)$$

As usual the denominator

$$p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)d\theta$$

is intractable so we cannot compute the posterior analytically but we can use HMC to generate samples from $p_{post}(\theta|\mathcal{D})$

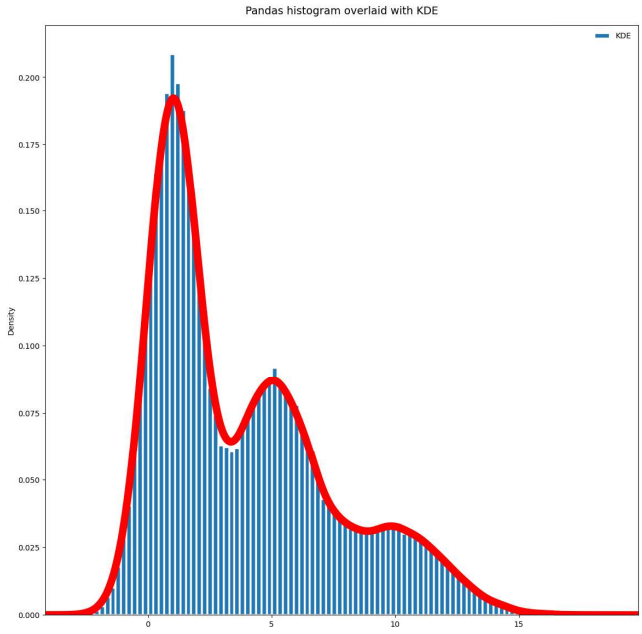

Figure 2:

0.1 Pyro

To do the *ELBO* computations and the optimization we shall use the Python package *pyro*. pyro is a Probabilistic Programming Language and is an extension of *Pytorch* to enable computations with random variables.

pyro is installed with the command `pip install pyro-ppl`

In pyro a random variable is a function whose values are random i.e. not deterministic but rather the values of a random variable are samples from a distribution. This is implemented by the `pyro.sample` command.

A probabilistic model is basically a parametrized joint distribution

$$p_\theta(\mathcal{D}, \mathbf{z}) = p(\mathcal{D}, \mathbf{z}|\theta) = p_\theta(\mathcal{D}|\mathbf{z})p_\theta(\mathbf{z})$$

where $\mathcal{D}$ are observations, $\mathbf{z}$ is a vector of latent variables and θ is a vector of parameters for the distribution p_θ. In many cases θ will be parameters of a Neural Net.

There can many relations among the variables. A common situation is where

$$\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$$

and the $\mathbf{x}_i$'s are not independent, but are independent conditional on $\mathbf{z}$ so

$$p_\theta(\mathcal{D}) \neq \prod_{\mathbf{x}\in\mathcal{D}} p_\theta(\mathbf{x})$$

but

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

The independence/dependence relations can be illustrated graphically. In Figure 1 the rectangle is called a *plate* and we can think of a stack of the $\mathbf{x}_i$'s as a stack of variables depending on $\mathbf{z}$ and θ and being independent conditional on $\mathbf{z}$ and θ

In pyro a stack of plates is indicated by the context `with pyro.plate(...):`

A model in pyro is specified by defining latent variables (including the parameters) using `z = pyro.sample('z',...)` commands, and the observations independent conditional on `z`

```
with pyro.plate('data',N):
pyro.sample("obs", ..some distribution dependent on z.., obs=data)
```

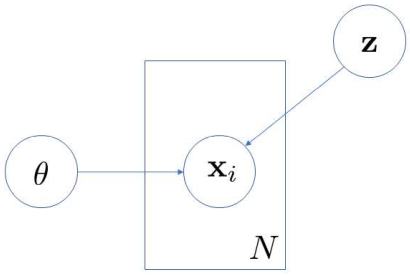

Figure 3:

```
def model(data):
    # specifying the latent variable z = (mu,sigma)
    # the distribution p(z) = p(mu,sigma) = p(sigma|mu)p(mu)
    # = LogNormal(sigma; mu,1)Normal(mu;0,1)
    #remark that all definitions of random variables are of the form
    # name = pyro.sample('name',..distribution..)
    mu = pyro.sample('mu',dist.Normal(0,1))
    sigma = pyro.sample('sigma',dist.LogNormal(mu,1))

    #specifying the observations
    with pyro.plate('N', len(observations)):
        pyro.sample('obs',dist.Normal(mu,sigma),obs=observations)
```

```
pyro.render_model(model,model_args=(observations,))
```

