Machine Learning Trick of the Day (4): Reparameterisation Tricks

12

29 Oct 2015 | Machine Learning and Statistics
Tags: monte carlo · reinforcement learning · reparameterisation · unbiased estimator · variational inference

Our ability to rewrite statistical problems in an equivalent but different form, to ***reparameterise*** them, is one of the most general-purpose tools we have in mathematical statistics. We used reparameterisation in all the tricks we explored in this series so far: trick 1 re-expressed a log-partition function in terms of copies (replicas) of the marginal probability, trick 2 re-expressed a binary MRF as an undirected model with Gaussian latent variables, and trick 3 re-expressed the computation of the matrix trace using a randomised form. Such reparameterisations allow us to use different tools for computation, give us a new understanding and approach for analysis, and better expose the connections to related research areas.

Today's trick will focus on one subset of reparameterisation methods, ones we shall refer to as ***random variate reparameterisations***: ***the substitution of a random variable by a deterministic transformation of a simpler random variable.*** Machine learning is filled to the brim with random variables, so we'll find many applications of today's trick, including problems in Monte Carlo sampling and stochastic optimisation.

One-liners

To develop the class of random variate reparameterisations, we will first need to develop the tricks that they themselves rely on: the methods by which non-uniform random numbers, or random variates, can be generated. The most popular methods are the ***one-liners***, which give us the simple tools to **generate random variates in one line of code**, following the classic paper by Luc Devroye of the same title [1].

Imagine water flowing through a series of pipes: one-line transformations specify a ***path*** $g(\epsilon;\theta)$ through which samples ϵ from an initial base distribution ***flow***, ultimately forming a sample from a desired distribution $p(z)$. To be a one-liner, the transformation $g(\epsilon;\theta)$ must be composed of primitive functions (arithmetic operations, log, exp, cos) and the base distribution $p(\epsilon)$ must be easy to sample from. The transformations, our system of pipes, can be designed in a number of ways. Three popular approaches are:

- **Inversion methods**. To generate a sample from a distribution $p(z)$, we can use the inverse cumulative distribution function (CDF) as our transformation, if it is known and invertible, with uniform random numbers as the base distribution. Many of the distributions we use every day are in this category, so this is a popular default approach.
- **Polar methods**. Sometimes it is more convenient to think of generating a pair of random variates (x, y) from the target distribution $p(z)$. We can map this pair to a representation in polar form $(r\cos\theta, r\sin\theta)$, which exposes other mechanisms for sampling, e.g., the famous Box-Muller transform is derived in this way.
- **Co-ordinate transformation methods**. Many co-ordinate transformations exist that allow us to transform one distribution into another form, e.g., using additive or multiplicative location-scale transformations. This will be our default approach since co-ordinate transformations allow for easy reparameterisation of a diverse and flexible class of distributions.

Using this reasoning, you can easily see that the transformations in the table below can be implemented in one line of code; many of the sampling algorithms in our software tools, such as Randomkit used in numpy and torch-distributions, use one-liners. Because generating random numbers is so fundamental, we inevitably have many names for this reasoning, including as non-uniform generators [2], as simple instances of normalising flows, and as one-liners (and their extended forms).

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	$\mathcal{L}(\mu;b)$	$\epsilon \sim [0;1]$	$\mu - b\,sgn(\epsilon)\ln(1-2\lvert\epsilon\rvert)$
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	$i\mathcal{G}(k;\theta)$	$\epsilon \sim \mathcal{G}(k;\theta^{-1})$	$\frac{1}{\epsilon}$

The point of describing these one-liners is that when we see a random variable z, we can often explore the implications of using random variate reparameterisation by *replacing z with the function $g(\epsilon;\theta)$*. In Monte Carlo sampling, this approach is sometimes known as a non-centred parameterisation [3], which can lead to more efficient mixing of the Markov chain. Another important application of random variate reparameterisation is in stochastic optimisation, which we explore next.

