Variational Inference and Optimization II

Arto Klami

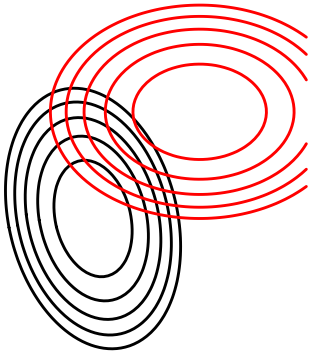
ProbAI, June 5, 2019

Bayesian inference using optimization

Even though MCMC algorithms provide samples from the posterior, the quality is quantified only implicitly. There is no clear learning objective, we just rely on the sampler being good enough

We can convert the search for the posterior distribution into an optimization problem as well

1. Choose some parametric family of distributions $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$
2. Find $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ that is close to $p(\boldsymbol{\theta}|\mathcal{D})$, by minimizing some dissimilarity measure wrt $\boldsymbol{\lambda}$



Classical VI recap

Standard VI uses Kullback-Leibler divergence between $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ and $p(\boldsymbol{\theta}|\mathcal{D})$ as the loss function

Equivalent formulation: Maximize a lower bound $\mathcal{L}_{ELBO}(\boldsymbol{\lambda})$ for $\log p(\mathcal{D})$:

$$\begin{aligned}\mathcal{L}_{ELBO}(\boldsymbol{\lambda}) &:= \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] - D_{KL}(q(\boldsymbol{\theta})|p(\boldsymbol{\theta})) \\ &= \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log p(\mathcal{D}, \boldsymbol{\theta})\right] + \mathcal{H}(q(\boldsymbol{\theta})) \leq \log p(\mathcal{D})\end{aligned}$$

Classical VI recap

Mean-field approximation $q(\boldsymbol{\theta}) = \prod_{d=1}^{D} q_d(\boldsymbol{\theta}_d)$ leads to very elegant coordinate ascent algorithm with

$$q_d(\boldsymbol{\theta}_d) \propto e^{\mathbb{E}_{q_{-d}(\boldsymbol{\theta})}[\log p(\mathcal{D}, \boldsymbol{\theta})]}$$

or equivalently

$$q_d(\boldsymbol{\theta}_d) \propto e^{\mathbb{E}_{q_{-d}(\boldsymbol{\theta})}[\log p(\boldsymbol{\theta}_d | \mathcal{D}, \boldsymbol{\theta}_{-d})]}$$

If these conditionals are in exponential family (with conjugate priors), we get simple update rules that only require computing expected natural parameters over other factors

Classical VI recap

Despite the nice analytic updates, the classical approach has serious limitations

- Slow and error-prone derivations
- Limited to mean-field
- Limited to conditionally conjugate models

Ideally we would want to get rid of all of these, to make VI practical for applied data analysis tasks

Detour: From neural networks to deep learning

My first task as an intern 20 years ago was to implement small multi-layer perceptron for class density estimation. I manually derived the gradients and implemented conjugate gradients in Matlab. It took most of the summer, and was very fragile

Today we would simply write something like

```
model = Sequential()
model.add(Dense(N, activation='relu'))
model.add(Dense(numClass, activation='softmax'))
opt = keras.optimizers.rmsprop(lr=0.001, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=opt)
model.fit(x, y, epochs=nIter)
```

The key ingredients here are automatic computation of the gradients for any model and good, easy-to-use optimization tools

Detour: Reverse-mode automatic differentiation

Assume a function $f(\boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}$ (think of loss functions or $\log p(\cdot)$)

Reverse-mode automatic differentiation allows computing all D partial derivatives of that function with small computational overhead

Split the function into simple expressions, store all intermediate results, and use derivatives of the simple expression to construct code that computes the derivative

Corresponds to backpropagation for neural networks, but works for general functions

Detour: Reverse-mode automatic differentiation

For $f(x_1, x_2) = x_1^2 + \cos(x_2^3)$ we first compute the intermediate results and then form the derivative processing the path backwards, starting with $\frac{\partial f}{\partial x_6} = 1$

Variable	Computation	Derivative
x_3	x_1^2	$\frac{\partial x_3}{\partial x_1} = 2x_1$
x_4	x_2^3	$\frac{\partial x_4}{\partial x_2} = 3x_2^2$
x_5	$\cos(x_4)$	$\frac{\partial x_5}{\partial x_4} = \sin(x_4)$
x_6	$x_3 + x_5$	$\frac{\partial x_6}{\partial x_3} = 1, \frac{\partial x_6}{\partial x_5} = 1$

$$\frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_6}\frac{\partial x_6}{\partial x_5} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_5}\frac{\partial x_5}{\partial x_4} = 1 \times \sin(x_4) = \sin(x_4)$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_6}\frac{\partial x_6}{\partial x_3} = 1$$

$$\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial x_4}\frac{\partial x_4}{\partial x_2} = \sin(x_4)3x_2^2$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_3}\frac{\partial x_3}{\partial x_1} = 1 \times 2x_1 = 2x_1$$

