3rd Symposium on Advances in Approximate Bayesian Inference, 2020 1–8

Roundoff Error in Metropolis-Hastings Accept-Reject Steps

Matthew D. Hoffman MHOFFMAN@GOOGLE.COM
Google Research

1. Introduction

The Metropolis-Hastings (M-H) accept-reject step (Metropolis et al., 1953; Hastings, 1970) is a central element of many Markov chain Monte Carlo (MCMC) algorithms that are essential to modern Bayesian inference. If performed using exact arithmetic, it can be used to ensure that a Markov chain's stationary distribution is any distribution of interest $p(\theta)$, even if that distribution is only known up to a normalizing constant.

But computers cannot do exact arithmetic on real numbers, and in fact the trend in machine learning is towards lower-precision computation. Lower-precision formats offer both memory savings and more floating-point operations per second (FLOPS) per transistor. Rather than use the traditional 64-bit floating-point format (F64), modern ML software frameworks such as TensorFlow (Abadi et al., 2015), PyTorch (Paszke et al., 2019), and JAX (Bradbury et al., 2018) tend to default to 32-bit floating-point (F32) numerics, or even lower-precision formats such as TensorFloat-32 (TF32; Kharya, 2020) or bfloat16 (Wang and Kanwar, 2019) on supported hardware. These lower-precision formats often work fine for training large neural networks using stochastic gradient descent (e.g., Kalamkar et al., 2019). But are there consequences to using low-precision arithmetic in MCMC?

In this note, we consider what happens to the M-H algorithm when it is fed log-density calculations that are subject to roundoff error and catastrophic cancellations. Below, we will briefly review the nature of these errors and how they can arise in M-H algorithms. Next, we will develop a theoretical model of roundoff error in M-H corrections, and find that it can lead to exponentially low acceptance rates in the magnitude of the errors (if the errors have Gaussian tails) and bias (if different types of states produce different types of errors). Finally, we will discuss some consequences of this phenomenon, and touch on some practical ways to avoid these consequences of catastrophic cancellations in software.

2. A Source of Roundoff Error in Metropolis-Hastings

Metropolis-Hastings updates (M-H; Hastings, 1970) proceed by proposing a move from the current state θ to a new state θ' drawn from some proposal distribution $q(\theta' \mid \theta)$. If we are trying to sample from some distribution $p(\theta)$, then we replace the current state θ with the proposed state θ' with probability $\alpha = \min\{1, \frac{p(\theta')q(\theta|\theta')}{p(\theta)q(\theta'|\theta)}\}$, and otherwise keep the current state. (For both convenience and accuracy, M-H implementations typically compute the densities and their ratio in log space, e.g. $\exp\{\log p(\theta') - \log p(\theta)\}$ instead of $\frac{p(\theta')}{p(\theta)}$.) This procedure leaves the distribution $p(\theta)$ invariant; that is, if $\theta \sim p$ then $\theta' \sim p$ as well. So repeating it with a suitable q for long enough that it forgets its initial state will yield samples from p.

© M.D. Hoffman.

The derivation of the M-H algorithm assumes that we use exact arithmetic to compute the acceptance probability α. Often, floating-point arithmetic is a good enough approximation to arithmetic on real numbers that we can ignore this issue. We have found F32 arithmetic to be problematic in at least two scenarios.

First, if the magnitude of $\log p(\theta') + \log q(\theta \mid \theta')$ is very large, then there may not be enough mantissa bits in an F32 scalar to accurately represent it. For example, the number 10000000.51 cannot be represented as an F32, and gets rounded to 10000001.0. This is a small *relative* error, but in the M-H accept-reject step *absolute* error is what matters. For example, if $\log p(\theta') + \log q(\theta \mid \theta') = 10000000.49$ and $\log p(\theta) + \log q(\theta' \mid \theta) = 10000000.51$, then the true acceptance probability should be $e^{-0.02} \approx 0.98$, but if these values are rounded before subtraction the computed acceptance probability is $e^{-1} \approx 0.37$.

