随机数值线性代数：原理及应用

张鹤龄 电子工程系 2024310593

摘要—线性代数作为一种常用的数据处理手段，被广泛应用于各个领域。近年来，随着各领域对数据处理需求的飞速上升，传统的分析方法已不再适用于越来越大的数据规模。相对地，随机数值线性代数（RandNLA）作为一种数据降维手段正日益被人们关注。通过对数据进行随机采样、嵌入低维子空间等手段降低待处理的数据规模，RandNLA 只牺牲少量精度便可明显降低数据处理的复杂度。在本文中，我们将简要介绍 RandNLA 实现数据降维的基本原理，随后以最小二乘、低秩逼近、矩阵相乘为例，展示其在解决实际问题中的应用。

关键词—随机数值线性代数，数据分析，矩阵分析，最小二乘，低秩逼近，深度学习

I. 引言

关于矩阵分析的讨论始终围绕着几个重要问题进行。在数据拟合领域，最小二乘是求解线性模型系数的重要手段；低秩逼近和对应的特征值/奇异值分析广泛应用于模式识别等领域；矩阵乘法则广泛存在于各个领域的科学计算中。然而，随着近年来数据规模的不断增大，在这些重要问题上，传统的数据分析方法正在逐渐遇到困难。为与本文着重讨论的随机方法相区分，我们称一切常用的除随机方法外的矩阵分析方法为“确定性方法”。

A. 最小二乘

最小二乘问题的一般描述为求解

$$\mathbf{x}^* = \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \tag{1}$$

其中 $\mathbf{A} \in \mathbb{R}^{m\times n}$ 为按行排列的数据矩阵，每行代表一条数据，不同列代表不同的数据分量，$\mathbf{b}$ 为回归目标，$\mathbf{x}$ 是线性模型的系数。由于数据量通常较大，这里假设 $m \gg n$。

最小二乘问题的解由著名的 Moore-Penrose 逆给出：

$$\mathbf{x}^* = \mathbf{A}^\dagger \mathbf{b} \tag{2}$$

其中 $\mathbf{A}^\dagger$ 为矩阵 $\mathbf{A}$ 的 Moore-Penrose 逆，而 $\mathbf{x}^*$ 是满足均方误差 $\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2$ 最小的线性空间中模最小的解。虽然这个结果足够令人满意，但求解 Moore-Perose 逆的时间复杂度为 $\mathcal{O}(mn^2)$，随问题规模 m 线性增长，随数据维度 n 平方增长 [1]。当数据规模增大时，求解 Moore-Penrose 逆也将变得非常困难。在部分应用场景中，$\mathbf{x}$ 的精度并没有很高的要求，这就需要一种牺牲部分精度以换取较低复杂度的方法。

B. 低秩逼近

为降低矩阵规模，减小计算和存储开销，人们倾向于用具有较为简单结构的矩阵进行组合，从而近似数据矩阵。低秩逼近是其中一个重要的近似方法。其描述如下：

给定数据矩阵 $\mathbf{A}$，寻找 $\tilde{\mathbf{A}} = \mathbf{Q}\mathbf{Q}^\mathbf{T}\mathbf{A}$ 使

$$\epsilon = \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^\mathbf{T}\mathbf{A}\| \tag{3}$$

尽量小，其中 $\|\cdot\|$ 为某种度量（依问题选定，通常为谱范数或 Frobenius 范数）。$\mathbf{Q}$ 为逼近矩阵 $\tilde{\mathbf{A}}$ 列空间的一组基。显然，$\mathbf{Q}$ 的选取应考虑如下两个核心问题：

1) 如何确定基向量的规模 k?
2) 如何选取基向量?