Detour: Modern stochastic gradient descent

Now we know how to compute gradients of arbitrary loss functions $\mathcal{L}(\mathcal{D}, \boldsymbol{\lambda})$ with respect to the parameters $\boldsymbol{\lambda}$ of the model

Most losses are sums over individual data instances, such that $\mathcal{L}(\mathcal{D}, \boldsymbol{\lambda}) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, \boldsymbol{\lambda})$, and for optimization we would typically use stochastic gradient descent

After initialization we iterate for

1. Obtain stochastic estimate for the gradient based on a subset of the data instances (here just one): $\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\mathcal{D}, \boldsymbol{\lambda}_t) \approx \nabla_{\boldsymbol{\lambda}} L(x_i, \boldsymbol{\lambda}_t) =: g(\boldsymbol{\lambda})$
2. Update the parameters $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_t g(\boldsymbol{\lambda})$ using some step length η_t

During optimization η_t should decrease at suitable pace, typically adaptively (AdaGrad, Adam, RMSProp, ...)

Surprisingly tricky to implement well, but good robust solutions exist in all reasonable libraries

Probabilistic programming

Probabilistic programming is currently transforming Bayesian modeling in similar manner as what happend in deep learning, by providing (a) easy interface for specifying models and (b) automatic and model-free inference

In fact, MCMC always was model-free; Metropolis-Hastings only requires evaluating log-likelihoods, but naive implementations are slow and fragile

Bugs provided a PP enrivonment for conjugate models already a long time ago, and today Stan [Carpenter et al., 2017] offers one for arbitrary differentiable models, thanks to improvements in Hamiltonian Monte Carlo

Variational approximation in probabilistic programming

The situation is worse for VI: Instead of making existing algorithms easy-to-use, we need new ones since the one we have seen works only for limited models and approximations

The question is: Can we do VI for arbitrary models and approximations, without

(a) Making the mean-field assumption

(b) Assuming conjugate models

(c) ...or even knowledge of the model when writing the inference algorithm

Variational approximation in probabilistic programming

The situation is worse for VI: Instead of making existing algorithms easy-to-use, we need new ones since the one we have seen works only for limited models and approximations

The question is: Can we do VI for arbitrary models and approximations, without

(a) Making the mean-field assumption

(b) Assuming conjugate models

(c) ...or even knowledge of the model when writing the inference algorithm

Yes, and under rather mild conditions. Besides being able to evalute $\log p(\mathcal{D}, \boldsymbol{\lambda})$ we need to assume either of the following:

1. The approximation $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ is differentiable wrt to $\boldsymbol{\lambda}$
2. The model $p(\mathcal{D}, \boldsymbol{\theta})$ is differentiable wrt to $\boldsymbol{\theta}$ and the approximation can be reparameterized

Variational approximation in probabilistic programming

Monte Carlo approximation for the bound itself is easy using

$$\mathcal{L}_{ELBO}(\boldsymbol{\lambda}) \approx \frac{1}{M} \sum_{m=1}^{M} \log \frac{p(\mathcal{D}, \boldsymbol{\theta}_m)}{q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})},$$

where $\boldsymbol{\theta}_m$ are drawn from the approximation. For $\log p(\mathcal{D})$ this would be severely biased, but for the bound it is okay

For optimization we would want (approximate) gradient, to be used for (stochastic) gradient ascent

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \alpha \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda})$$

Automatic differentiation can be used for both $\log q(\cdot)$ and $\log p(\cdot)$ (assuming they are differentiable), but the challenge is in handling the expectation

Differentiating expectations

Changing the order of integral and derivative is here perfectly valid, but unfortunately does not really help

For simplicity, focus on $\mathbb{E}_{q(\boldsymbol{\theta})}[\log p(\mathcal{D}, \boldsymbol{\theta})]$, since handling the entropy is easy. We have

$$\nabla_{\boldsymbol{\lambda}_d} \mathbb{E}_{q(\boldsymbol{\theta})}[\log p(\mathcal{D}, \boldsymbol{\theta})] = \int \nabla_{\boldsymbol{\lambda}_d} q(\boldsymbol{\theta}|\boldsymbol{\lambda}) \log p(\mathcal{D}, \boldsymbol{\theta}) d\boldsymbol{\theta},$$

which is not an expectation and hence cannot be evaluated using Monte Carlo

However, there are two alternative strategies for computing the gradients: score function estimators and reparameterization

Score-function estimator

Chain-rule of differentiation gives

$$\nabla \log f(\boldsymbol{\theta}) = \frac{\nabla f(\boldsymbol{\theta})}{f(\boldsymbol{\theta})} \qquad \rightarrow \qquad \nabla f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) \nabla \log f(\boldsymbol{\theta})$$

