A primer on variational inference

Applibugs seminar

Philippe Veber

December 12th, 2025

Laboratoire de Biométrie et Biologie Évolutive

Problem statement

Given a model

$$Z \sim \mathcal{D}_Z$$
$$X \sim \mathcal{D}_X(Z)$$

s.t. densities $p(x \mid z)$ and $p(z)$ can be computed up to a constant factor

For an observation x of X,

derive an efficient sampler for the posterior distribution $Z \mid X = x$

Variational inference in a nutshell

Let $\mathcal{Q}$ be a set of distributions

$$\text{Find } q \in \mathcal{Q} \text{ closest to } p(z \mid X = x)$$

- $\mathcal{Q}$ is called the **variational family**
- $\mathcal{Q}$ should contain distributions "close enough" to the target
- the key idea here is to cast Bayesian inference as an **optimization** problem

Plan

1. how to measure distance to target distribution?
2. how to minimize said distance?
3. how do we choose the variational family?

How to measure distance to the target posterior distribution?

Intuition

Let p and q be distributions on $\mathcal{X}$

Intuition

Let $x \in \mathcal{X}$, discrepancy between p and q at x is best measured by the ratio

$$\frac{p(x)}{q(x)}$$

because densities are multiplicative quantities

Intuition

To obtain a global measure, we consider the geometric average (multiplicative quantities) or equivalently the arithmetic average of the log

$$\int \log \frac{p(x)}{q(x)} \, \mathrm{d}\mu(x)$$

Under which measure though, p or q?

Kullback-Leibler divergence

$$\begin{aligned}\mathrm{KL}(p \parallel q) &= \int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x \\ &= \mathbb{E}_{x \sim p}\left[\log \frac{p(x)}{q(x)}\right]\end{aligned}$$

Kullback-Leibler divergence

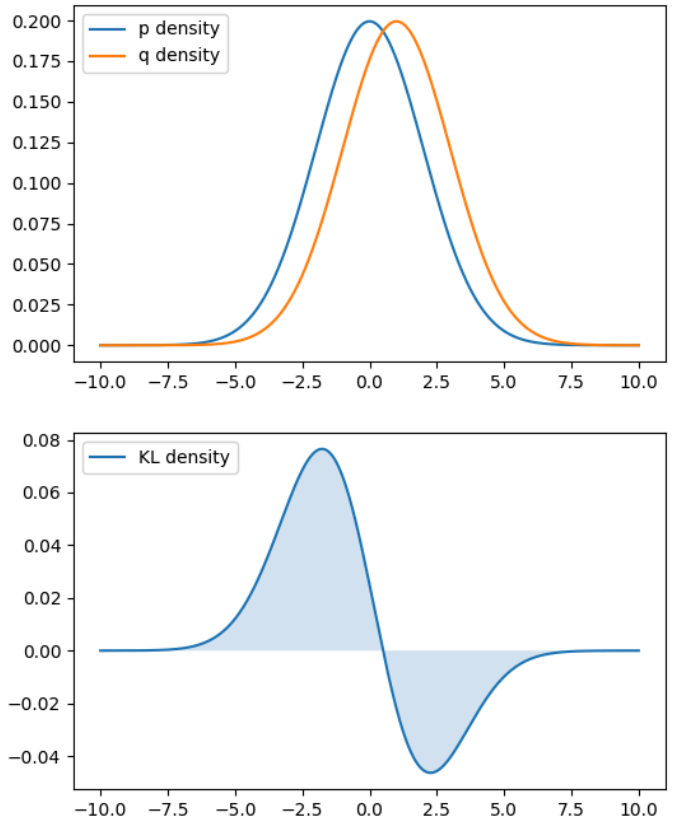
$$\begin{aligned}\mathrm{KL}(p \parallel q) &= \int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x \\ &= \mathbb{E}_{x \sim p}\left[\log \frac{p(x)}{q(x)}\right]\end{aligned}$$

- not symmetric, hence not a distance
- $\mathrm{KL}(p \parallel q) = 0 \Rightarrow p = q$
- $\mathrm{KL}(p \parallel q) \geq 0$

How can it always be positive?

Because q sums to 1:

- if q is high when p is low (negative contribution)
- then it has to be low when p is high (positive contribution)
- but positive contributions have more weight!



$\mathbf{KL}(p \parallel q)$ or $\mathbf{KL}(q \parallel p)$?

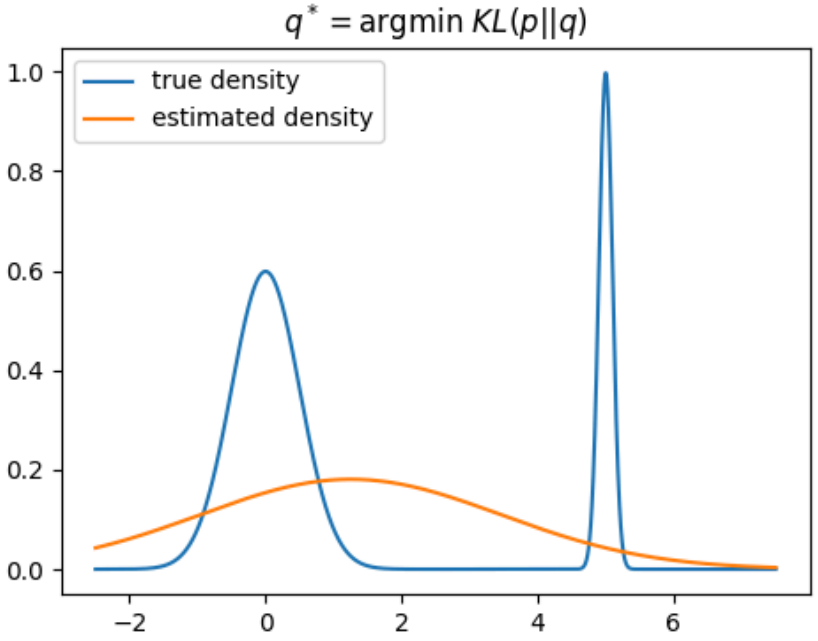
Say p is the "true" (target) distribution while q is the approximating distribution