Second, when $\log p(\theta)$ is a sum of many terms, small errors in summing these terms can accumulate. Figure 1 illustrates this phenomenon when computing two million independent Poisson log-likelihoods with different rate parameters. Summing the terms for each observation in F32 yields roughly Gaussian errors about 10 times larger than the nearly uniform errors obtained when summing in F64 and rounding the final result to F32. The errors due to summing in F32 are clearly large enough to affect the dynamics of the M-H algorithm, causing some states that might be rejected to be accepted and vice versa. In the next section, we will explore how these errors may introduce bias or degrade the efficiency of the M-H algorithm.

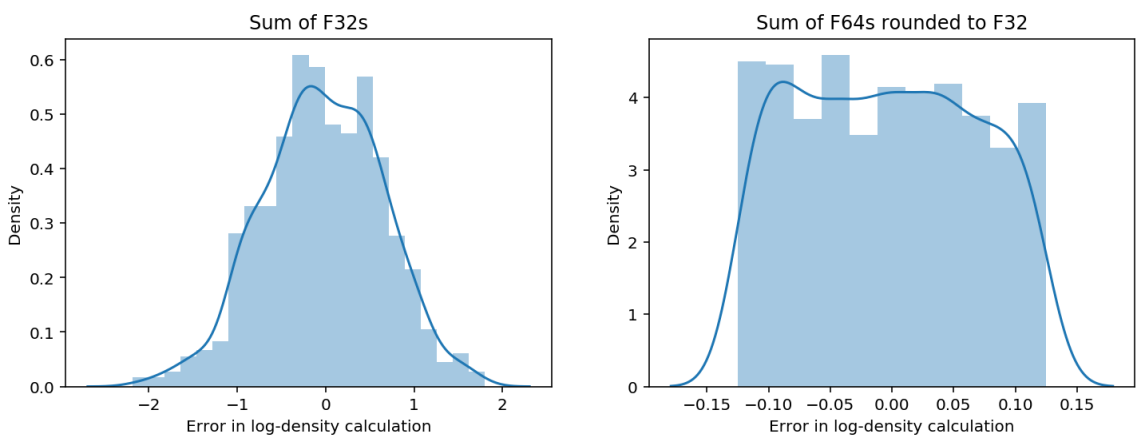

Figure 1: Errors when computing the sum of the log-likelihoods of two million Poisson random variables. NumPy code to generate the plots is in Appendix A.

3. Analysis of Metropolis-Hastings with Roundoff Error

In this section, we analyze the effects of roundoff error in the log-densities fed to the Metropolis-Hastings algorithm. We denote the state of the Markov chain as θ, the exact target log-density as $\log p(\theta)$, and the computed target log-density with roundoff error ϵ

as $\log \hat{p}(\theta) = \log p(\theta) + \epsilon(\theta)$. Since $\epsilon(\theta)$ is obtained by rounding bounded floating-point numbers, it is also bounded, so we can define the normalizing constant $Z = \int_\theta \hat{p}(\theta) = \mathbb{E}_p[e^{\epsilon(\theta)}]$. Note that, although ϵ is a deterministic function of θ, when θ is a draw from a Markov chain ϵ is also a random variable whose distribution depends on that of θ.

At each step, the algorithm proceeds from a state θ and cached $\log \hat{p}(\theta)$ to sample $\theta' \sim q(\cdot \mid \theta)$, computes the perturbed Metropolis-Hastings acceptance probability

$$\hat{\alpha} = \min\{1, \tfrac{\hat{p}(\theta')q(\theta|\theta')}{\hat{p}(\theta)q(\theta'|\theta)}\} = \min\{1, \tfrac{p(\theta')q(\theta|\theta')}{p(\theta)q(\theta'|\theta)} e^{\epsilon(\theta')-\epsilon(\theta)}\}, \tag{1}$$

and accepts the new state θ' with probability $\hat{\alpha}$.

