The Reparameterization Trick

A common explanation for the reparameterization trick with variational autoencoders is that we cannot backpropagate through a stochastic node. I provide a more formal justification.

PUBLISHED
29 April 2018

Informal justifications

In *Auto-Encoding Variational Bayes*, (Kingma & Welling, 2013), Kingma presents an unbiased, differentiable, and scalable estimator for the ELBO in variational inference. A key idea behind this estimator is the *reparameterization trick*. But why do we need this trick in the first place? When first learning about variational autoencoders (VAEs), I tried to find an answer online but found the explanations too informal. Here are a few examples:

> StackExchange: We need the reparameterization trick in order to backpropagate through a random node.

> Reddit: The "trick" part of the reparameterization trick is that you make the randomness an input to your model instead of something that happens "inside" it, which means you never need to differentiate with respect to sampling (which you can't do).

> Quora: The problem is because backpropogation cannot flow through a random node.

I found these unsatisfactory. What does a "random node" mean and what does it mean for backprop to "flow" or not flow through such a node?

The goal of this post is to provide a more formal answer for why we need the reparameterization trick. I assume the reader is familiar with variational inference and variational autoencoders. Otherwise, I recommend (Blei et al., 2017) and (Doersch, 2016) as introductions.

Undifferentiable expectations

Let's say we want to take the gradient w.r.t. θ of the following expectation,

$$\mathbb{E}_{p(z)}[f_\theta(z)]$$

where p is a density. Provided we can differentiate $f_\theta(z)$, we can easily compute the gradient:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p(z)}[f_\theta(z)] &= \nabla_\theta \left[\int_z p(z) f_\theta(z) dz \right] \\ &= \int_z p(z) \left[\nabla_\theta f_\theta(z) \right] dz \\ &= \mathbb{E}_{p(z)} \left[\nabla_\theta f_\theta(z) \right]\end{aligned}$$

In words, the gradient of the expectation is equal to the expectation of the gradient. But what happens if our density p is also parameterized by θ?

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p_\theta(z)}[f_\theta(z)] &= \nabla_\theta \left[\int_z p_\theta(z) f_\theta(z) dz \right] \\ &= \int_z \nabla_\theta \left[p_\theta(z) f_\theta(z) \right] dz \\ &= \int_z f_\theta(z) \nabla_\theta p_\theta(z) dz + \int_z p_\theta(z) \nabla_\theta f_\theta(z) dz \\ &= \underbrace{\int_z f_\theta(z) \nabla_\theta p_\theta(z) dz}_{\text{What about this?}} + \mathbb{E}_{p_\theta(z)} \left[\nabla_\theta f_\theta(z) \right]\end{aligned}$$

The first term of the last equation is not guaranteed to be an expectation. Monte Carlo methods require that we can sample from $p_\theta(z)$, but not that we can take its gradient. This is not a problem if we have an analytic solution to $\nabla_\theta p_\theta(z)$, but this is not true in general.

Now that we have a better understanding of the problem, let's see what happens when we apply the reparameterization trick to our simple example. To be consistent with Kingma, I'll switch to bold text for vectors and denote the ith sample of vector $\mathbf{v}$ as $\mathbf{v}^{(i)}$ and $l \in L$ to denote the lth Monte Carlo sample:

$$\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$$

$$\mathbf{z} = g_\theta(\boldsymbol{\epsilon}, \mathbf{x})$$

$$\mathbb{E}_{p_\theta(\mathbf{z})}[f(\mathbf{z}^{(i)})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_\theta(\boldsymbol{\epsilon}, \mathbf{x}^{(i)}))]$$

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p_\theta(\mathbf{z})}[f(\mathbf{z}^{(i)})] &= \nabla_\theta \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_\theta(\boldsymbol{\epsilon}, \mathbf{x}^{(i)}))] && (1) \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_\theta f(g_\theta(\boldsymbol{\epsilon}, \mathbf{x}^{(i)}))] && (2) \\ &\approx \frac{1}{L} \sum_{l=1}^{L} \nabla_\theta f(g_\theta(\boldsymbol{\epsilon}^{(l)}, \mathbf{x}^{(i)})) && (3)\end{aligned}$$

In my mind, the above line of reasoning is key to understanding VAEs. We use the reparameterization trick to express a gradient of an expectation (1) as an expectation of a gradient (2). Provided g_θ is differentiable—something Kingma emphasizes—then we can then use Monte Carlo methods to estimate $\nabla_\theta \mathbb{E}_{p_\theta(\mathbf{z})}[f(\mathbf{z}^{(i)})]$ (3).

Sanity check

It is worth noting that this explanation aligns with Kingma's own justification:

> **Kingma:** This reparameterization is useful for our case since it can be used to rewrite an expectation w.r.t $q_\phi(\mathbf{z} \mid \mathbf{x})$ such that the Monte Carlo estimate of the expectation is differentiable w.r.t. ϕ.

The issue is not that we cannot backprop through a "random node" in any technical sense. Rather, backpropping would not compute an estimate of the derivative. Without the reparameterization trick, we have no guarantee that sampling large numbers of $\mathbf{z}$ will help converge to the right estimate of ∇_θ.