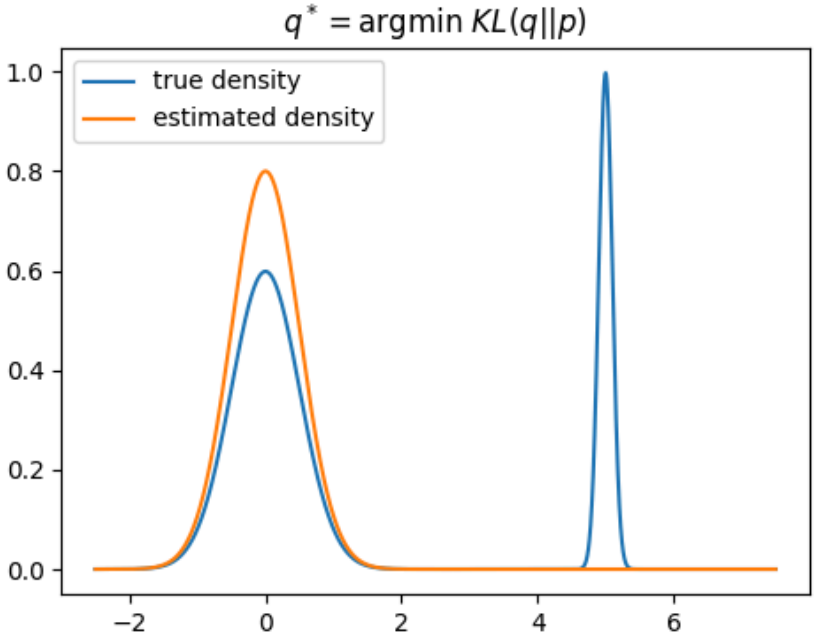
Let's choose $p = \frac{2}{3}\mathcal{N}\left(0, \frac{1}{4}\right) + \frac{1}{3}\mathcal{N}\left(\frac{1}{2}, \frac{1}{100}\right)$ and

$$Q = \left\{ \mathcal{N}(\mu, \sigma) \mid \mu \in \mathbb{R}, \sigma \in \mathbb{R}_*^+ \right\}$$

Note that $p \notin Q$!

$\mathrm{KL}(p \parallel q)$ or $\mathrm{KL}(q \parallel p)$?





Two different behaviours: **mode covering** vs **mode seeking**

How to minimize KL divergence to the target posterior distribution?

Inference as an optimization/approximation problem

Reminder, given:

$$X \sim \mathcal{D}_X(Z)$$
$$Z \sim \mathcal{D}_Z$$

and

$$Q = \left\{ q_\phi \mid \phi \in \mathbb{R}^n \right\}$$

we look for $q(z) \in Q$ closest to $p(z \mid X = x)$

Inference as an optimization/approximation problem

Strategy:

- need to choose between $\mathrm{KL}(p \parallel q)$ and $\mathrm{KL}(q \parallel p)$
- we don't have a sample of $p(z \mid X = x)$ (that's what we're after!)
- so let's go for

$$q^* = \operatorname*{argmin}_{\phi} \mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big)$$

Mathematical analysis

$$\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big)$$

$$= \mathbb{E}_{z \sim q_\phi}\left[\log q_\phi(z) - \log p(z \mid X = x)\right]$$

Mathematical analysis

$$\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big)$$

$$= \mathbb{E}_{z\sim q_\phi}\left[\log q_\phi(z) - \log p(z \mid X = x)\right]$$

$$= \mathbb{E}_{z\sim q_\phi}\left[\log q_\phi(z) - (\log p(X = x \mid z) + \log p(z) - \log p(X = x))\right]$$

Mathematical analysis

$$\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big)$$

$$= \mathbb{E}_{z\sim q_\phi}\left[\log q_\phi(z) - (\log p(X = x \mid z) + \log p(z) - \log p(X = x))\right]$$

$$= \mathbb{E}_{z\sim q_\phi}[-\log p(X = x \mid z)] + \mathbb{E}_{z\sim q_\phi}\left[\log q_\phi(z) - \log p(z)\right] + \log p(X = x)$$

Mathematical analysis

$$\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big)$$

$$= \mathbb{E}_{z\sim q_\phi}[-\log p(X = x \mid z)] + \mathbb{E}_{z\sim q_\phi}\big[\log q_\phi(z) - \log p(z)\big] + \log p(X = x)$$

$$= \mathbb{E}_{z\sim q_\phi}[-\log p(X = x \mid z)] + \mathrm{KL}\big(q_\phi(z) \parallel \log p(z)\big) + \log p(X = x)$$

Mathematical analysis

$$\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big)$$

$$= \mathbb{E}_{z \sim q_\phi}[-\log p(X = x \mid z)] + \mathrm{KL}\big(q_\phi(z) \parallel \log p(z)\big) + \log p(X = x)$$

$$= -\ \mathrm{ELBO}(\phi) + \log p(X = x)$$

Evidence Lower BOund

$$\mathrm{ELBO}(\phi) = \underbrace{\mathbb{E}_{z \sim q_\phi}[\log p(X = x \mid z)]}_{\text{fit to the data}} - \underbrace{\mathrm{KL}\big(q_\phi(z) \parallel p(z)\big)}_{\text{distance to prior}}$$

