Gauss Quadrature in Machine Learning

1st Chen Yihang
Peking University
1700010780

***Abstract*—In this report, we discuss the potential applications of Gauss quadrature in modern machine learning. In the begining, we combine the quadrature rule with Lanczos algorithm to efficiently calculate bilinear form $v^\top f(A)v$. In sequel, we use the idea to estimate the Hessian eigenvalue distribution in large-scale networks. In another regime, we discuss the relation between kernel approximation and numerical quadrature.**

I. Bilinear Forms with Matrix Functions

A. Settings

We want to develop methods for approximating expressions of the form [1]

$$u^\top f(A)v \tag{I.1}$$

where $u, v \in \mathbb{R}^n$ and $A \in S^n_{++}$. It suffices to consider $u = v$ thanks to the identity

$$u^\top f(A)v = \frac{1}{4}(u+v)f(A)(u+v) - \frac{1}{4}(u-v)^\top f(A)(u-v)$$

B. Gauss Quadrature

Since A is positive definite, assuming $\lambda_1 \geq \cdots \geq \lambda_n$ is the eigenvalue, we can decompose A into $A = \sum_{j=1}^n \lambda_j q_j q_j^\top$

It follows that we can write this bilinear form as a Riemann-Stieltjes integral

$$v^\top f(A)v = \int_{\lambda_n}^{\lambda_1} f(\lambda)\mathrm{d}\alpha(\lambda) \tag{I.2}$$

where

$$\alpha(\lambda) = \begin{cases} 0 & \lambda < \lambda_n \\ \sum_{j=1}^k (q_j^\top v)^2, & \lambda_k \leq \lambda < \lambda_{k+1} \\ \sum_{j=1}^n (q_j^\top v)^2, & \lambda \geq \lambda_1 \end{cases} \tag{I.3}$$

We wish to compute nodes t_j and weights $w_j, j = 1, \cdots, n$, such that

$$v^\top f(A)v = \int_{\lambda_n}^{\lambda_1} f(\lambda)\mathrm{d}\alpha(\lambda) \approx \sum_{j=1}^n f(t_j)w_j \tag{I.4}$$

C. Orthogonal Polynomials

Define $\langle f, g\rangle = \int_{\lambda_n}^{\lambda_1} f(\lambda)g(\lambda)\mathrm{d}\alpha(\lambda) = v^\top f(A)g(A)v$, we want to obtain polynomial q_j of degree j, such that $\langle q_i, q_j\rangle = \delta_{ij}$.

We define n-vector $Q_n(x)$ by

$$Q_n(x) = (q_0(x), q_1(x), \cdots, q_{n-1}(x))^\top \tag{I.5}$$

By Gram-Schmidt orthogonalization, we have

$$p_j(x) = (x - \alpha_j)q_{j-1}(x) - \beta_{j-1}q_{j-2}(x) \tag{I.6}$$

where $\alpha_j = \langle xq_{j-1}, q_{j-1}\rangle, \beta_j = \langle xq_{j-1}, q_j\rangle, j \geq 1$. Then we normalize p_j and get q_j. By a simple calculation, $\beta_j = \sqrt{\langle p_j, p_j\rangle}$. The iteration start with $p_0(x) = 1$.

Hence

$$xq_{k-1}(x) = \beta_k(x)q_k(x) + \alpha_k q_{k-1}(x) + \beta_{k-1}q_{k-2}(x) \tag{I.7}$$

we get that if t_j is a root of $q_n(x)$, then $t_j Q_n(t_j) = J_n Q_n(t_j)$, where

$$J_n = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{pmatrix} \tag{I.8}$$

Hence, t_j is an eigenvalue J_n. We can prove that they are in $[\lambda_n, \lambda_1]$. They are guaranteed to be well-conditioned, and they can be computed quite efficiently using the symmetric QR algorithm.

If we define $x_j = q_{j-1}(A)v, r_j = p_j(A)v, j \geq 1$. Then $\alpha_j = x_j^\top A x_j$, $\beta_j = \|r_j\|_2$. Acording to relation I.6, we can write Lanczos algorithms in Algorithm 1. For more detailed analysis, please refer to [2] and [3].

Algorithm 1 Lanczos algorithm, update J_n.

```
r_0 ← v, x_0 ← 0
for j = 1, 2, ··· , n do
    β_{j-1} = ‖r_{j-1}‖_2
    x_j = r_{j-1}/β_{j-1}
    α_j = x_j^⊤ A x_j
    r_j = (A − α_j I)x_j − β_{j-1} x_{j-1}
end for
```

II. Hessian eigenvalues estimation

Empirical analysis of the Hessian has been of significance interest in the deep learning community. Due to computational costs of computing the exact eigenvalues ($\mathcal{O}(n^3)$ for an explicit $n \times n$ matrix), most of the papers in this line of research either focus on smaller models or on low-dimensional projections of the loss surface. Ghorban et al. [4] provides an accurate and scalable estimation of Hessian eigenvalue densities.

