ENM 360: Introduction to Data-driven Modeling

Lecture #25: Variational inference

Paris Perdikaris

November 24, 2020

Bayesian inference

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

MCMC: cleverly construct a Markov Chain whose stationary distribution approximates $p(\theta|\mathcal{D})$

- It's asymptotically exact, meaning that if we run the chain long enough, we should get samples from the posterior distribution

- Tuning MCMC can be tricky

- In practice requires thousands of samples, and each draw involves multiple (likely tens, perhaps hundreds) of evaluations of the un-normalised log posterior.

- This is fine for many models, but sometimes it just takes too long.

- It is hard to scale to large data-sets and models with many parameters and singular posteriors (e.g. deep learning).

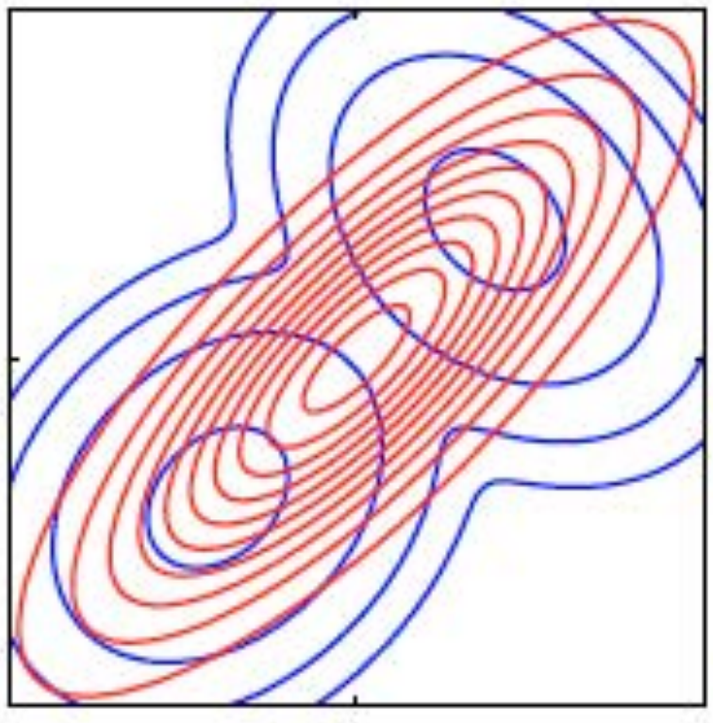
Variational inference

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

- Variational inference is another way to do Bayesian inference.
- The idea is that we'll approximate the posterior distribution with a family of distributions that is easy to work with.
- It will provide us with a set of tools for transforming the sampling problem (integration) to an optimization problem, that can be scaled to large models (i.e. with many parameters) and large data-sets.
- It also tends to favor approximations that underestimate the variance, and it usually will result in approximate distributions that get the means right but underestimate the variance.