Evidence Lower BOund

$$\mathrm{ELBO}(\phi) = \underbrace{\mathbb{E}_{z\sim q_\phi}[\log p(X = x \mid z)]}_{\text{fit to the data}} - \underbrace{\mathrm{KL}\big(q_\phi(z) \parallel p(z)\big)}_{\text{distance to prior}}$$

Since $\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big) = -\,\mathrm{ELBO}(\phi) + \log p(X = x) \geq 0$

maximizing the ELBO is equivalent to minimizing $\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big)$

Evidence Lower BOund

$$\mathrm{ELBO}(\phi) = \underbrace{\mathbb{E}_{z\sim q_\phi}[\log p(X = x \mid z)]}_{\text{fit to the data}} - \underbrace{\mathrm{KL}\big(q_\phi(z) \parallel p(z)\big)}_{\text{distance to prior}}$$

Since $\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big) = -\,\mathrm{ELBO}(\phi) + \log p(X = x) \geq 0$

maximizing the ELBO is equivalent to minimizing $\mathrm{KL}\big(q_\phi(z) \parallel p(z \mid X = x)\big)$

also we have

$$\log p(X = x) \geq \mathrm{ELBO}(\phi)$$

Hence the ELBO can be used as an (under)-approximation to the marginal likelihood.

Stochastic gradient

ELBO maximization typically achieved through gradient ascent

but computing the gradient of an expectation can be tricky

Stochastic gradient

ELBO maximization typically achieved through gradient ascent

but computing the gradient of an expectation can be tricky

$$\nabla_\phi \left[\mathbb{E}_{z \sim q_\phi} \left[\log p(X = x \mid z) + \log p(z) - \log q_\phi(z) \right] \right]$$

Stochastic gradient

ELBO maximization typically achieved through gradient ascent

but computing the gradient of an expectation can be tricky

$$\nabla_\phi \left[\mathbb{E}_{z \sim q_\phi} \left[\log p(X = x \mid z) + \log p(z) - \log q_\phi(z) \right] \right]$$

- the naive Monte Carlo estimate has too high of a variance
- several alternatives have been proposed (see review [1])

Reparameterization trick [2]

Assume the variational density q_ϕ can be obtained through a model of the form

$$\varepsilon \sim p(\varepsilon)$$
$$\tilde{z} = g_\phi(\varepsilon)$$

with density q_ϕ then for any function f of z

$$\int q_\phi(z) f(z)\,\mathrm{d}z = \int f\big(g_\phi(\varepsilon)\big) p(\varepsilon)\,\mathrm{d}\varepsilon$$

Reparameterization trick [2]

Assume the variational density q_ϕ can be obtained through a model of the form

$$\varepsilon \sim p(\varepsilon)$$
$$\tilde{z} = g_\phi(\varepsilon)$$

with density q_ϕ then for any function f of z

$$\int q_\phi(z) f(z)\,\mathrm{d}z = \int f\big(g_\phi(\varepsilon)\big) p(\varepsilon)\,\mathrm{d}\varepsilon$$

Now the expectation distribution is parameter free!

What does it bring?

Once reparameterized, just taking a Monte Carlo estimate of the expectation leads to a well-behaving gradient estimate (more details in [1])

In practice:

1. compute a finite MC estimate
2. use autodiff directly on the computed expression

Reparameterization: Gaussian example

For the Gaussian family

$$\left\{\mathcal{N}(\mu, \sigma^2) \mid \mu \in \mathbb{R}, \sigma \in \mathbb{R}^*\right\}$$

one can choose

$$\varepsilon \sim \mathcal{N}(0, 1)$$
$$g_{\mu,\sigma}(\varepsilon) = \mu + \sigma\varepsilon$$

Our first variational inference algorithm

Consider the Gaussian regression model:

$$Y_i \sim \mathcal{N}(\beta x_i, 1)$$
$$\beta \sim \mathcal{N}(0, 1)$$

whose log densities are

$$\log p(Y = y \mid \beta) = -\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta x_i)^2 + \text{cst}$$

$$\log p(\beta) = -\frac{\beta^2}{2} + \text{cst}$$

Our first variational inference algorithm

and the Gaussian variational family

$$\left\{ \mathcal{N}(\mu, \sigma^2) \mid \mu \in \mathbb{R}, \sigma \in \mathbb{R}^* \right\}$$

and the reparameterization seen before

Our first variational inference algorithm

In that case the target posterior is known:

$$\beta \mid Y = y \sim \mathcal{N}\left(\mu_{\text{post}}, \sigma^2_{\text{post}}\right)$$

where

$$\sigma^2_{\text{post}} = \frac{1}{1 + \sum_{i=1}^{N} x_i^2}$$

$$\mu_{\text{post}} = \frac{\sum_{i=1}^{N} x_i y_i}{1 + \sum_{i=1}^{N} x_i^2}$$

Our first variational inference algorithm

Concretely, need to compute

$$\mathrm{ELBO}(\mu, \sigma)$$

$$= \mathbb{E}_{\beta \sim q_{\mu,\sigma}} \left[\log p(Y = y \mid \beta, X = x) + \log p_\beta(\beta) - \log q_{\mu,\sigma}(\beta) \right]$$

$$= \mathbb{E}_{\varepsilon \sim p(\varepsilon)} \left[\log p(Y = y \mid \mu + \sigma\varepsilon, X = x) + p_\beta(\mu + \sigma\varepsilon)\big) - \log q_{\mu,\sigma}(\mu + \sigma\varepsilon) \right]$$

$$= \mathbb{E}_{\beta \sim q_{\mu,\sigma}} \left[-\frac{1}{2} \sum_{i=1}^{N} (y_i - (\mu + \sigma\varepsilon) x_i)^2 - \frac{(\mu + \sigma\varepsilon)^2}{2} + \log \sigma + \frac{\varepsilon^2}{2} \right]$$

Our first variational inference algorithm

and finally take a Monte Carlo estimate

$$\mathrm{ELBO}(\mu, \sigma)$$

$$\approx \frac{1}{L}\sum_{i=1}^{L} -\frac{1}{2}\sum_{i=1}^{N}\left(y_i - \left(\mu + \sigma\varepsilon^{(l)}\right)x_i\right)^2 - \frac{\left(\mu + \sigma\varepsilon^{(l)}\right)^2}{2} + \log\sigma + \frac{\varepsilon^{(l)^2}}{2}$$

which can be safely (and automatically) differentiated

How does it look like codewise?