This algorithm leaves the perturbed distribution $\frac{\hat{p}}{Z}$ invariant. If expectations of interest with respect to $\frac{\hat{p}}{Z}$ are close to their values under the true p, then the asymptotic bias of this algorithm will be small. Loosely speaking, this bias will be small if we are trying to estimate the expected value of some function $h(\theta)$ such that the average of $e^{\epsilon(\theta)}$ over any region in which $h(\theta)$ is nearly constant is approximately Z.

But if the variance of $\epsilon(\theta)$ is large, then the average acceptance rate $\hat{\alpha}$ may get very small, dramatically slowing down the algorithm. Intuitively, the issue is that $e^{\epsilon(\theta)}$ may have quite heavy tails, and so most of the mass in $\hat{p}$ will be concentrated on values of θ for which $e^{\epsilon(\theta)}$ are abnormally large. Unfortunately, there is no way to preferentially select states whose roundoff errors are large and positive, so the chain must wait until it gets a favorable error by chance. (Pseudo-marginal MCMC suffers from a very similar issue (Andrieu and Roberts, 2009).)

We can make this intuition a bit more quantitative. At stationarity, the expected acceptance probability is

$$\begin{aligned}\mathbb{E}_{\hat{p}}[\hat{\alpha}] &= \int_{\theta,\theta'} \frac{p(\theta)}{Z} e^{\epsilon(\theta)} q(\theta' \mid \theta) \min\left\{1, \frac{p(\theta')q(\theta \mid \theta')}{p(\theta)q(\theta' \mid \theta)} e^{\epsilon(\theta')-\epsilon(\theta)}\right\} d\theta d\theta' \\ &= \int_{\theta,\theta'} p(\theta)q(\theta' \mid \theta) \min\left\{\frac{e^{\epsilon(\theta)}}{\mathbb{E}_p[e^{\epsilon(\theta)}]}, \frac{p(\theta')q(\theta \mid \theta')}{p(\theta)q(\theta' \mid \theta)} \frac{e^{\epsilon(\theta')}}{\mathbb{E}_p[e^{\epsilon(\theta)}]}\right\} d\theta d\theta'.\end{aligned} \tag{2}$$

Consider what happens if we have an ideal proposal distribution $q(\theta' \mid \theta) = p(\theta')$. Without roundoff, this would yield perfect samples and an acceptance rate of 1. But with roundoff, we can bound the acceptance rate as

$$\hat{\alpha}^\star \triangleq \int_{\theta,\theta'} p(\theta)p(\theta') \min\left\{\frac{e^{\epsilon(\theta)}}{\mathbb{E}_p[e^{\epsilon(\theta)}]}, \frac{e^{\epsilon(\theta')}}{\mathbb{E}_p[e^{\epsilon(\theta)}]}\right\} d\theta d\theta' \tag{3}$$

$$= \mathbb{E}_p[e^{\epsilon(\theta)}]^{-1} \int_0^\infty p(\min\{e^{\epsilon(\theta)}, e^{\epsilon(\theta')}\} = m) m dm \tag{4}$$

$$= \mathbb{E}_p[e^{\epsilon(\theta)}]^{-1} \int_0^\infty 2p(e^{\epsilon(\theta)} = m) P(e^{\epsilon(\theta)} > m) m dm \tag{5}$$

$$\leq \mathbb{E}_p[e^{\epsilon(\theta)}]^{-1} \int_0^\infty 2p(e^{\epsilon(\theta)} = m) \frac{\mathbb{E}_p[e^{\epsilon(\theta)/2}]}{\sqrt{m}} m dm \tag{6}$$

$$= \frac{\mathbb{E}_p[e^{\epsilon(\theta)/2}]^2}{\mathbb{E}_p[e^{\epsilon(\theta)}]}, \tag{7}$$

where the bound follows from applying Markov's inequality to $e^{\epsilon(\theta)/2}$.