在随机方法中，首先对随机矩阵进行采样，随后基于采样后的矩阵寻找合适的基向量。由于采样得到的矩阵规模较小，对其可以采用传统方法，以低得多的复杂度寻找基向量。根据基向量选取方式的不同，发展出 SVD、ID 等多种低秩逼近方法，这些方法已经被广泛应用于数据处理和深度学习等多个领域 [2]。本文的第三节将对这些方法分别进行简要介绍。

C. 矩阵相乘

给定矩阵 $\mathbf{A} \in \mathbb{R}^{m\times n}$ 和 $\mathbf{B} \in \mathbb{R}^{n\times p}$，计算矩阵乘积 $\mathbf{AB}$ 的复杂度为 $\mathcal{O}(mnp)$。使用计算机计算矩阵乘法需要从存储器加载被计算矩阵到 RAM，之后完成二者的相乘。当 m，n 和 p 均很大时，由于计算和存储资源的限制，直接计算矩阵乘积很可能是不可行的。然而，考虑到 $\mathbf{A}$ 和 $\mathbf{B}$ 可能具有的低秩特性，我们可以分别从 $\mathbf{A}$

和 **B** 中抽样出部分行和部分列代替 **A** 和 **B** 本身估计矩阵乘积的结果，从而降低计算的复杂度。利用蒙特卡洛方法，进行多次独立的估计后取均值，将进一步降低误差。[3]

对上述问题的分析给出一个共同的结论：为更有效率地进行数据分析，对原数据进行降维是不可避免的。作为对数据降维的有效手段，RandNLA 主要关注以下几个问题：

1) 如何对数据进行随机采样，使得采样的结果保持原数据的性质?
2) 如何将这些采样应用于具体问题？基于采样结果的解在多大程度上接近原问题的解?
3) 给定解的精度要求，随机方法下的计算复杂度将如何变化？哪些方法可以进一步降低复杂度?

作为一篇旨在介绍 RandNLA 的文献综述，本文将依次对这些问题做出解答。内容安排如下：第二节概述 RandNLA（随机数值线性代数）实现数据降维的核心原理；第三、四、五通过最小二乘、低秩逼近、矩阵相乘等具体问题展示其实际应用。第六节则对全文内容做一总结。

II. 数据降维

我们希望对数据矩阵 **A** 中的数据进行随机抽取或线性组合：

$$\mathbf{Y} = \mathbf{SA} \tag{4}$$

其中 **Y** 为采样得到的矩阵，左乘 $\mathbf{S} \in \mathbb{R}^{k\times m}$ 对数据进行采样和（或）组合。我们不规定 **S** 的具体形式，但其应具有以下两个特征：

1) **S** 从某个分布 Π 中随机地产生；
2) **S** 的行数 $k \ll m$。

作为采样矩阵，**S** 不应破坏原数据矩阵的性质。最基本地，经过 **S** 的作用，**A** 包含的不同数据分量不应被扩张或压缩得太严重，以至于影响后续问题的求解精度。由此有以下定义 [4], [5]:

定义 1 (子空间嵌入精度)**.** 若对 **A** 和任意 **x**，矩阵 **S** 满足

$$(1-\epsilon)\|\mathbf{Ax}\|_2 \le \|\mathbf{SAx}\|_2 \le (1+\epsilon)\|\mathbf{Ax}\|_2 \tag{5}$$

则称 **S** 为 **A** 的一个精度为 ϵ 的 l_2 子空间嵌入。如果我们要求得更严格，还可以定义 Johnson-Lindenstrauss 嵌入。[5] 由于二者类似，这里不再赘述。

如何获得具有这样性质的矩阵 **S**? 幸运的是，对这个问题不需要做太多的努力，通过从独立高斯分布中采样便可获得；通过增大采样后的数据规模 k，可以提高在一定精度 ϵ 下 **S** 满足精度要求的成功概率，或是固定成功概率而提高精度。以下给出相关的结论：[5]

定理 1. *若 $\mathbf{S} = \frac{1}{k}\mathbf{R}$, **R** 的每个元素均为独立且服从标准高斯分布的随机变量，$k = C(n + log(1/\delta))\epsilon^{-2}$，则对任意 $\mathbf{A} \in \mathbb{R}^{m\times n}$，**S** 为 **A** 的一个精度为 ϵ 的 l_2 子空间嵌入之概率不低于 $1-\delta$。这里 C 为正常数。*