The Hessian, $\nabla^2\mathcal{L}(\theta) \in \mathbb{R}^{n\times n}$ is a symmetric matrix such that $\nabla^2\mathcal{L}(\theta)_{i,j} = \frac{\partial^2}{\partial\theta_i\partial\theta_j}\mathcal{L}(\theta)$. Note that our Hessians

are all "full-batch" Hessians (i.e., they are computed using the entire dataset). We represent $\nabla^2\mathcal{L}(\hat{\theta}_t)$ with $H \in \mathbb{R}^{n\times n}$. Throughout the section, H has the spectral decomposition $Q\Lambda Q^T$ where $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$, $Q = [q_1, \ldots, q_n]$ and $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_n$.

To understand the Hessian, we would like to compute the eigenvalue (or spectral) density, defined as $\phi(t) = \frac{1}{n}\sum_{i=1}^{n}\delta(t-\lambda_i)$ where δ is the Dirac delta operator. The naive approach requires calculating λ_i; however, when the number of parameters, n, is large this is not tractable. We relax the problem by convolving with a Gaussian density of variance σ^2 to obtain:

$$\phi_\sigma(t) = \frac{1}{n}\sum_{i=1}^{n} f(\lambda_i; t, \sigma^2) \tag{II.1}$$

where $f(\lambda; t, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\exp -\frac{(t-\lambda)^2}{2\sigma^2}$. For small enough σ^2, $\phi_\sigma(t)$ provides all practically relevant information regarding the eigenvalues of H. Explicit representation of the Hessian matrix is infeasible when n is large, but using Pearlmutter's trick [5] we are able to compute Hessian-vector products for any chosen vector. Thus, we can efficiently implement Lanczos algorithm.

A. Stochastic Lanczos Quadrature

Since H is diagonalizable and f is analytic, we can define $f(H) = Qf(\Lambda)Q^T$ where $f(\cdot)$ acts point-wise on the diagonal of Λ. Now observe that if $v \sim N(0, \frac{1}{n}I_{n\times n})$, we have

$$\phi_\sigma(t) = \frac{1}{n}\mathrm{tr}\left(f(H, t, \sigma^2)\right) = \mathbb{E}\left[v^T f(H, t, \sigma^2)v\right] \tag{II.2}$$

Thus, as long as $\phi_\sigma^{(v)}(t) \equiv v^T f(H, t, \sigma^2)v$ concentrates fast enough, to estimate $\phi_\sigma(t)$, it suffices to sample a small number of random v's and average $\phi_\sigma^{(v)}(t)$.

By definition, we can write

$$\phi_\sigma^{(v)}(t) = v^T Q f(\Lambda; t, \sigma^2)Q^T v = \sum_{i=1}^{n}\beta_i^2 f(\lambda_i; t, \sigma^2) \tag{II.3}$$

where $\beta_i := (v^T q_i)$. Instead of summing over the discrete index variable i, we can rewrite this as a Riemann-Stieltjes integral over a continuous variable λ weighted by μ (as I.3):

$$\phi_\sigma^{(v)}(t) = \int_{\lambda_n}^{\lambda_1} f(\lambda; t, \sigma^2)d\mu(\lambda) \tag{II.4}$$

To evaluate this integral, we apply a quadrature rule. In particular, we want to pick a set of weights ω_i and a set of nodes l_i so that

$$\phi_\sigma^{(v)}(t) \approx \sum_{i=1}^{m}\omega_i f(\ell_i; t, \sigma^2) \equiv \widehat{\phi}^{(v)}(t) \tag{II.5}$$

The hope is that there exists a good choice of $(\omega_i, \ell_i)_{i=1}^m$ where $m \ll n$ such that $\phi_\sigma^{(v)}(t)$ and $\widehat{\phi}^{(v)}(t)$ are close for all t, and that we can find the nodes and weights efficiently for our particular integrand f and the CDF μ.

We use Lanczos algorithm (with full re-orthogonalization) to perform this computation in a numerically stable manner. We summerize our algorithm in Algorithm 2.

Algorithm 2 Two Stage Estimation of $\phi_\sigma(t)$

Draw k i.i.d realizations of v, $\{v_1, \ldots, v_k\}$.

I. Estimate $\phi_\sigma^{(v_i)}(t)$ by a quantity $\widehat{\phi}^{(v_i)}(t)$:
- Run the Lanczos algorithm for m steps on matrix H starting from v_i to obtain tridiagonal matrix T.
- Compute eigenvalue decomposition $T = ULU^T$.
- Set the nodes $\ell_i = (L_{ii})_{i=1}^m$ and weights $\omega_i = (U_{1,i}^2)_{i=1}^m$.
- Output $\widehat{\phi}^{(v_i)}(t) = \sum_{i=1}^m \omega_i f(\ell_i; t, \sigma^2)$.