Using pytorch

```
def elbo(eps, mu, sigma, xs, ys):
  beta = mu + sigma * eps

  log_lik = torch.sum(- (ys - torch.outer(beta, xs)) ** 2 / 2,
                      axis = 1)
  p_log_prior = - beta ** 2 / 2
  q_log_prior = - eps ** 2 / 2 + torch.log(sigma)

  return torch.mean(log_lik)
         + torch.mean(p_log_prior) + torch.mean(q_log_prior)
```

How does it look like codewise?

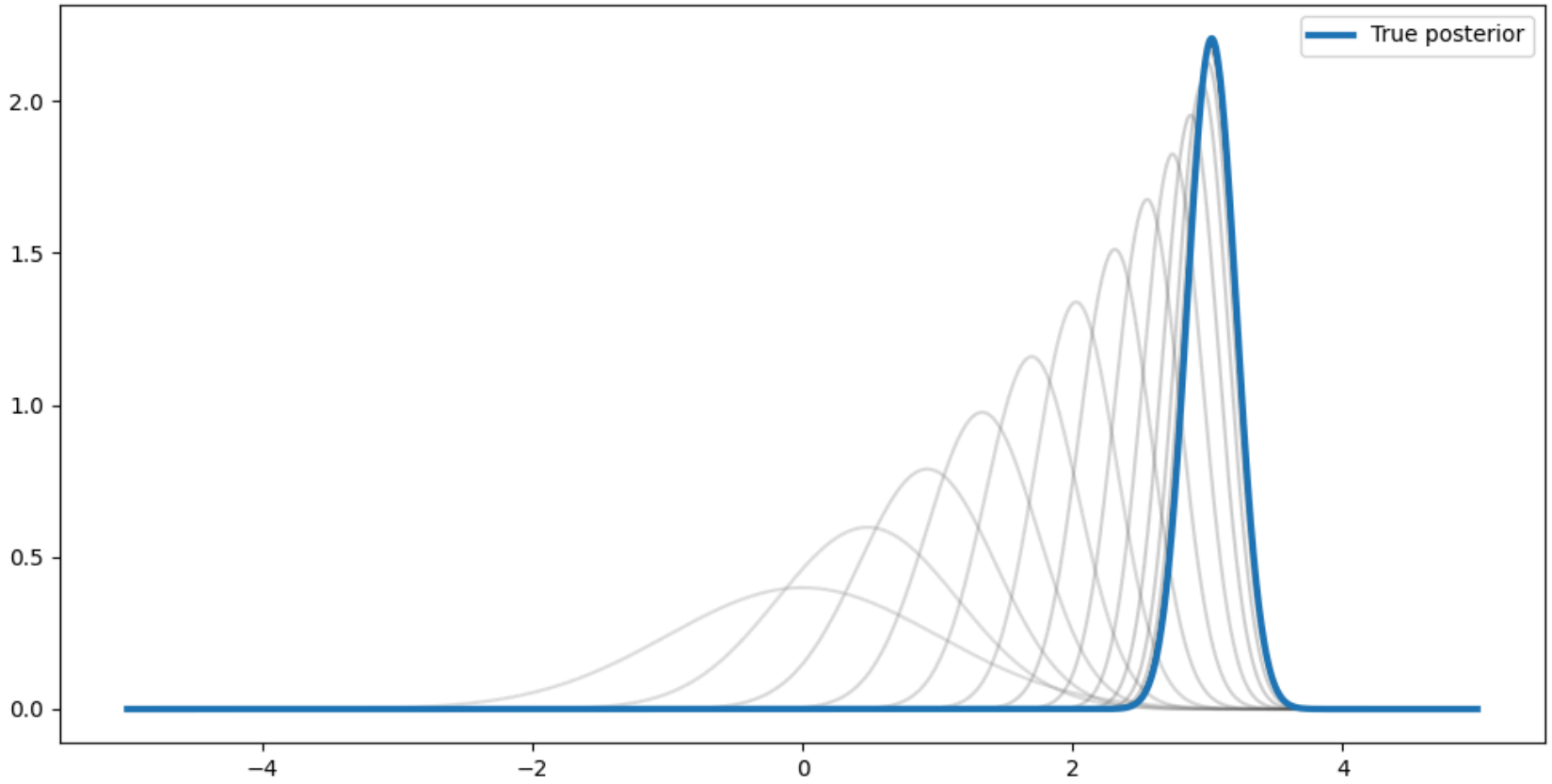
```
def VI(xs, ys, L = 100):
  eps = torch.randn(L)

  mu = torch.tensor(0., requires_grad = True)
  log_sigma = torch.tensor(0., requires_grad = True)
  optimizer = torch.optim.Adam([mu, log_sigma], lr = 1e-3)

  for step in range(10000):
    loss = - elbo(eps, mu, torch.exp(log_sigma), xs, ys)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

  return (mu, log_sigma)
```

Convergence illustration



How to choose a variational family?

An ideal variational distribution is...