The derivative of the expected log-likelihood then becomes

$$\begin{aligned} \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\theta}}} \left[\log p(\mathcal{D}, \boldsymbol{\theta})\right] &= \int \log p(\mathcal{D}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} \\ &= \int \log p(\mathcal{D}, \boldsymbol{\theta}) \left[q(\boldsymbol{\theta}|\boldsymbol{\lambda}) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right] d\boldsymbol{\theta} = \mathbb{E}_{q_{\boldsymbol{\theta}}} \left[\log p(\mathcal{D}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right] \end{aligned}$$

which can be approximated using

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\theta}}} \left[\log p(\mathcal{D}, \boldsymbol{\theta})\right] \approx \frac{1}{M} \sum_{m=1}^{M} \log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda}) = \frac{1}{M} \sum_{m=1}^{M} w_m \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$$

for $\boldsymbol{\theta}_m \sim q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ and $w_m = \log p(\mathcal{D}, \boldsymbol{\theta}_m)$

Score-function estimator

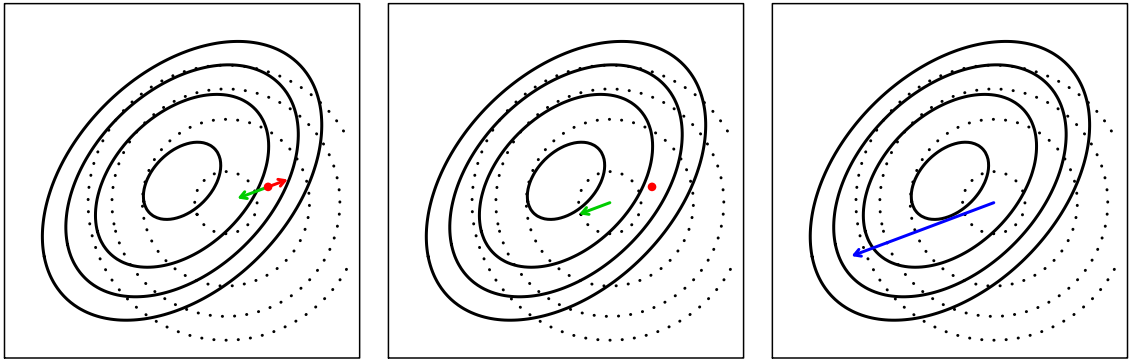
Even though the estimator

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\theta}}} \left[\log p(\mathcal{D}, \boldsymbol{\theta}) \right] \approx \frac{1}{M} \sum_{m=1}^{M} \log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m | \boldsymbol{\lambda})$$

is valid and unbiased, it has very high variance

- The range of $\log p(\mathcal{D}, \boldsymbol{\theta}_m)$ can be very large, so only the largest elements matter for the expectation
- The likelihood does not directly contribute to the direction of gradient

Score-function estimator

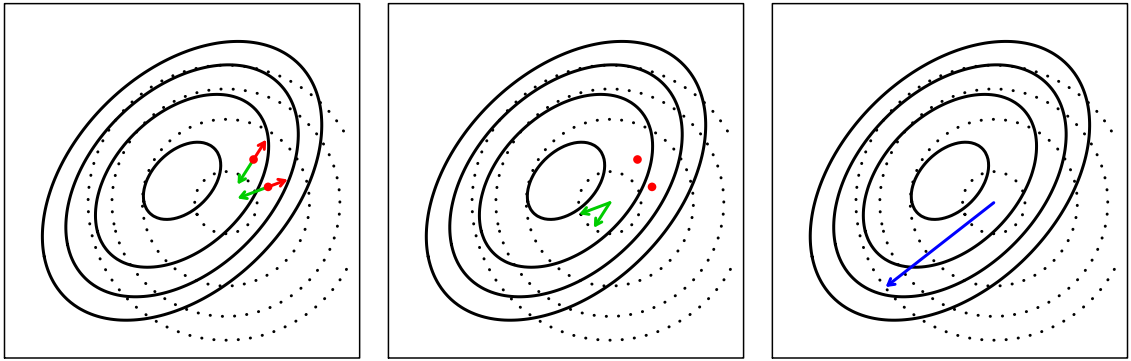


$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$ $\log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

$\frac{1}{M} \sum_{m=1}^{M} \log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

Note: Scales altered for better visualization

Score-function estimator

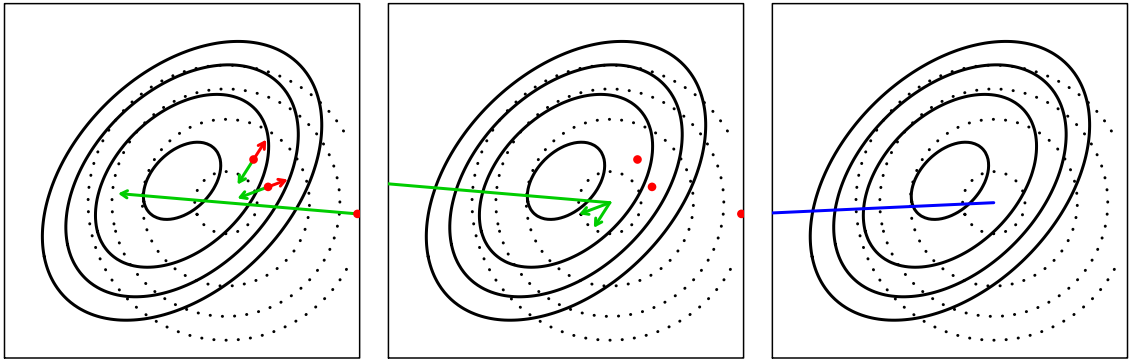


$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$ $\log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