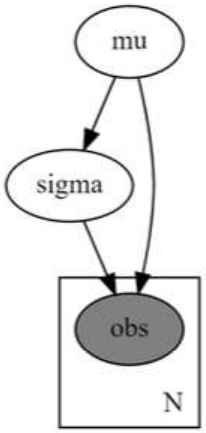

Figure 4:

Figure 3 is a very simple example. We can see that the latent variables are the parameters μ and σ. The prior distribution is

$$p_{prior}(\mu, \sigma) = LogNormal(\sigma|\mu, 1)\mathcal{N}(\mu|0, 1)$$

$$\begin{aligned} \sigma|\mu &\sim LogNormal(\mu, 1) \\ \mu &\sim \mathcal{N}(0, 1) \end{aligned}$$

We have specified that the likelihood

$$p(\mathcal{D}|\mu, \sigma) = \prod_{\mathbf{x}]in\mathcal{D}} \mathcal{N}(\mathbf{x}|\mu, \sigma)$$

In this example the task would be to find an approximation to the posterior

$$p_{post}(\mu, \sigma|\mathcal{D}) = \frac{p(\mathcal{D}|\mu, \sigma)p_{prior}(\mu, \sigma)}{p(\mathcal{D})}$$

where $\mathcal{D}$ is a data set.

In pyro the *Variational Distribution* q_ϕ is called the *guide* and we have a lot of freedom to choose from which family of distributions we choose q_ϕ.

Assume the parameter space is finite so integrals become summations. If we were only interested in getting point estimates of the parameters we could let q_ϕ be a Dirac distribution δ_ϕ which is the distribution $\delta_\phi(\phi)$ where δ is the Dirac delta function

$$\delta_\phi(\theta) = 0 \text{ for } \theta \neq \phi \text{ and } \delta_\phi(\phi) = 1$$

The ϕ that maximizes the $ELBO$ would be the *Maximum A posterior Probability* (MAP).

Indeed

$$\mathbb{E}_{\delta_\phi}(f(\theta)) = \sum \delta_\phi(\theta) f(\phi) = f(\phi)$$

So

$$ELBO(\phi) = \mathbb{E}_{\delta_\phi}(\log p(\mathcal{D}|\theta)) + \log p(\theta) - \log \delta_\phi)$$

but

$$E_{\delta_\phi}(\log \delta_\phi) = \log 1 = 0$$

so

$$ELBO(\phi) = \log p(\mathcal{D}|\phi)p(\phi)$$

It follows that

$$\underset{\phi}{\text{argmax}}\ ELBO(\phi) = \underset{\phi}{\text{argmax}}\ \log p(\mathcal{D}|\phi)p(\phi)$$

and since $p(\mathcal{D})$ does not depend on ϕ

$$\underset{\phi}{\text{argmax}}\ \log p(\mathcal{D}|\phi)p(\phi) = \underset{\phi}{\text{argmax}}\ \log \frac{p(\mathcal{D}|\phi)p(\phi)}{p(\mathcal{D})} = \underset{\phi}{\text{argmax}}\ p_{post}(\phi|\mathcal{D})$$

A very common family of distributions, $\mathcal{Q}$, is the *mean field variational family.* A general member of this family is of the form

$$q_\phi(\mathbf{z}) = \prod q_{\phi_i}(z_i)$$

where the product is over the latent variables $\mathbf{z} = \{z_i\}_i$.