- easy to sample from
  - Monte Carlo estimate of the ELBO
  - actual sampling of the posterior
- flexible enough to match the posterior
- has a tractable density
  - needed for the ELBO computation
- is compatible with the reparameterization trick

Reparameterization-friendly distributions [2]

- tractable inverse CDF applied to $\mathcal{U}(0, 1)$
  - Exponential, Cauchy, Logistic, Rayleigh, Pareto, Weibull, Gompertz, Gumbel and Erlang
- location-scale type
  - Gaussian, Laplace, Logistic, Student's t, Uniform, Triangular
- compatible with the implicit reparameterization technique [3]
  - Gamma, Beta/Dirichlet, von Mises

Mean-field variational family

To approximate a joint posterior $p(z_1, ..., z_K \mid X = x)$ a simple choice is to treat each variable independently

$$q_\phi(z_1, ..., z_K) = \prod_{i=1}^{K} q_{\phi_i}^{(i)}(z_i)$$

Mean-field variational family

To approximate a joint posterior $p(z_1, ..., z_K \mid X = x)$ a simple choice is to treat each variable independently

$$q_\phi(z_1, ..., z_K) = \prod_{i=1}^{K} q_{\phi_i}^{(i)}(z_i)$$

- may lead to analytical integrals
- but cannot represent posterior correlations!

Example: 2D Gaussian regression

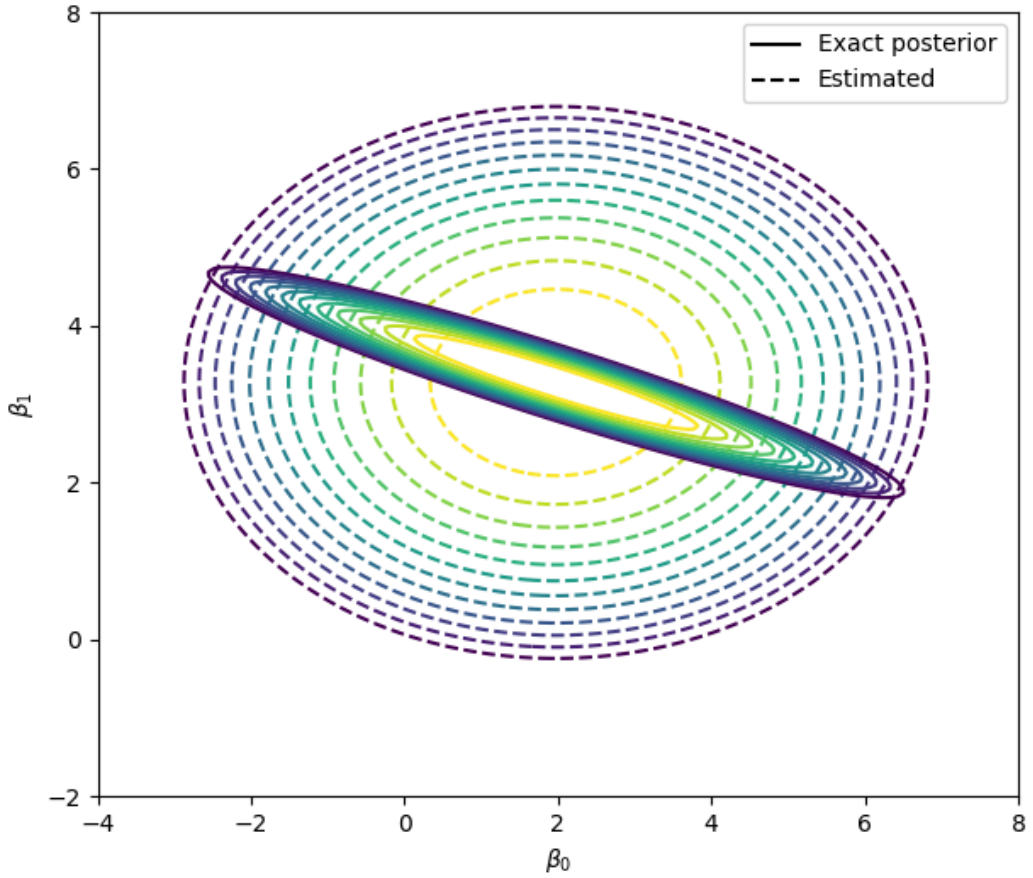
Model

$$Y_i \sim \mathcal{N}(\beta_1 x_i + \beta_0, 1)$$
$$\beta_0, \beta_1 \sim \mathcal{N}(0, 1)$$