$\frac{1}{M} \sum_{m=1}^{M} \log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

Note: Scales altered for better visualization

Score-function estimator

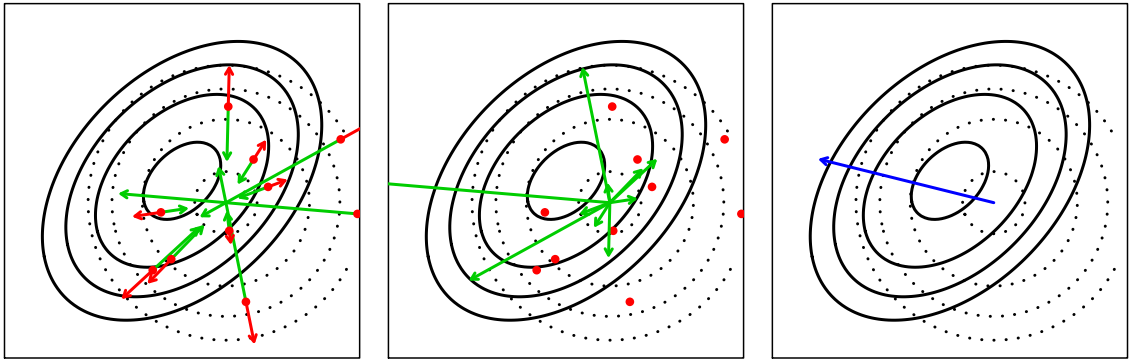


$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$ $\log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

$\frac{1}{M} \sum_{m=1}^{M} \log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

Note: Scales altered for better visualization

Score-function estimator

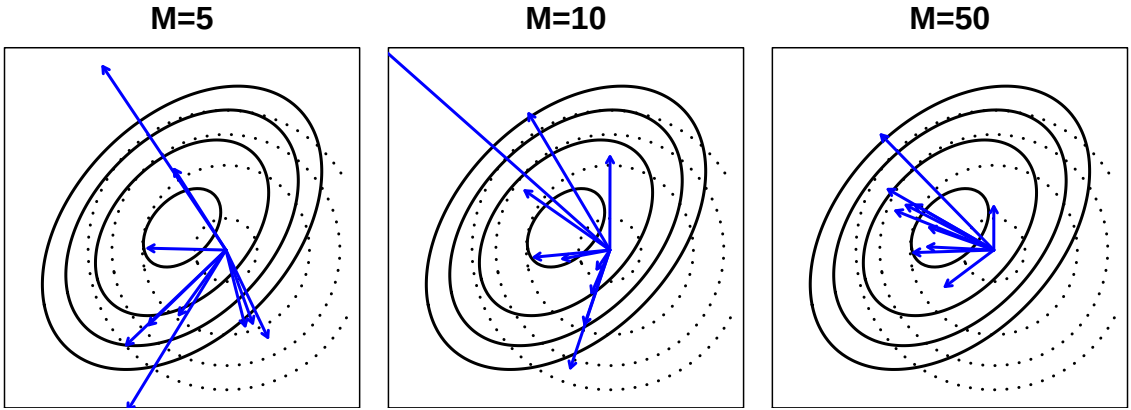


$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$ $\log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

$\frac{1}{M} \sum_{m=1}^{M} \log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

Note: Scales altered for better visualization

Score-function estimator



$\frac{1}{M}\sum_{m=1}^{M} \log p(\mathcal{D}, \boldsymbol{\theta}_m) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$

Note: Scales altered for better visualization

Score-function estimator

To create a practical black-box VI algorithm [Ranganath et al., 2014] based on the score function estimator, we need either very large M or other means for reducing the variance

Control variates: Assuming we already know $\mathbb{E}[g(X)]$ for some function $g(\cdot)$, we can compute

$$\mathbb{E}[f(X)] = \mathbb{E}[f(X) - g(X)] + \mathbb{E}[g(X)]$$

due to linearity of expectation. The variance of the first term is

$$\mathrm{Var}(f - g) = \mathrm{Var}(f) - 2\mathrm{Cov}(f, g) + \mathrm{Var}(g)$$

which is smaller than $\mathrm{Var}(f)$ for positively correlated functions

Variance reduction

Apply the control variate idea by using $g(X) = C\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})$, resulting in

$$\mathbb{E}_{q_{\boldsymbol{\theta}}} \left[\log p(\mathcal{D}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda}) - C\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right] + \mathbb{E}_{q_{\boldsymbol{\theta}}} \left[C\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right]$$