In particular the latent variables are independent under q_ϕ

In pyro guides (= Variational Distributions) are very much like models. They have to have the same *signature* as the model so if the model is of the form

```
def model(data)
```

then the *guide* has to have the same form

```
def guide(data)
```

Actually pyro makes it very easy to make guides using the `pyro.infer.AutoGuide`
The `AutoGuide` comes in several variants depending on the family of variational distributions we consider

```
AutoDelta, AutoNormal, AutoMultivariableNormal, AutoGaussian,...
```

The `AutoGuide` is simply called by

```
guide = AutoGuide(model)
```

where `AutoGuide` can be any of the variants depending on which family is appropriate

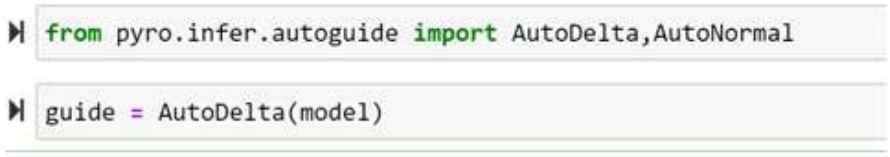

Figure 5:

Once we have the model and the guide we have the ingredients for the $ELBO$. In pyro the $ELBO$ is called the `Trace_ELBO`. The `Trace` is because it recalculates the $ELBO$ for every iteration of the gradient ascent

```
elbo = Trace_ELBO()
```

Computing the gradient

$$\nabla_{\phi,\theta} ELBO(\phi,\theta) = \nabla_{\phi,\theta} \mathbb{E}_{q_\phi}(\log p(\mathcal{D}|\theta)) + \log p(\theta) - \log q_\phi(\theta))$$

can a little complicated.

The gradient with respect to θ can be moved in under the expectation

$$\nabla_\theta ELBO(\phi, \theta) = \mathbb{E}_{q_\phi}(\nabla_\theta \log p(\mathcal{D}|\theta) + \nabla_\theta \log p(\theta) - \nabla_\theta \log q_\phi(\theta))$$

But the gradient with respect to ϕ is more complicated because we are taking the expectation with respect to a distribution q_ϕ so we can't just move the gradient under the expectation.

We shall consider the case where $\mathbb{E}_{q_\phi}(f_\phi(z))$ can be reparametrized in the sense that we can find a change of variables $z = g_\phi(x)$ such that

$$\mathbb{E}_{q_\phi}(f_\phi(z)) = \int q_\phi(z) f_\phi(z) dz = \int q_\phi(g_\phi(x)) f_\phi(g_\phi(x)) g'_\phi(x) dx$$

and such that

$$q_\phi(g_\phi(x)) g'_\phi(x) = q(x)$$

is a distribution independent of ϕ

For instance if

$$q_\phi(z) = \mathcal{N}(z|\mu(\phi), \sigma^2(\phi)) = \frac{1}{\sqrt{2\pi}\sigma(\phi)} \exp\left(-\frac{(z-\mu(\phi))^2}{2\sigma(\phi)^2}\right)$$

Put

$$z = \mu(\phi) + \sigma(\phi)x = g_\phi(x)$$

then

$$\sigma(\phi) q_\phi(z) = \sigma(\phi) q_\phi(g_\phi(x)) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) = \mathcal{N}(x|0,1) = q(x)$$

$$\begin{aligned}
&\int q_\phi(z) f_\phi(z) dz = \int q_\phi(g_\phi(x)) f_\phi(g_\phi(x)) g'_\phi(x) dx \\
&= \int q_\phi(g_\phi(x)) f_\phi(g_\phi(x)) \sigma(\phi) dx = \int q(x) f_\phi(g_\phi(x)) dx \\
&= \mathbb{E}_q(f_\phi(g_\phi(x)))
\end{aligned}$$

Now the distribution we are taking expectation over no longer depends on ϕ so

$$\nabla_\phi \mathbb{E}_{q_\phi}(f_\phi(z)) = \mathbb{E}_q(\nabla_\phi f_\phi(g_\phi(x)))$$

This *reparametrization trick* is used in a number of generative models as we will see later.