Reparameterisation for Stochastic Optimisation

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:

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

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:

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

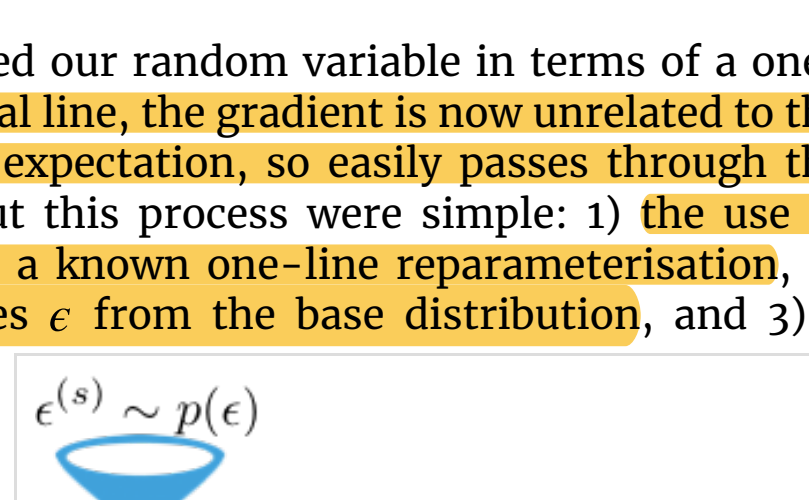
Re-expressing the troublesome stochastic optimisation problem using random variate reparameterisation, we find:

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

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

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

In the second line, we reparameterised our random variable in terms of a one-line generating mechanism. In the final line, the gradient is now unrelated to the distribution with which we take the expectation, so easily passes through the integral. Our assumptions throughout this process were simple: 1) the use of a continuous random variable z with a known one-line reparameterisation, 2) the ability to easily generate samples ϵ from the base distribution, and 3) a differentiable function *f*.



Using reparameterisation has given us a new approach for optimisation: we can obtain an unbiased estimator of the gradient by computing the expectation by Monte Carlo integration. Using the water-pipes analogy, once samples of ϵ are available, they flow through the path specified by the transformation to produce random variates. Gradients flow backwards through this path, allowing for computation by automatic differentiation and composition with other gradient-based systems.

$$\mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g(\epsilon,\theta))] = \frac{1}{S} \sum_{s=1}^{S} \nabla_\theta f(g(\epsilon^{(s)},\theta)), \quad \epsilon^{(s)} \sim p(\epsilon)$$

As with the one-liners, this approach appears under many different names:

- **Perturbation analysis and pathwise derivatives.** In stochastic optimisation, the reparameterised gradient estimator appears under the heading of perturbation analysis [4]. Much of the initial work used a reparamaterisation using one-liners based on the inversion method. The resulting estimators are sometimes called pathwise derivative (PD) estimators [5]. The works by Fu, and Glasserman are comprehensive references using this perspective.
  - Stochastic Gradient Estimation, M. Fu.
  - Gradient estimation via Perturbation analysis, P. Glasserman, Springer, 1991.
- **Stochastic backpropagation.** There are a number of recent uses of random variate reparameterisation in machine learning to develop scalable variational inference in deep generative models and Gaussian processes. At least three papers concurrently explored this approach:
  - Stochastic Backpropagation and Approximate Inference in Deep Generative Models
    - Random variate reparameterisation is one consequence of the technique described as stochastic backpropagation in this paper [6]. There also are other ways to derive unbiased gradient estimators and this paper discusses this aspect and provides some simple variance analysis.
  - Auto-Encoding Variational Bayes
    - We can attribute the popularity of the expression 'reparameterisation trick' to this paper [7], which provides a clear description of the trick and the range of transformations available.
  - Doubly Stochastic Variational Bayes for non-Conjugate Inference
- **Affine-independent inference**.