II. Set $\widehat{\phi}_\sigma(t) = \frac{1}{k}\sum_{i=1}^k \widehat{\phi}^{(v_i)}(t)$.

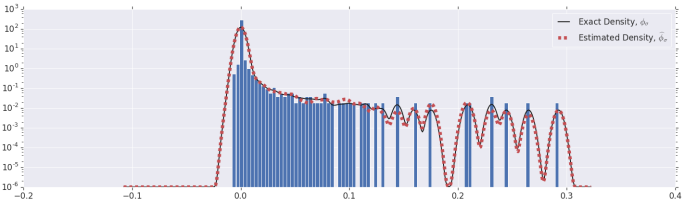

Figure II.1. Comparison of the estimated smoothed density (dashed) and the exact smoothed density (solid) in the interval $[-0.2, 0.4]$. We use $\sigma^2 = 10^{-5}, k = 10$ and degree 90 quadrature. For completeness, the histogram of the exact eigenvalues is also plotted.

B. Concentration Inequality

Theorem 1. Let t be a fixed evaluation point and k be the number of realizations of v in step II. Let $a = \|f(H; t, \sigma^2)\|_F$ and $b = \|f(H; t, \sigma^2)\|_2$. Then for any $x > 0$,

$$P\left(|\phi_\sigma(t) - \widehat{\phi}_\sigma(t)| > \frac{2a}{n\sqrt{k}}\sqrt{x} + \frac{2b}{kn}x\right) \leq 2\exp(-x).$$

Alternatively, since $f(\cdot)$ is a Gaussian density, we can give norm independent bounds: $\forall x > 0$,

$$P\left(|\phi_\sigma(t) - \widehat{\phi}_\sigma(t)| > \epsilon(x)\right) \leq 2\exp(-x). \tag{II.6}$$

where $\epsilon(x) \equiv \sqrt{\frac{2}{\pi\sigma^2}}(\sqrt{\frac{x}{nk}} + \frac{x}{nk})$.

III. Gauss Quadrature for Kernel Features

A kernel machine is one that handles input $x_1, \cdots, x_n$, represented as vectors in $\mathbb{R}^d$, only in terms of some kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ of pairs of data points $k(x_i, x_j)$. One well-known downside of kernel machines is the fact that they scale poorly to large datasets. We instead construct a feature map $z : \mathbb{R}^d \to \mathbb{R}^D$ such that $k(x, y) \approx \langle z(x); z(y)\rangle$. This approximation enables kernel machines to use scalable linear methods for solving classification.

A. Shift-invariant kernels

In the case of shift-invariant kernels, one technique that was proposed for constructing the function z is random Fourier features.

Based on Bochner's theorem [6], we can write k in terms of its Fourier transform Λ

$$\begin{aligned} k(x,y) = k(x-y) &= \int_{\mathbb{R}^d} \Lambda(\omega) \exp(i\omega\top(x-y))\mathrm{d}\omega \\ &= \mathbb{E}_{\omega\sim\Lambda}\left\langle \exp(i\omega^\top x), \exp(i\omega^\top y)\right\rangle \end{aligned} \tag{III.1}$$

Our objective is to choose ω_j and a_j to approximate III.1 by $\tilde{k}(x-y) = \sum_{j=1}^{D} a_j \exp(i\omega_j\top(x-y))$. The feature map z can be readily constructed by $z(x) = (\sqrt{a_j}\exp(i\omega_j^\top x))_{j=1}^D$.

This data-independent method approximates the Fourier transform integral of the kernel by averaging Monte-Carlo samples, which allows for arbitrarily-good estimates of the kernel function k. However, Dao et al. [7] propose a deterministic method to approximate the kernel's Fourier transform integral III.1 with a discrete sum.

1) Dense grid quadrature: The simplest way to do this is to factor the integral III.1 into $\prod_{j=1}^d \left(\int_{-\infty}^{+\infty} \Lambda_j(\omega)\exp(i\omega e_j^\top(x-y))\mathrm{d}\omega\right)$. We can then approximate them all with a one-dimensional quadrature rule. Unfortunately, this scheme suffers heavily from the curse of dimensionality, thus being futile in real problems.