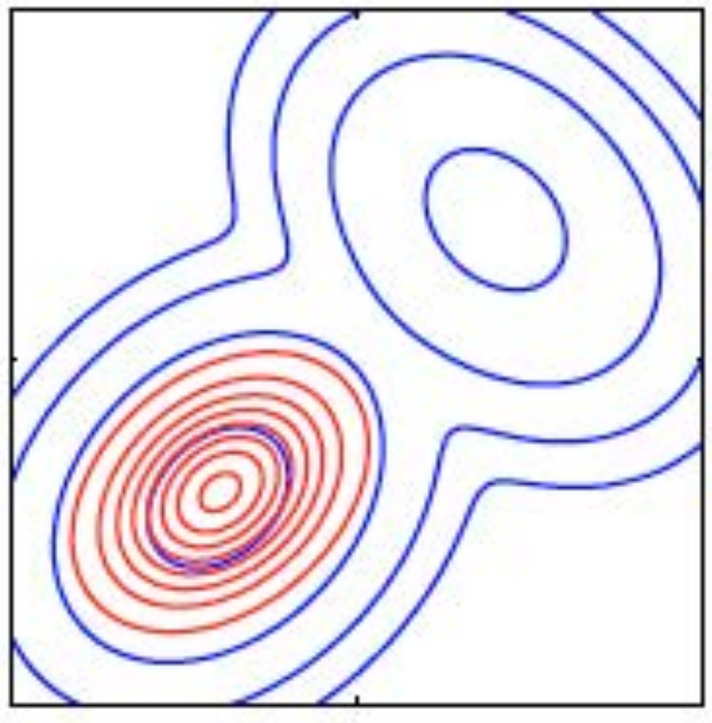
Variational inference

$\mathbb{KL}[q_\phi(x)||p(x)]$



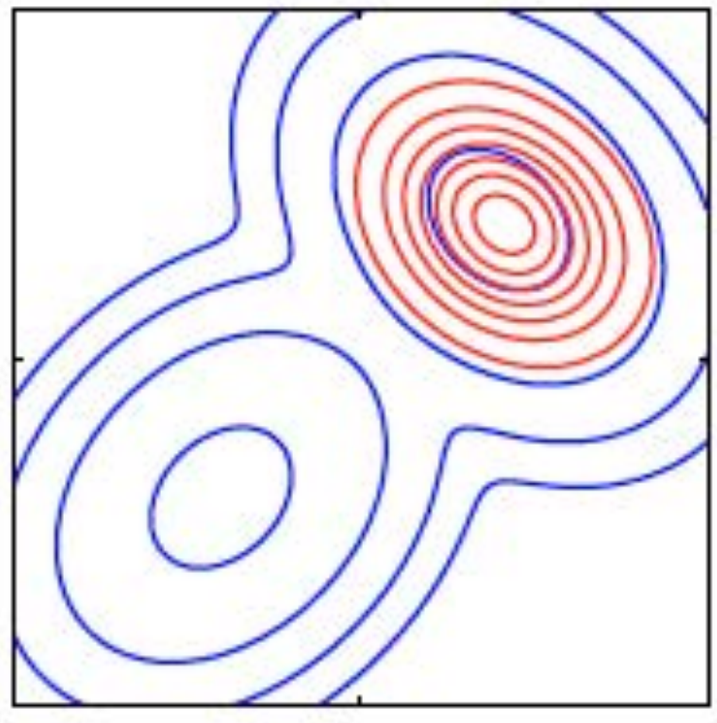
KL

$\mathbb{KL}[p(x)||q_\phi(x)]$



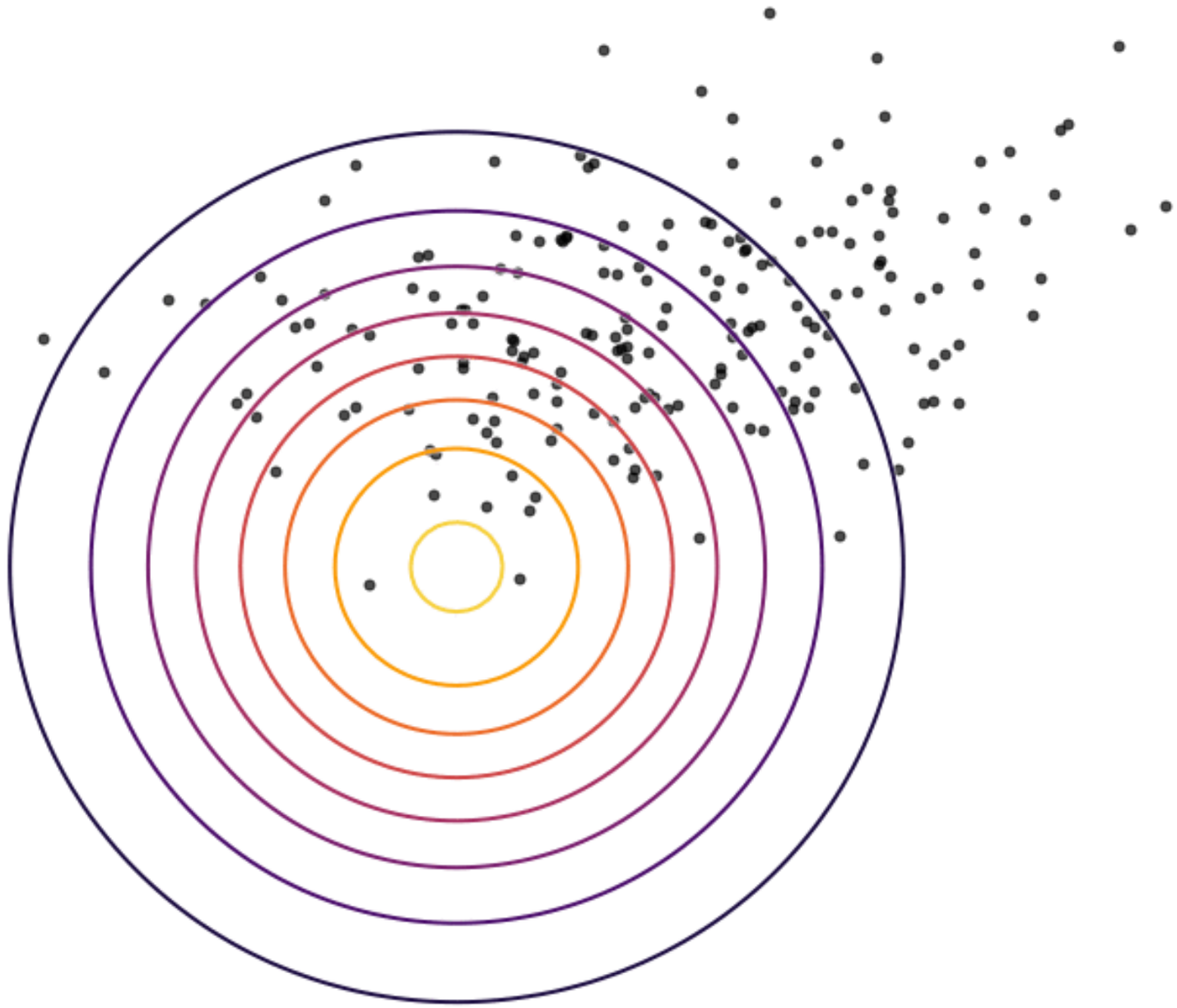
Reverse KL

$\mathbb{KL}[p(x)||q_\phi(x)]$



Reverse KL

Variational inference



The re-parametrization trick

One oft-encountered problem is computing the gradient of an expectation of a smooth function *f*:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \nabla_\theta \int p(z;\theta) f(z) dz$$

This is a recurring task in machine learning, needed for posterior computation in variational inference, value function and policy learning in reinforcement learning, derivative pricing in computational finance, and inventory control in operations research, amongst many others. This gradient is often difficult to compute because the integral is typically unknown and the parameters θ, with respect to which we are computing the gradient, are of the distribution $p(z;\theta)$. But where a random variable z appears we can try our random variable reparameterisation trick, which in this case allows us to compute the gradient in a more amenable way:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g(\epsilon,\theta))]$$