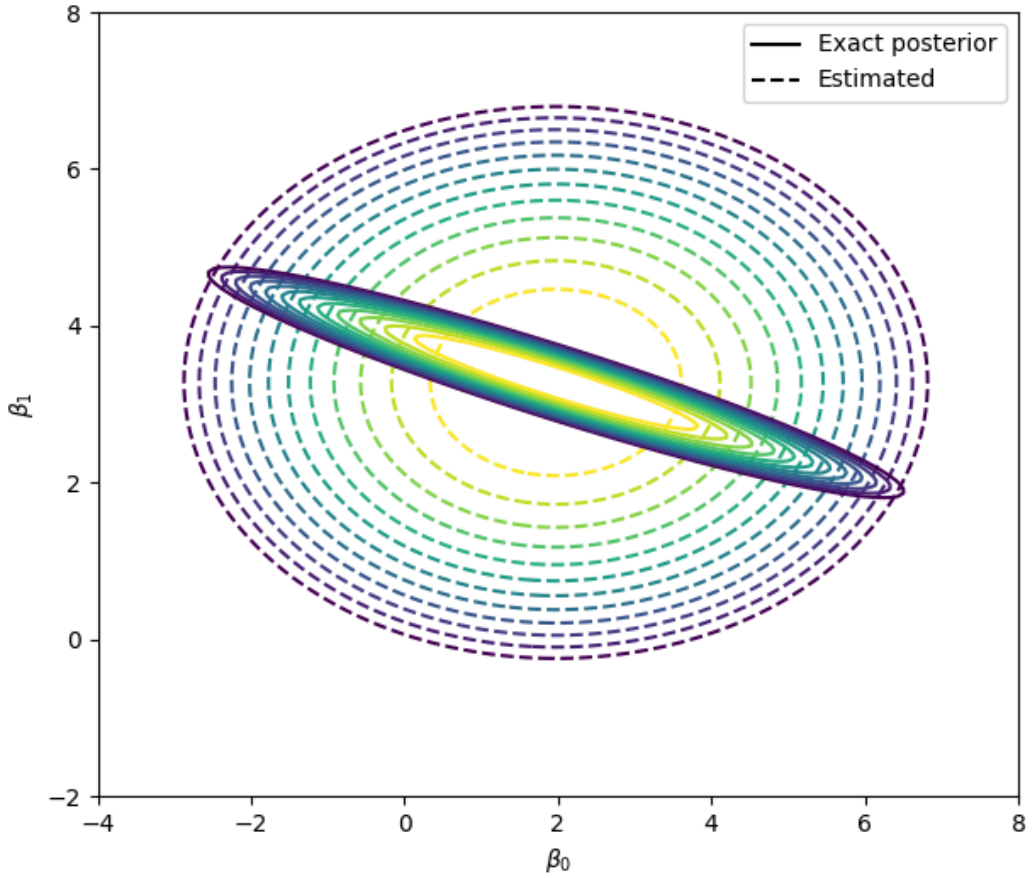
Variational family

$$\varepsilon \sim N(0, I)$$
$$\beta = \begin{pmatrix} \mu_0 \\ \mu_1 \end{pmatrix} + \begin{pmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{pmatrix} \varepsilon$$

Example: 2D Gaussian regression



Example: 2D Gaussian regression



fix: use multivariate Gaussian for the variational family

Normalizing flows: motivations

We can construct new distributions by

1. choosing a simple base distribution
2. applying a function to reshape it, like in:

$$\varepsilon \sim \mathcal{N}(0, 1)$$
$$z = g(\varepsilon)$$

Problem

- what is the density of z?
- necessary for computing the ELBO!

Normalizing flows: motivations

We can construct new distributions by

1. choosing a simple base distribution
2. applying a function to reshape it, like in:

$$\varepsilon \sim \mathcal{N}(0, 1)$$
$$z = g(\varepsilon)$$

Problem

- what is the density of z?
- necessary for computing the ELBO!

One case where this is possible is when g is a **diffeomorphism**

Change of variable

Let $g : \mathbb{R}^N \to \mathbb{R}^n$ be a **differentiable** and **invertible** function

If

$$\varepsilon \sim q_\varepsilon$$
$$z = g(\varepsilon)$$

Then

$$q_z(g(\varepsilon)) = \left|J_{g^{-1}}(g(\varepsilon))\right| q_\varepsilon(\varepsilon) = \left|J_g(g(\varepsilon))\right|^{-1} q_\varepsilon(\varepsilon)$$

where J_g is the Jacobian of g.

(bonus: it is just the right form to use the reparameterization trick!)

Families of invertible mappings

Many proposals for function families s.t.

1. functions are diffeomorphisms
2. determinant of the Jacobian can be efficiently computed
3. flexible enough to adjust to data (e.g. using multi-layer perceptrons)

See [4] for a long list:

- linear, planar, radial, coupling, autoregressive flows

An example: affine coupling [5]

$$y_{1:d} = x_{1:d}$$
$$y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d})$$

where s and t are multi-layer perceptrons, $\odot$ is coordinate-wise multiplication

An example: affine coupling [5]

$$y_{1:d} = x_{1:d}$$
$$y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d})$$

where s and t are multi-layer perceptrons, $\odot$ is coordinate-wise multiplication

- affine $\Rightarrow$ easy to invert
- upper half is identity
  - $\Rightarrow$ Jacobian is triangular
  - $\Rightarrow$ determinant is the product of diagonal elements

$$J = \begin{pmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}} & \operatorname{diag}\left(\exp s(x_{1:d})\right) \end{pmatrix}$$