By again applying the log-derivative identity, the last term simplifies to

$$C\mathbb{E}_{q_{\boldsymbol{\theta}}} \left[\frac{\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda})}{q(\boldsymbol{\theta}|\boldsymbol{\lambda})}\right] = C \int \nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} = C\nabla_{\boldsymbol{\lambda}} \int q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} = 0$$

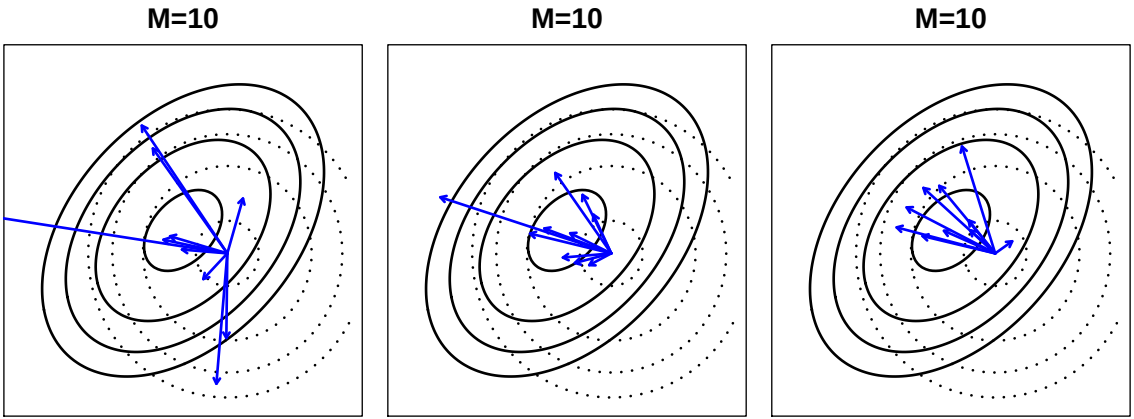
Variance reduction

Having got rid of the last term, the estimator becomes

$$\begin{aligned}&\mathbb{E}_{q_\theta}\left[\log p(\mathcal{D}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda}) - C \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right] \\ &= \mathbb{E}_{q_\theta}\left[(\log p(\mathcal{D}, \boldsymbol{\theta}) - C) \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right],\end{aligned}$$

where we can choose C however we want to, in an attempt to reduce the variance – the equation is correct for all choices, including ones that depend e.g. on the data instance

Score-function estimator



Control variate $C\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ for $C = 0, C = -3,$ and $C = -6$

Reparameterization

The other main strategy for estimating gradients of ELBO builds on the idea of reparameterizing the approximation [Titsias and Lázaro-Gredilla, 2014, Kingma and Welling, 2014, Kucukelbir et al., 2017, Rezende et al., 2014]

Assume $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be written as a system

$$\mathbf{z} \sim \phi(\mathbf{z})$$
$$\boldsymbol{\theta} = f(\mathbf{z}, \boldsymbol{\lambda})$$

where $\phi(\mathbf{z})$ is some simple distribution that does not depend on $\boldsymbol{\lambda}$ and $f(\mathbf{z}, \boldsymbol{\lambda})$ is a deterministic transformation

The common example is $x \sim \mathcal{N}(\mu, \sigma)$ re-written using

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

Note that this is how we sample from $\mathcal{N}(\mu, \sigma)$ in implementations in all programming languages

Reparameterization

With such representation we can re-write

$$\mathcal{L} = \mathbb{E}_q \left[\log p(\mathcal{D}, \boldsymbol{\theta})\right] = \mathbb{E}_\phi \left[\log p(\mathcal{D}, f(\mathbf{z}, \boldsymbol{\lambda}))\right]$$

where the expectation is now over a distribution with no parameters, and the original parameters of our approximation only appear in the transformation.

Now we can easily push the derivative inside the expectation and use standard Monte Carlo

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L} = \mathbb{E}_\phi \left[\nabla_{\boldsymbol{\lambda}} \log p(\mathcal{D}, f(\mathbf{z}, \boldsymbol{\lambda}))\right] \approx \frac{1}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\lambda}} \log p(\mathcal{D}, f(\mathbf{z}_m, \boldsymbol{\lambda}))$$

for $\mathbf{z}_m \sim \phi(\mathbf{z})$

Reparameterization

Suddenly we are computing derivatives of the actual model, since

$$\nabla_{\boldsymbol{\lambda}} \log p(\mathcal{D}, f(\mathbf{z}, \boldsymbol{\lambda})) = \nabla_{\theta} \log p(\mathcal{D}, \theta) \nabla_{\boldsymbol{\lambda}} f(\mathbf{z}, \boldsymbol{\lambda})$$

Good news because now the model actually has means for influencing the gradient and the variance gets considerably smaller, but also bad news because we need to assume it can be differentiated (efficiently)

Think of Metropolis-Hastings vs Hamiltonian Monte Carlo – it is clearly beneficial to use gradients of the model is available