In general to use this trick we need a function g_ϕ such that $q_\phi(g_\phi(x)) \dfrac{\partial}{\partial x} g_\phi(x)$ does not depend on ϕ

This means that g_ϕ must satisfy the partial differential equation

$$0 = \frac{\partial}{\partial \phi}\left(q_\phi(g_\phi(x))\,\frac{\partial}{\partial x} g_\phi(x)\right) = \frac{\partial}{\partial \phi} q_\phi(g_\phi(x)) \frac{\partial}{\partial \phi} g_\phi(x) \frac{\partial}{\partial x} g_\phi(x) + q_\phi(g_\phi(x)) \frac{\partial^2}{\partial \phi \partial x} g_\phi(x)$$

If the reparametrization is not available the situation is more problematic.

Write

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{q_\phi}(f_\phi) &= \nabla_\phi \int q_\phi(z) f_\phi(z) dz \\
&= \int \nabla_\phi \left(q_\phi(z) f_\phi(z)\right) dz = \int [(\nabla_\phi q_\phi(z)) f_\phi(z) + q_\phi(z) \nabla_\phi f_\phi(z)] dz \\
&= \int q_\phi(z) \left(\frac{\nabla_\phi q_\phi(z)}{q_\phi(z)} f_\phi(z) + \nabla_\phi f_\phi(z)\right) dz \\
&= \int q_\phi(z) \left(\nabla_\phi \log q_\phi(z) f_\phi(z) + \nabla_\phi f_\phi(z)\right) dz \\
&= \mathbb{E}_{q_\phi}(\nabla_\phi \log q_\phi(z) f_\phi(z) + \nabla_\phi f_\phi(z))
\end{aligned}$$

In both cases we can express

$$\nabla_\phi \mathbb{E}_{\phi_\phi}(f_\phi(z))$$

as an expectation.

An expectation can be approximated by averaging over samples from the underlying distribution (Monte Carlo Estimation). Once we can estimate gradients we can perform (approximate) gradient ascent to maximize the $ELBO$.

This process, approximate gradients and maximize $ELBO$ is known as *Stochastic Variational Inference (SVI)*. In pyro we can automate this process using the class `SVI`

```
from pyro.infer import SVI, Trace_ELBO
from pyro.optim import Adam

optimizer = Adam({'lr':0.01})

svi = SVI(model,guide,optimizer,loss=Trace_ELBO)
```

We need to instantiate an *optimizer*, here we use the `Adam` optimizer which is usually a safe choice. The parameters for the optimizer are passed as a `dict`, this is different from the optimizer class in Pytorch.

Here we set the learning rate to 0.01 and leave the other parameters at their default values.

The `svi` object contains two methods: `step()` and `evaluate_loss()`

The `step()` takes in the training data and computes the gradients, takes a gradient step i.e.updates the parameters and computes the new $ELBO$

$$\phi_{new}, \theta_{new} = \phi_{old}, \theta_{old} + \alpha \nabla_{\phi,\theta} ELBO(\phi, \theta)$$

The updated parameters are passed to the `model` and the `guide`

It is now very simple to train the model,

```
for epoch in range(1000):
    svi.step(observations)
```

This will run through 1000 iterations of the gradient ascent process (actually gradient descent because the process is minimizing $-ELBO$)

The trained parameters are stored in the `pyro.param_store`

```
pyro.get_param_store.keys() = dict_keys(['AutoDelta.mu','AutoDelta.sigma'])
pyro.param('AutoDelta.mu').item()
    3.976011276245117
pyro.param('AutoDelta.sigma').item()
    1.3952281475067139
```

The actual values are 4.0 and 1.4

If we use another guide, `AutoNormal` we get the variational distribution of the parameters μ and σ. The guide will treat the parameters as being independent and the variational approximation to $p_{post}(\mu, \sigma|\mathcal{D})$ is a product of two Gaussians

$$p_{var}(\mu, \sigma|\mathcal{D}) = \mathcal{N}(\mu; locs.\mu, scales.\mu)\mathcal{N}(\sigma; locs.\sigma, scales.\sigma)$$

The model computes the parameters for the two Gaussians
locs.mu 3.5832772254943848
scales.mu 0.09438330680131912
locs.sigma 0.4783938229084015
scales.sigma 0.06678292900323868