Furthermore, this is the exact problem we have with the ELBO we want to estimate:

$$\begin{aligned}\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \left[\mathbb{E}_{q_\phi(\mathbf{z})}[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})] \right] && (4) \\ \downarrow & \\ \nabla_{\theta,\phi} \text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \underbrace{\nabla_{\theta,\phi} \left[\mathbb{E}_{q_\phi(\mathbf{z})}[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})] \right]}_{\text{Gradient w.r.t. } \phi \text{ over expectation w.r.t. } \phi}\end{aligned}$$

At this point, you might have noticed that the above equation does not look like what we compute in a standard VAE. In the paper, Kingma actually presents two estimators, which he denotes $\mathcal{L}^A$ and $\mathcal{L}^B$. Equation (4) is $\mathcal{L}^B$, while $\mathcal{L}^B$ is an estimator we can use when we have an analytic solution to the KL-divergence term in the ELBO, for example when we assume both the prior $p_\theta(\mathbf{z})$ and the posterior approximation $q_\phi(\mathbf{z} \mid \mathbf{x})$ are Gaussian:

$$\nabla_{\theta,\phi} \mathcal{L}^B = -\nabla_{\theta,\phi} \overbrace{\left[\text{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)}) \| p_\theta(\mathbf{z})) \right]}^{\text{Analytically compute this}} + \nabla_{\theta,\phi} \overbrace{\left[\frac{1}{L} \sum_{l=1}^{L} \left(\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(l)}) \right) \right]}^{\text{Monte Carlo estimate this}}$$

Now that we can compute the full loss through a sequence of *differentiable operations*, we can use our favorite gradient-based optimization technique to maximize the ELBO.

Implementation

I always find it useful to close the loop and talk about implementation. The equation above is what the standard VAE implements (example) because Kingma derives an analytic solution for the KL term in Appendix 2. A common framing for this version of the model is to think of the likelihood as a "decoder" and the approximate posterior as an "encoder":

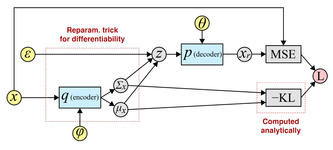
$$\mathcal{L}^B = -\text{KL}[\overbrace{q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})}^{\text{Encoder}} \| \overbrace{p_\theta(\mathbf{z})}^{\text{Fixed}}] + \frac{1}{L} \sum_{l=1}^{L} \overbrace{\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(l)})}^{\text{Decoder}}$$

I think of it like this: the KL-divergence term encourages the approximate posterior to be close to the prior $p_\theta(\mathbf{z})$. If the approximate posterior could exactly match both the real posterior and the prior, then using Bayes' rule we would know that $p(\mathbf{x}) = p(\mathbf{x} \mid \mathbf{z})$. This is exactly what we would want from a generative model. We could sample $\mathbf{z}$ using the reparameterization trick and then condition on $\mathbf{z}$ to generate a realistic sample $\mathbf{x}$.

Concretely, here is one pass through the computational graph when the prior and approximate posterior are Gaussian:

$\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z = M(\mathbf{x}), \Sigma(\mathbf{x})$	Push $\mathbf{x}$ through encoder
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, 1)$	Sample noise
$\mathbf{z} = \boldsymbol{\epsilon}\boldsymbol{\sigma}_z + \boldsymbol{\mu}_z$	Reparameterize
$\mathbf{x}_r = p_\theta(\mathbf{x} \mid \mathbf{z})$	Push $\mathbf{z}$ through decoder
$\text{recon. loss} = \text{MSE}(\mathbf{x}, \mathbf{x}_r)$	Compute reconstruction loss
$\text{var. loss} = -\text{KL}[\mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z) \| \mathcal{N}(0, I)]$	Compute variational loss
$\text{L} = \text{recon. loss} + \text{var. loss}$	Combine losses

Since we computed every variable in this computational graph through a sequence of differentiable operations, we can use a method like backpropagation to compute the required gradients. I find this easiest to see via a diagram:



Above, the input or root nodes are in yellow. I like to denote the network parameters as inputs because it emphasizes how a tool like `autograd` (Maclaurin et al., 2015) actually works: for a given parameter w, we can compute $\partial L / \partial w$ at the graph node that directly takes w as input.

What's in a name?

When I first read Kingma's paper, I wondered why it focused on the *stochastic gradient variational Bayes* (SGVB) estimator and associated algorithm, while the now-famous *variational autoencoder* was just given as an example halfway through the paper.

But with a better understanding of the differentiability of this Monte Carlo estimator, we can understand the focus of the paper and the name of the estimator. *Variational Bayes* refers to approximating integrals using Bayesian inference. The method is *stochastic* because it approximates an expectation with many random samples. And a VAE using neural networks is an example of a model you could build with the SGVB estimator because the estimator is *gradient*-based.

Acknowledgements

I want to thank Bianca Dumitrascu for many helpful conversations on this topic.

1. Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *ArXiv Preprint ArXiv:1312.6114*.
2. Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, *112*(518), 859–877.
3. Doersch, C. (2016). Tutorial on variational autoencoders. *ArXiv Preprint ArXiv:1606.05908*.
4. Maclaurin, D., Duvenaud, D., & Adams, R. P. (2015). Autograd: Effortless gradients in numpy. *ICML 2015 AutoML Workshop*.