2) Sparse grid quadrature: We start by letting let $G_j^L(u_j)$ be the approximation of $k_j(u_j)$ that results from applying the one-dimensional Gaussian quadrature rule with L points: for the appropriate sample points and weights,

$$G_j^L(u_j) = \sum_{l=1}^{L} a_l \exp(iu_j\omega_l).$$

One of the properties of Gaussian quadrature is that it is exact in the limit of large L. In particular, this limit means that we can decompose $k_j(u_j)$ as the infinite sum

$$\begin{aligned} k_j(u_j) &= G_j^1(u_j) + \sum_{m=1}^{\infty}\left(G_j^{2^m}(u_j) - G_j^{2^{m-1}}(u_j)\right) \\ &= \sum_{m=0}^{\infty} \Delta_{i,m}(u_j) \end{aligned}$$

where $\Delta_{j,m}(u_j) = G_j^{2^m}(u_j) - G_j^{2^{m-1}}(u_j)$. To represent $k(u)$, it suffices to use the product

$$k(u) = \sum_{\mathbf{m}\in\mathbb{N}^d}\prod_{j=1}^{d}\Delta_{j,m_j}(u_j) = \sum_{\mathbf{m}\in\mathbb{N}^d}\Delta_{\mathbf{m}}(u)$$

where $\Delta_{\mathbf{m}}(u) = \prod_{j=1}^d \Delta_{j,m_j}(u_j)$. We can think of these $\Delta_{\mathbf{m}}$ forming a "grid" of terms in $\mathbb{N}^d$.

Smolyak's sparse grid approximation [8] approximates this sum by using only those $\Delta_{\mathbf{m}}$ that can be computed with a "small" number of samples. Specifically, the sparse grid up to level A is defined as,

$$\tilde{k}(u) = \sum_{\mathbf{m}\in\mathbb{N}^d,\, \mathbf{1}^\top\mathbf{m}\le A} \Delta_{\mathbf{m}}(u).$$

Now, for any u, each $\Delta_{\mathbf{m}}(u)$, the number of samples required is no greater than $3^{\mathbf{1}^\top\mathbf{m}}$. Combining this with the previous equation gives us a rough upper bound on the sample count of the sparse grid construction

$$D \le \sum_{\mathbf{m}\in\mathbb{N}^d,\, \mathbf{1}^\top\mathbf{m}\le A} 3^{\mathbf{1}^\top\mathbf{m}} \le 3^A\binom{d+A}{A}.$$

Hence the simplex of terms used by the sparse grid contains exponentially (in d) fewer quadrature points than the hypercube of terms used by a dense grid.

3) Reweighted grid quadrature: A data-dependent method is proposed based on the dense grid quadrature. Since k is real-valued, $\tilde{k}(x-y) = \sum_{j=1}^D a_j\cos(\omega_j\top(x-y))$. We first choose the set of potential grid points w_j by sampling from a dense grid of Gaussian quadrature points. To solve for the weights a_j, we independently sample n pairs (x_k, y_k) from the dataset, then minimize the empirical mean squared error. In order to make solution sparse, we add an ℓ_1-penalty term.

$$\min \quad \frac{1}{n}\sum_{k=1}^{n}(k-\tilde{k})^2(x_k,y_k) + \lambda\|a\|_1 \tag{III.2a}$$

$$\text{s.t.} \quad a_j \ge 0,\ i = 1,\cdots,D \tag{III.2b}$$

Solving III.2 amounts to solve a non-negative least-squares problem with ℓ_1 penalty. [9]

REFERENCES

[1] G. H. Golub and G. Meurant, "Matrices, moments and quadrature," *Pitman Research Notes in Mathematics Series*, pp. 105–105, 1994.

[2] C. Li, S. Sra, and S. Jegelka, "Gaussian quadrature for matrix inverse forms with applications," in *Proceedings of the 33rd International Conference on Machine Learning - Volume 48*. JMLR.org, 2016, p. 1766–1775.

[3] S. Ubaru, J. Chen, and Y. Saad, "Fast estimation of tr(f(a)) via stochastic lanczos quadrature," *SIAM Journal on Matrix Analysis and Applications*, vol. 38, no. 4, pp. 1075–1099, 2017.

[4] B. Ghorbani, S. Krishnan, and Y. Xiao, "An investigation into neural net optimization via hessian eigenvalue density," in *International Conference on Machine Learning*, 2019, pp. 2232–2241.

[5] B. A. Pearlmutter, "Fast exact multiplication by the hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994.

[6] P. L. Falb, "On a theorem of bochner," *Publications Mathématiques de l'IHÉS*, vol. 36, pp. 59–67, 1969.

[7] T. Dao, C. De Sa, and C. Ré, "Gaussian quadrature for kernel features," *Advances in neural information processing systems*, vol. 30, 09 2017.

[8] S. A. Smolyak, "Quadrature and interpolation formulas for tensor products of certain classes of functions," in *Doklady Akademii Nauk*, vol. 148, no. 5. Russian Academy of Sciences, 1963, pp. 1042–1045.

[9] L. Wu, Y. Yang, and H. Liu, "Nonnegative-lasso and application in index tracking," *Computational Statistics & Data Analysis*, vol. 70, pp. 116–126, 2014.