Flows

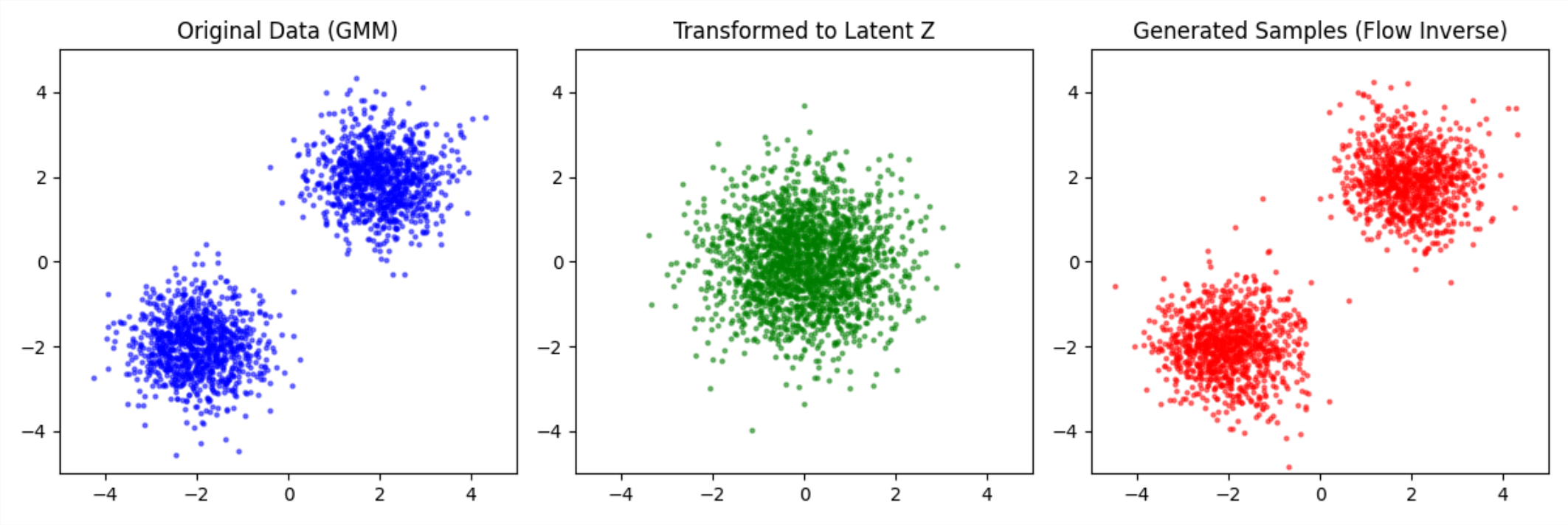
Noting that

- function composition preserves inversibility
- $\det(f_1 \circ ... \circ f_n) = \prod_i \det f_i$

it's tempting to consider functions built as compositions of (simpler) invertible differentiable mappings

- that's what's called a **flow** (the initial density 'flows' through the sequence)
- it's normalizing because we ensure the resulting function sums to 1
- can achieve arbitrarily complex transformations (see universality results [6])

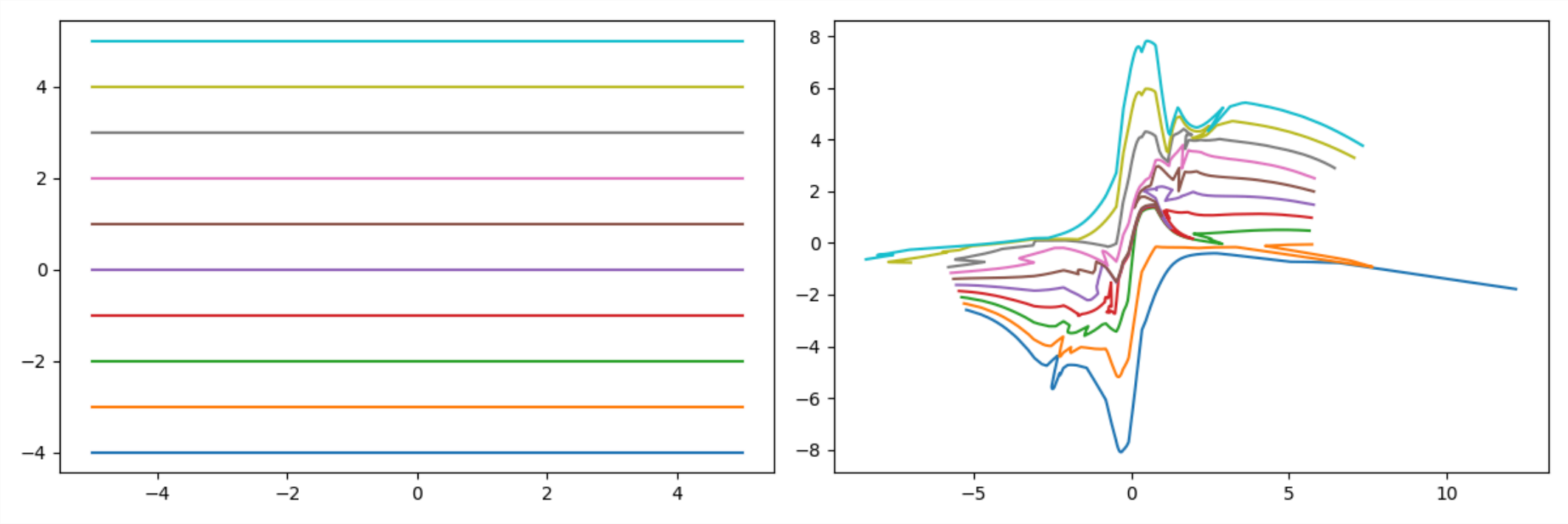
Illustration on a Gaussian mixture model



(obtained with 4 affine coupling layers equipped with 3 layer MLPs of 8 hidden units)

Illustration on a Gaussian mixture model

What the learned transform does to straight lines:



Epilogue: where VI really shines

Let's spice it up a bit

Assume we have a LARGE number N of i.i.d. observations x_i, with the following structure

$$Z_i \sim \mathcal{D}_Z(\theta)$$
$$X_i \sim \mathcal{D}_X(\theta, Z_i)$$

Let's spice it up a bit

Assume we have a LARGE number N of i.i.d. observations x_i, with the following structure

$$Z_i \sim \mathcal{D}_Z(\theta)$$
$$X_i \sim \mathcal{D}_X(\theta, Z_i)$$

We'd like to produce:

- a point estimate θ^* for θ
- an approximation to $p(z \mid x, \theta^*)$ for any new x

(typically there will be low uncertainty on θ because we have so much data)

Applying VI

Variational family:

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

Applying VI

Variational family:

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

Now the ELBO depends on θ

$$\mathrm{ELBO}(\phi, \theta) = \mathbb{E}_{z \sim q_\phi}[\log p_\theta(X = x \mid z)] - \mathrm{KL}\big(q_\phi(z) \parallel p_\theta(z)\big)$$

and we achieve our goal by jointly maximizing on θ and ϕ

Exploiting independence

$$
\begin{aligned}
\mathrm{ELBO}(\phi, \theta) &= \mathbb{E}_{z \sim q_\phi}[\log p_\theta(X = x \mid z)] - \mathrm{KL}\big(q_\phi(z) \parallel p_\theta(z)\big) \\
&= \sum_{i=1}^{N} \mathbb{E}_{z_i \sim q_{\phi_i}}[\log p_\theta(X_i = x_i \mid z_i)] - \mathrm{KL}\Big(q_{\phi_i}(z_i) \parallel p_\theta(z_i)\Big) \\
&= \sum_{i=1}^{N} \mathrm{ELBO}_i(\phi_i, \theta)
\end{aligned}
$$