Reparameterization

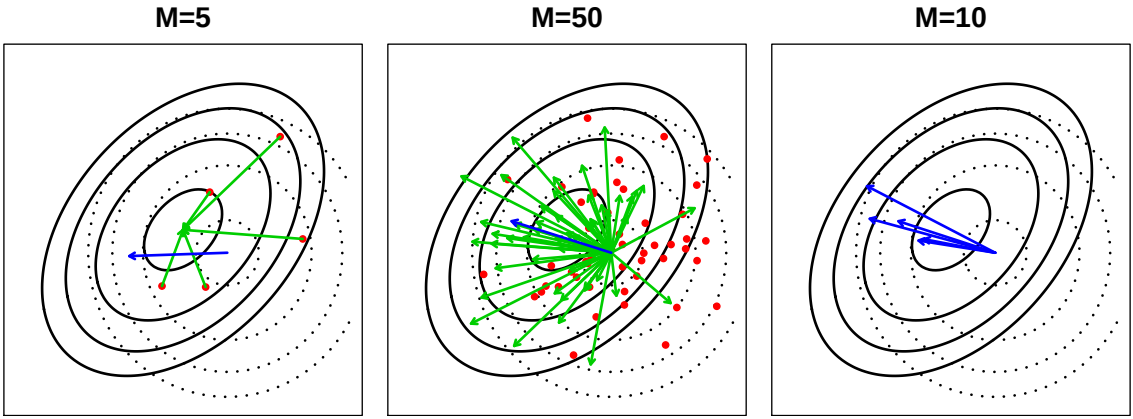
Simplest example:

$$q(\theta|\mu,\sigma) = \mathcal{N}(\mu,\sigma^2) \qquad \equiv \qquad z \sim \mathcal{N}(0,1), \quad \theta = \mu + \sigma z$$

The gradient becomes

$$\frac{\partial \mathcal{L}}{\partial \mu} = \frac{\partial \mathcal{L}}{\partial \theta}\frac{\partial f(z,\boldsymbol{\lambda})}{\partial \mu} = \frac{\partial \mathcal{L}}{\partial \theta} \qquad \frac{\partial \mathcal{L}}{\partial \sigma} = \frac{\partial \mathcal{L}}{\partial \theta}\frac{\partial f(z,\boldsymbol{\lambda})}{\partial \sigma} = \frac{\partial \mathcal{L}}{\partial \theta} z$$

Reparameterization



Reparameterization

Reparameterization can be done for:

- One-liner samplers (normal, exponential, cauchy, ...)
- Rejection samplers (gamma, beta, dirichlet, ...) [Naesseth et al., 2017]
- Chains of transformations (log-normal, inv-gamma)
- Implicit reparameterization [Figurnov et al., 2018]

Example of a chain: $z \sim \mathcal{N}(0,1) \quad t = \mu + \sigma z \quad \theta = e^t$

Target	$p(z;\theta)$	Base $p(\epsilon)$	One-liner $g(\epsilon;\theta)$
Exponential	$\exp(-x); x > 0$	$\epsilon \sim [0;1]$	$\ln(1/\epsilon)$
Cauchy	$\frac{1}{\pi(1+x^2)}$	$\epsilon \sim [0;1]$	$\tan(\pi\epsilon)$
Laplace	$\mathcal{L}(0;1) = \exp(-\lvert x\rvert)$	$\epsilon \sim [0;1]$	$\ln(\frac{\epsilon_1}{\epsilon_2})$
Laplace	$\mathcal{L}(\mu;b)$	$\epsilon \sim [0;1]$	$\mu - bsgn(\epsilon)\ln(1-2\lvert\epsilon\rvert)$
Std Gaussian	$\mathcal{N}(0;1)$	$\epsilon \sim [0;1]$	$\sqrt{\ln(\frac{1}{\epsilon_1})}\cos(2\pi\epsilon_2)$
Gaussian	$\mathcal{N}(\mu;RR^\top)$	$\epsilon \sim \mathcal{N}(0;1)$	$\mu + R\epsilon$
Rademacher	$Rad(\frac{1}{2})$	$\epsilon \sim Bern(\frac{1}{2})$	$2\epsilon - 1$
Log-Normal	$\ln\mathcal{N}(\mu;\sigma)$	$\epsilon \sim \mathcal{N}(\mu;\sigma^2)$	$\exp(\epsilon)$
Inv Gamma	$i\mathcal{G}(k;\theta)$	$\epsilon \sim \mathcal{G}(k;\theta^{-1})$	$\frac{1}{\epsilon}$

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

Score function vs reparameterization

Score function: Every gradient points towards the mode of the approximation, and only the weighting by $\log p(\cdot)$ influences the direction

Reparameterization: Every gradient points towards the mode of the posterior

The latter is clearly better in case $\log p(\mathcal{D}, \boldsymbol{\theta})$ is differentiable and we can reparameterize the approximation, but the former is more general

Often for reparameterization gradients $M = 1$ is enough, whereas score function estimator requires orders of magnitude more even with good control variates

Doubly-stochastic VI

For both estimators the final estimate is usually *doubly stochastic* [Titsias and Lázaro-Gredilla, 2014], since we also subsample the data