By the strict convexity of the square, this upper bound is strictly less than 1 unless the variance of ϵ is 0. How much less depends on both the scale and the tails of the distribution of ϵ. Two cases are of particular interest: first, where ϵ follows a normal distribution (e.g., due to summing many independent roundoff errors), and second, where ϵ follows a uniform distribution (e.g., due to rounding an accurately computed log-density to fit into a single-precision float). In both of these cases we can compute the exact acceptance rate $\hat{\alpha}^\star$.

The Gaussian case: If $\epsilon \sim \mathcal{N}(\mu, \sigma)$, then equation 5 can be written as

$$\hat{\alpha}^\star = e^{-\mu-\sigma^2/2} \int_\epsilon 2\phi(\epsilon)\Phi(-\epsilon)e^{\sigma(\epsilon+\mu)}d\epsilon = e^{-\sigma^2/2} \int_\epsilon 2\phi(\epsilon)\Phi(-\epsilon)e^{\sigma\epsilon}d\epsilon, \tag{8}$$

where $\phi(\epsilon)$ and $\Phi(\epsilon)$ are the pdf and cdf of a standard normal distribution. Since $2\phi(x)\Phi(-x)$ is the pdf of a skew-normal distribution with shape parameter $\alpha = -1$, we can use the skew-normal's moment-generating function M to compute the integral:

$$\hat{\alpha}^\star = e^{-\sigma^2/2}\mathrm{M}(\sigma; -1) = e^{-\sigma^2/2}2e^{\sigma^2/2}\Phi(-\sigma/\sqrt{2}) = 2\Phi(-\sigma/\sqrt{2}). \tag{9}$$

If σ is close to zero, then a first-order Taylor approximation implies that the acceptance rate will be

$$\hat{\alpha}^\star = 2\Phi(-\sigma/\sqrt{2}) = 1 - \frac{1}{\sqrt{\pi}}\sigma + O(\sigma^3), \tag{10}$$

so small amounts of roundoff error will not hurt the acceptance rate too badly. However, as σ gets large, the acceptance rate approaches zero very rapidly, as Figure 2 shows. For example, if $\sigma = 2$ then $\hat{\alpha}^\star \approx 0.16$; if $\sigma = 4$ then $\hat{\alpha}^\star \approx 0.005$.

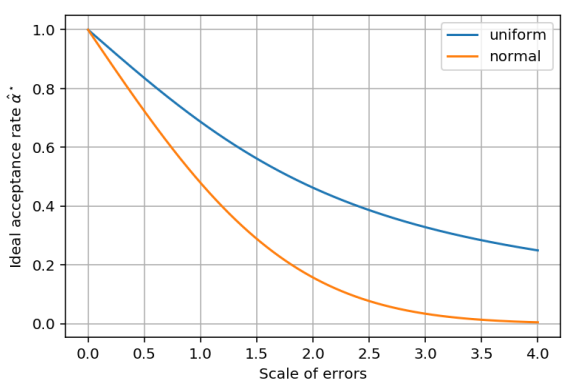

Figure 2: Average acceptance probability of the ideal M-H proposal $q(\theta' \mid \theta) = p(\theta')$ assuming the log-density is corrupted by uniformly or normally distributed roundoff error as in Equation (11) and Equation (9).