http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/

The re-parametrization trick

Let's derive this expression and explore the implications of it for our optimisation problem. One-liners give us a transformation from a distribution $p(\epsilon)$ to another $p(z)$, thus the differential area (mass of the distribution) is invariant under the change of variables. This property implies that:

$$p(z) = \left|\frac{d\epsilon}{dz}\right| p(\epsilon) \implies |p(z)dz| = |p(\epsilon)d\epsilon|$$

Re-expressing the troublesome stochastic optimisation problem using random variate reparameterisation, we find:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \nabla_\theta \int p(z;\theta) f(z) dz$$

$$= \nabla_\theta \int p(\epsilon) f(z) d\epsilon = \nabla_\theta \int p(\epsilon) f(g(\epsilon,\theta)) d\epsilon$$

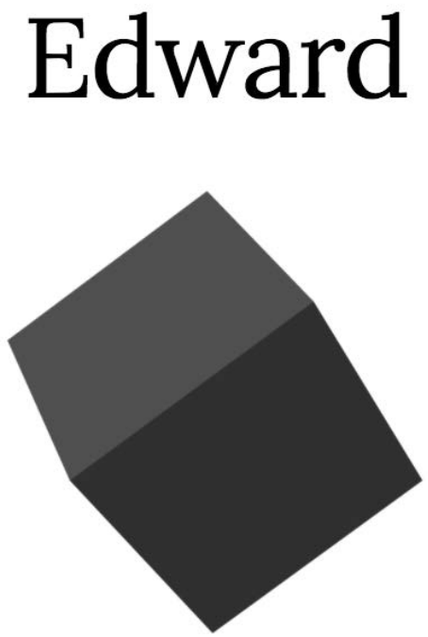
$$= \nabla_\theta \mathbb{E}_{p(\epsilon)}[f(g(\epsilon,\theta))] = \mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g(\epsilon,\theta))]$$

http://www.inference.vc/design-patterns/

Probabilistic programming

http://mc-stan.org/

https://github.com/pymc-devs/pymc3



http://edwardlib.org/

https://github.com/uber/pyro