For latent variable models $p(\beta, \mathbf{y}, \mathcal{D}) = p(\beta) \prod_{i=1}^{n} p(y_i, x_i|\beta)$ (where y is a latent variable), we would

- Approximate the expectation over $q(\mathbf{z})$ with M samples from the distribution
- Approximate $\log p(\mathcal{D}, \mathbf{y}, \beta)$ using

$$\log p(\beta) + \frac{N}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \log p(x_i, y_i|\beta)$$

for some mini-batch $\mathcal{B}$ of the samples

Automatic variational inference

The above derivations lead to practical algorithms for model-free variational inference

- Assume a differentiable (and usually still factorized) approximation $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$
- Black-box VI: If nothing can be assumed about $\log p(\boldsymbol{\theta}, \boldsymbol{\lambda})$ then use the score fuction estimator

$$\mathbb{E}_{q_{\boldsymbol{\theta}}}\left[(\log p(\mathcal{D}, \boldsymbol{\theta}) - C(\mathcal{D}))\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right]$$

  with suitable control variate, coupled with analytic gradient for the entropy $\mathcal{H}(q(\boldsymbol{\theta}|\boldsymbol{\lambda}))$ [Ranganath et al., 2014]
- Reparameterization VI: If $\log p(\boldsymbol{\theta}, \boldsymbol{\lambda})$ is differentiable and $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be reparameterized, use the reparameterization estimate

$$\mathbb{E}_{\phi}\left[\nabla_{\boldsymbol{\lambda}} \log p(\mathcal{D}, f(\mathbf{z}, \boldsymbol{\lambda}))\right]$$

  and again analytic gradients for the entropy (or Monte Carlo for that as well)

Automatic variational inference

Irrespective of the estimator, the actual learning is done with stochastic gradient ascent, in a similar manner as we would do for deep learning

- Use automatic differentiation for computing the required gradients
- You can safely use mini-batches for evaluating the gradient – it is stochastic anyway due to finite M, so no point in looping over all n
- Update the parameters $\boldsymbol{\lambda}$ according to the gradient, using adaptive step lengths etc

Transformations

The requirement for reparameterization can alternatively be relaxed by transforming the model parameters $\boldsymbol{\theta}$ into $\hat{\boldsymbol{\theta}}$ so that $q(\hat{\theta}_d) = \mathcal{N}(\hat{\theta}_d)$ is a reasonable approximation for every factor – we can always reparameterize it

For example, for $\theta \geq 0$ we can use $\hat{\theta} = \log(e^{\theta} - 1)$ to map to the whole real line and then approximate the posterior with $q(\hat{\theta})$

Due to the transformation we need to add the log-Jacobian of the backwards transformation $\theta = e^{\hat{\theta}} + 1$ when evaluating the gradient, but this can also be computed with automatic differentiation

This is how Stan [Kucukelbir et al., 2017] does VI, but it is not necessarily as good as using more accurate approximations would be – the quality depends on the choice of the transformation

Amortized inference

All the time we have written $q(\theta_d|\boldsymbol{\lambda}_d)$ to indicate that there is a separate independent parameter for each term in a factorized distribution (e.g. one for each latent variable)

Amortized inference relaxes this assumption in search for more scalability. For n latent variables we can alternatively write

$$\boldsymbol{\lambda}_i = f(\mathbf{x}_i, \boldsymbol{\alpha})$$

for some parametric function $f(\cdot)$ that takes as input the observation corresponding to that latent variable

With flexible enough function we may still have free choice of $\boldsymbol{\lambda}_i$, but instead of n independent parameters we only have a fixed-dimensional parameter $\boldsymbol{\alpha}$

For inference we simply use the chain-rule: All of the gradient terms $\nabla_{\boldsymbol{\lambda}}\mathcal{L}(\mathbf{x}_i, \boldsymbol{\lambda})$ are multiplied with $\nabla_{\boldsymbol{\alpha}} f(\mathbf{x}_i, \boldsymbol{\alpha})$

Amortized inference

Variational autoencoders [Kingma and Welling, 2014] are simple latent variable models that use reparameterization gradients and amortized inference

- Model:

$$
\begin{aligned}
z_i &\sim \mathcal{N}(0, I) \\
\epsilon_i &\sim \mathcal{N}(0, \sigma^2) \\
\mathbf{x}_i &= f(\mathbf{z}_i, \eta) + \epsilon_i
\end{aligned}
$$

- Approximation:

$$q(\mathbf{z}_i|\boldsymbol{\lambda}) = \mathcal{N}(g_\mu(\mathbf{x}_i, \boldsymbol{\lambda}_\mu), g_\Sigma(\mathbf{x}_i, \boldsymbol{\lambda}_\Sigma))$$

Here $f(\cdot)$ and $g(\cdot)$ are neural networks that map inputs to the parameters of the approximation

Amortized inference

Some observations:

- VAE is not really an autoencoder: It is standard probabilistic model that conditions the posterior approximation on the inputs, but there is no "encoder" per se
- As a probabilistic program it is a bit simplified; we do Bayesian inference only over the latent variables, not over the network parameters
- The model can have arbitrarily flexible mapping from inputs to the posterior means and variances, but still assumes independent Gaussian approximations

Using variational approximations

In the end we want to compute expectations $\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[f(\boldsymbol{\theta})]$. The naive estimate simply plugs in the approximation in place of the posterior

$$\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[f(\boldsymbol{\theta})] \approx \mathbb{E}_{q(\boldsymbol{\theta})}[f(\boldsymbol{\theta})] \approx \frac{1}{M}\sum_{m=1}^{M} f(\boldsymbol{\theta}_m),$$

but this is possibly severely biased. Nevertheless, this is what people usually do

To reduce the bias we can use importance sampling estimator

$$\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[f(\boldsymbol{\theta})] \approx \frac{\sum_{m=1}^{M} w_m f(\boldsymbol{\theta}_m)}{\sum_{m=1}^{M} w_m}, \text{ for } w_m = \frac{p(\mathcal{D}, \boldsymbol{\theta}_m)}{q(\boldsymbol{\theta}_m|\boldsymbol{\lambda})},$$

but it often has very large variance due to few w_m (sampled from the tails) dominating

A practical alternative is pareto smoothed importance sampling [Yao et al., 2018], which also provides a way of evaluating whether the approximation is accurate enough

Current research

Non-exhaustive examples of current VI research

- Structured approximations: Beyond mean-field by conditioning [Hoffman and Blei, 2015]
- Boosting VI: Improve $q(\boldsymbol{\theta})$ iteratively [Guo et al., 2017, Locatello et al., 2018]
- Other divergences: Replace KL-divergence with other dissimilarity measures; see [Knoblauch et al., 2019] for overview
- Normalizing flows: Beyond VAE by propagating the density (not its parameters) through a neural network [Rezende and Mohamed, 2015]
- Collapsed VI: Integrate out terms analytically [Teh et al., 2007]
- Re-using gradient computations: Update reparameterization gradients without computing $\nabla \log p(\cdot)$ every iteration[Sakaya and Klami, 2017]
- Stein VI: Particle-based inference [Liu and Wang, 2016]
- Partitioned VI: Federated learning meets VI [Bui et al., 2018]

Take home messages

- Variational inference over arbitrary probabilistic programs is possible, but besides model we need to somehow specify the approximation as well
- Can be done with the same basic tools as deep learning, since all we need is automatic differentiation, sampling from standard distributions and SGD
- Consequently easy to merge with DL models as well, but typically we still take point estimates over the network parameters
- But: We do pay a cost – if coordinate ascent updates are available, using them is usually both faster and clearly more reliable, since we do not need to play around with various SGD tuning parameters etc

References I

Thang D. Bui, Cuong V. Nguyen, Siddharth Swaroop, and Richard E. Turner. Paritioned variational inference: A unified framework encompassing federated and continual learning. In *arXiv:1811.11206*, 2018.

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, 76(1), 2017.

Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit Reparameterization Gradients. In *Advances in Neural Information Processing Systems 31*, 2018.

Fangjian Guo, Xiangyu Wang, Kai Fan, Tamara Broderick, and David B Dunson. Boosting Variational Inference. *arXiv preprint arXiv:1611.05559*, 2017.

Matthew Hoffman and David Blei. Stochastic Structured Variational Inference. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, 2015.

D.P. Kingma and M. Welling. Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

References II

Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. Generalized Variational Inference. *arXiv preprint arXiv:1904.02063*, 2019.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic Differentiation Variational Inference. *The Journal of Machine Learning Research*, 18(1), 2017.

Giang Liu and Dilin Wang. Stein variational gradient descent: A general purpose Bayesian inference algorithm. In *Advances in Neural Information Processing Systems*, 2016.

Francesco Locatello, Gideon Dresdner, Rajiv Khanna, Isabel Valera, and Gunnar Raetsch. Boosting Black Box Variational Inference. In *Advances in Neural Information Processing Systems 31*, 2018.

Christian Naesseth, Francisco Ruiz, Scott Linderman, and David Blei. Reparameterization Gradients through Acceptance-Rejection Sampling Algorithms. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, 2014.

Danilo Rezende and Shakir Mohamed. Variational Inference with Normalizing Flow. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

References III

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

Joseph Sakaya and Arto Klami. Importance sampled stochastic optimization for variational inference. In *Proceedings of Uncertainty in Artificial Intelligence*, 2017.

Yee W Teh, David Newman, and Max Welling. A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation. In *Advances in Neural Information Processing Systems 19*, 2007.

Michalis Titsias and Miguel Lázaro-Gredilla. Doubly Stochastic Variational Bayes for non-Conjugate Inference. In *Proceedings of 31st International Conference on Machine Learning*, 2014.

Yuling Yao, Aki Vehtari, Daniel Simpson, and Andrew Gelman. Yes, but did it work?: Evaluating variational inference. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.