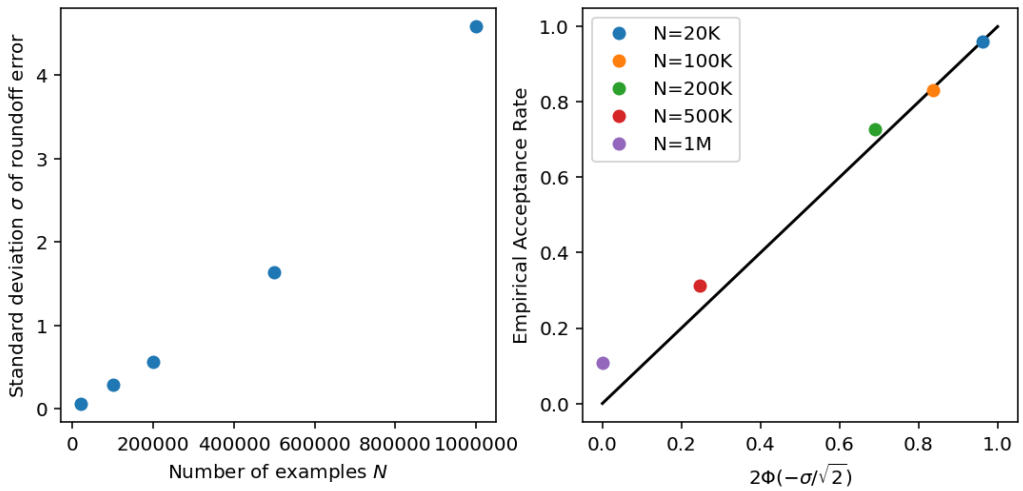

Figure 3: Left: Standard deviation of the error ϵ of the F32 log-posterior calculation as a function of number of examples. Right: Predicted (x-axis) and empirical (y-axis) acceptance rates of HMC chains with different numbers of examples N (and therefore different amounts of roundoff error).

The uniform case: If $\epsilon \sim \text{Uniform}([\mu - \sigma, \mu + \sigma])$, then equation 5 can be written as

$$\begin{aligned} \hat{\alpha}^\star &= \left(\frac{1}{2\sigma} \int_{-\sigma}^{\sigma} e^{\mu+\epsilon} d\epsilon \right)^{-1} \frac{1}{2\sigma} \int_{-\sigma}^{\sigma} \left(1 - \frac{\epsilon}{\sigma}\right) e^{\mu+\epsilon} d\epsilon \\ &= \frac{1}{\sigma} + 1 - \frac{1}{\tanh(\sigma)}. \end{aligned} \tag{11}$$

For small σ, this is $1 - \frac{\sigma}{3} + O(\sigma^3)$. For large σ, the tanh saturates and it becomes $\frac{1}{\sigma}$. This is a much more graceful degradation in acceptance rate than the Gaussian case, but for large roundoff errors it may still be quite bad.

In summary, we have argued that roundoff errors in computing the log-density that is used in the Metropolis-Hastings accept-reject step may or may not induce bias, but they can aggressively degrade the acceptance rate to the point that the algorithm becomes unusable.

4. Empirical Example: Bayesian Linear Regression

In this section, we will empirically demonstrate the issues described above by trying to use Hamiltonian Monte Carlo (HMC; **?**) to sample from the posterior of a simple Bayesian linear regression model applied to synthetic data. The model is

$$\beta \sim \mathcal{N}(0, I); \quad y_n \sim \mathcal{N}(x_n^\top \beta, 0.1), \tag{12}$$

where $\beta \in \mathbb{R}^2$, $x \in \mathbb{R}^{N \times 2}$, and $y \in \mathbb{R}^N$. To generate the dataset, we drew each x_{nd} i.i.d. from a standard normal and sampled β and y from the generative process above. We ran

HMC in both F32 and F64 on a CPU using TensorFlow Probability (**?**) in JAX (Bradbury et al., 2018) to sample from $p(\beta \mid x, y)$ for $N \in \{20000, 100000, 200000, 500000, 1000000\}$. All chains were run for 500 iterations with 20 leapfrog steps and a step size of $0.005/\sqrt{N}$. Each chain was initialized with the value of β that was used to generate y, which is a valid draw from the posterior $p(\beta \mid x, y)$. When run in F64, the average acceptance probabilities for these chains were very high: between 0.99975 and 0.99977 for all values of N.

Figure 3 summarizes the results of the F32 experiments. as N gets larger, the number of terms being added together to compute the posterior log-density gets larger, and the number of roundoff errors being added together likewise grows. As the magnitude of the roundoff errors grows, the average acceptance probability drops roughly as equation 9 predicts it should if the roundoff errors are Gaussian, although it is a bit higher than predicted for $N = 1000000$. This may be because the chain has not actually converged to its stationary distribution; it might take many tries before it randomly selects a β with a large enough positive roundoff error $\epsilon(\beta)$ to qualify as "typical" under the perturbed stationary distribution $\hat{p}(\beta)$.