Approximating the gradient with a subset of data

$$
\begin{aligned}
\nabla_\theta \, \mathrm{ELBO}(\phi, \theta) &= \sum_{i=1}^{N} \nabla_\theta \mathrm{ELBO}_i(\phi_i, \theta) \\
&= N \left(\frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \mathrm{ELBO}_i(\phi_i, \theta) \right) \\
&\approx N \left(\frac{1}{M} \sum_{l=1}^{M} \nabla_\theta \mathrm{ELBO}_{i_l}\left(\phi_{i_l}, \theta\right) \right)
\end{aligned}
$$

where $M \ll N$

A subset of the data used to approximate the gradient is called a **mini-batch**

Stochastic Gradient Descent (SGD)

1. choose an arbitrary initial point x
2. repeat
   - until all examples are used:
     - randomly draw without replacement a mini-batch
     - compute an approximate gradient
     - update x
   - if convergence criterion is met, return x

An iteration of this algorithm is called an **epoch**

After one epoch, all examples have been used exactly once

Stochastic Gradient Descent (SGD)

SGD has been instrumental in the success of deep learning:

- enables learning with millions of data points
- less prone to being trapped in local optima

Amortized inference

After ELBO maximization we estimated the optimal θ and posterior for all observations

- assume there is a new observation
- to get the corresponding posterior, we need to run a new ELBO maximization
- somehow all the work done before (except for the θ estimate) has been wasted

Amortized inference

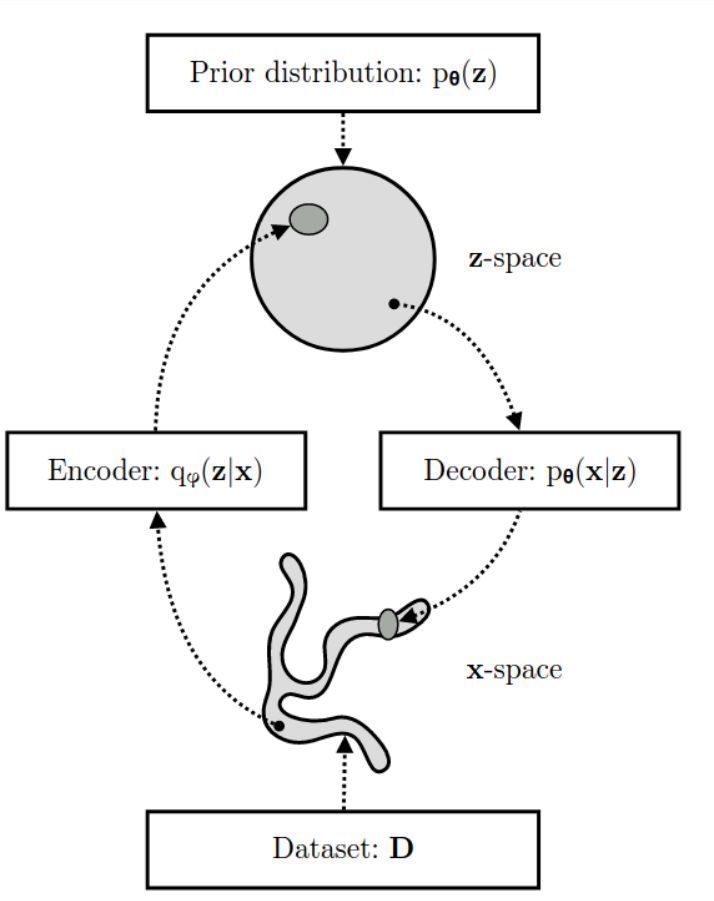
A (super) cool idea: can we **learn** to estimate the posterior of z given an observation x?

- typically, a neural network that takes x as an input
- and outputs the parameters of a distribution
- then when a new observation comes, inference is **super** fast!

Amortized inference

In **variational auto-encoders** architectures (VAE, [2])

- variational parameters encode a **function** from x to a variational distribution $q(z \mid x)$
- ELBO optimization has far less variables



[7]

Wrap up

- VI is Bayesian inference seen as an optimization/approximation problem
- the ELBO is the key quantity that is optimized
  - measures closeness between approx and target posterior distribution
  - is an estimate of marginal likelihood
- VI enables uses of arbitrarily complex distributions
  - useful for inference
  - **but also for specifying more accurate model**
- thanks to batch learning, scales to very large datasets
- with VAE, very efficient inference after training phase
- of course there are limitations and caveats, but hey, this was just a teaser!

Bibliography

[1] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning," Journal of Machine Learning Research, vol. 21, no. 132, pp. 1–62, 2020.

[2] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," arXiv preprint arXiv:1312.6114, 2013.

[3] M. Figurnov, S. Mohamed, and A. Mnih, "Implicit reparameterization gradients," Advances in neural information processing systems, vol. 31, 2018.

[4] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing flows: An introduction and review of current methods," IEEE transactions on pattern analysis and machine intelligence, vol. 43, no. 11, pp. 3964–3979, 2020.

[5] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using Real NVP," in 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=HkpbnH9lx

[6] F. Draxler, S. Wahl, C. Schnörr, and U. Köthe, "On the universality of volume-preserving and coupling-based normalizing flows," in Proceedings of the 41st International Conference on Machine Learning, in ICML'24. Vienna, Austria: JMLR.org, 2024.

[7] D. P. Kingma, M. Welling, and others, "An introduction to variational autoencoders," Foundations and Trends® in Machine Learning, vol. 12, no. 4, pp. 307–392, 2019.