Since we obtain a Monte Carlo estimator, there are now important questions to ask about the variance of the estimator. Being pathwise gradients (i.e. they implement the chain-rule), these estimators provide an implicit tracking of dependencies between parameters—a provenance tracking—which contributes to the low variance. Gradient estimators based on this approach often have the lowest variance amongst competing approaches. We can also think about ways in which reparameterisation might allow for efficient computation of higher-order gradients. This stochastic optimisation problem can be solved in another very different way and our next trick, the log-derivative trick, will continue this theme and explore these aspects further.

Summary

Random variate reparameterisation is a tool by which we substitute random variables of some known distribution by a deterministic transformation of another random variable. One of the underlying tools that provide us with the deterministic transformations that are needed to achieve this, are the one-liners. Armed with these tricks, we can develop alternative approaches to often-encountered machine learning problems, resulting in faster mixing Markov chains, or scalable Monte Carlo gradient estimators. These methods remain simple and easy to implement, and is what cements their popularity as a tool for developing accurate and scalable machine learning systems.

Source file http://www.shakirm.com/blog-bib/trickOfTheDay/oneliners.bib could not be read.

Some References

12 thoughts on "Machine Learning Trick of the Day (4): Reparameterisation Tricks"

Luke — Apr 2,2019 4:23 pm

Awesome post! Quick note: the one-liner for a standard Gaussian should be sqrt(2 * ln(1/eps_1) * cos(2 * pi * eps_2)

see : https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform

Reply

Pingback: Machine Learning Trick of the Day (8): Instrumental Thinking ← The Spectator

Zijian — Mar 27,2018 8:10 pm

Fabulous introduction on the reparametrization tricks! Will keep learning from you blog and presumably will get tremendous abundance of knowledge. Just a little confusing about the variable transformation part, according to the introduction on the Wiki link you posted, the implication arrow between p(ϵ)|dϵ|=p(z)|dz| and p(z)=|dϵ/dz|p(ϵ) the other term should be perhaps the other way around. Please correct me if i've made mistakes, but following from the measure theory the measure of dz, i.e. p(z), should be 'preserved' under the inverse mapping f^-1: z->ϵ, so that p(ϵ)|dϵ|=p(z)|dz|, then the LHS equation can be implied. Thank you again for your posting and wish i may contribute to the work a little, too.

Reply

Pingback: Machine Learning Trick of the Day (7): Density Ratio Trick | Open Data Science

Pingback: Machine Learning Trick of the Day (7): Density Ratio Trick ← The Spectator

Kaspar — Jan 11,2018 6:24 pm

Thanks a bunch! One question: is the reparametrization trick restricted to transformations where the Jacobian is diagonal? I'm just asking because you took |d eps / dz | apart as if it was.

Reply

Pingback: Some Observations on Relaxed Bernoulli Variables. – Mean Fields and Friendly Skies

David — Jun 12,2017 7:18 pm

GREAT article, thanks!

Reply

Stratis — Nov 29,2016 8:22 am

Hi!

Very nice article, it makes for a nice read and explains some basic concepts. I have a question though. Somewhere you say that the vanilla gradient is hard to compute because

"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;θ)p(z;θ)."

I recognize the first difficulty. Indeed, if your function f() is complicated, like a neural network, computing analytically the integral is hard.

However, what do you mean exactly by the second difficulty? Do you mean that because the variable θ with respect to which you take the gradient is also in the pdf, makes the gradient computation much harder, because the overall composite function becomes harder?

Or because now you need to take the expectation over a multi-dimensional random variable, whereas after the reparameterization trick the expectation is taken over the single dimensional ε? Which implies that you need much fewer samples to compute a good enough expectation, which in turn implies that you can much less variance (or smaller bias) of the estimator?

Reply

Reza — Dec 24,2015 1:07 am

Thanks Shakir for great posts, please keep doing this.

Reply

Pingback: A Year of Approximate Inference: Review of the NIPS 2015 Workshop ← The Spectator

Pingback: Machine Learning Trick of the Day (5): Log Derivative Trick ← The Spectator

Leave a Reply