5. Discussion and Practical Considerations

We have illustrated a way in which roundoff error can severely degrade the performance of Metropolis-Hastings procedures, leading in some cases to bias and in others to such low acceptance rates that the method is useless. Unfortunately these issues seem most pronounced in precisely the sort of large-scale problems that motivate the use of cheap, low-precision arithmetic.

We have found a few of ways of diagnosing these roundoff issues. The simplest is of course to run the algorithm in F64 and see if anything changes, but not all implementations make that easy. One can also inspect the values of $\log p(\theta)$ that the algorithm is generating; if they have more than six digits to the left of the decimal point, then there may be cause for concern. Finally, most Metropolis-Hastings algorithms have some kind of step size parameter; in an exact-arithmetic implementation, as the step size approaches zero, the acceptance rate would go to one. If it is difficult or impossible to drive the acceptance rate to one by reducing the step size, that is a warning sign that numerical issues may be present. Likewise, if one is employing a step-size adaptation scheme that targets a particular acceptance rate, and instead of converging it drives the step size to zero, then one should suspect roundoff error as the culprit.

There are multiple ways to address roundoff issues. A simple approach is to compute the terms being summed in $\log p(\theta)$ in F32, but convert them to F64 before the final (relatively cheap) summation. If F64 summation is not available or convenient, one can also use methods like Kahan summation to eliminate catastrophic cancellations. In many cases it is more accurate to compute a sum of differences than a difference of sums, e.g., $\sum_n \log p(x_n \mid \theta') - \log p(x_n \mid \theta)$ instead of $(\sum_n \log p(x_n \mid \theta')) - (\sum_n \log p(x_n \mid \theta))$; however, this approach makes caching of previous results more memory-intensive and is somewhat cumbersome, since it breaks the abstraction of a scalar log-density. Also, hardware that supports stochastic rounding may dramatically reduce roundoff error (Gupta et al., 2015).

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Christophe Andrieu and Gareth O Roberts. The pseudo-marginal approach for efficient monte carlo computations. *Ann. Stat.*, 37(2):697–725, April 2009.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision, 2015.

W. K. Hastings. Monte Carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. doi: 10.1093/biomet/57.1.97. URL http://dx.doi.org/10.1093/biomet/57.1.97.

Dhiraj D. Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. A study of BFLOAT16 for deep learning training. *CoRR*, abs/1905.12322, 2019. URL http://arxiv.org/abs/1905.12322.

Paresh Kharya. Tensorfloat-32 in the a100 gpu accelerates ai training, hpc up to 20x, May 2020. URL https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In

H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Shibo Wang and Pankaj Kanwar. Bfloat16: the secret to high performance on cloud tpus. *Google Cloud Blog*, 2019.

Appendix A. Code to generate figure 1

```
def poisson_logpdf(x, rate):
  return (-scipy.special.gammaln(x+1) + x * np.log(rate) - rate).sum()
x = np.random.poisson(1, size=2000000)
sum_errors = []
final_roundoff_errors = []
for i in range(1000):
  rate = np.exp(0.2 * np.random.randn())
  logpdf64 = poisson_logpdf(x, rate).sum()
  logpdf32 = poisson_logpdf(x.astype(np.float32), rate.astype(np.float32)).sum()
  sum_errors.append(logpdf32 - logpdf64)
  final_roundoff_errors.append(logpdf64.astype(np.float32) - logpdf64)

plt.figure(figsize=[12, 4])
plt.subplot(1, 2, 1)
plt.title('Sum of F32s')
sns.distplot(sum_errors)
plt.xlabel('Error in log-density calculation')
plt.subplot(1, 2, 2)
plt.title('Sum of F64s rounded to F32')
sns.distplot(final_roundoff_errors)
plt.xlabel('Error in log-density calculation')
sns.distplot(errors)
```