对于 J-L 嵌入也有类似的结论。注意到采样规模 k 对数依赖于失败概率 δ 而与精度 ϵ 成平方反比关系，这意味着通过增大采样规模提高成功概率相对简单，但提高精度是非常困难的。在精度要求不高的场合，单次随机采样足以给出可靠的数据近似 **Y** 用于下游任务；若需进一步提升精度则需迭代处理（详见第七节）。

尽管从高斯分布中采样较为直接，但采样本身的复杂度较高，同时由于采样的结果是浮点数，计算矩阵相乘也相对复杂。因此，常用其他方法产生采样矩阵 **S** 来达到相似的效果。

A. 稀疏符号采样

用元素为 1 或 0 的符号矩阵代替高斯矩阵是一种常用的实现方法，实践证明其效果与高斯采样相近。[4], [6] 在采样矩阵的每一列中，随机选取 d 个元素为 1，其余元素为 0。而后进行归一化：

$$\mathbf{S} = \sqrt{\frac{1}{d}}[\mathbf{s_1}, ..., \mathbf{s_m}] \tag{6}$$

在 **A** 列满秩的情况下，选取采样规模 $k = \mathcal{O}(nlogn)$ 和稀疏度 $d = \mathcal{O}(logn)$ 可以保证精度。

特别地，计算 **SA** 的复杂度仅为 $\mathcal{O}(mnd)$ 而非通常矩阵相乘的 $\mathcal{O}(mnk)$。通常 d 的值甚至小于 $log(n)$，因此不仅相对于高斯采样，稀疏符号采样相对于其他快速采样方法仍然具有速度优势。[6]

相比左乘采样矩阵 **S**，直接选取数据矩阵 **A** 中的某些元素进行采样更加简单，但在 **A** 能量分布明显不均匀时，随机选取其中的元素得到的结果未必能保持矩阵的原有性质。例如，若 **A** 仅有少数元素为 1，其他元素均为 0，则均匀随机选取元素的结果很可能是一个全零矩阵 [2]。因此，在选取元素之前，需对数据矩阵 **A** 进行预处理。由此发展出以下的几种方法：

B. 快速 *J-L* 变换

不确定性原理说明，稀疏的信号在频域（或其他变换域）必然是稠密的。因此，对于稀疏矩阵 **A**，可以对其进行傅里叶变换 **F** 或进行 Hadamard 变换 **H**，将能量分散到整个矩阵当中。随机选取变换后的矩阵元素，将得到较好的效果。快速 J-L 变换实现了这一过程。[5]

$$\mathbf{S} = \mathbf{PHD} \tag{7}$$

采样矩阵 **S** 由三部分构成。其中，**D** 为对角矩阵，对角元为独立同分布的随机变量，分别以 1/2 概率取 1 或-1。其作用为随机旋转数据矩阵，保证经 Hadamard 变换后的矩阵能量分布足够均匀；**H** 为 Hadamard 变换矩阵；**P** 则对矩阵元素进行随机选取，三者依次作用于数据矩阵。

由于元素选取和随机翻转的复杂度较低，快速 J-L 变换的复杂度集中于 Hadamard 变换。与高斯采样对应的矩阵相乘操作（复杂度为 $\mathcal{O}(mnk)$）相比，Hadamard 变换可以快速进行，其复杂度为 $\mathcal{O}(mnlogm)$，这就提升了计算效率。

C. 随机抽样三角变换

与快速 J-L 变换类似，随机抽样三角变换同样对数据矩阵进行预处理，但其采用的并非 Hadamard 变换，而是一类三角变换：在被处理矩阵为实数的情况，一般采用离散余弦变换和 Hartley 变换；复数情况则采用离散傅里叶变换。随机抽样三角变换的表达如下 [4]：

$$\mathbf{S} = \sqrt{\frac{m}{k}}\mathbf{RFE}\Pi \tag{8}$$

其中 Π 为随机置换矩阵，**E** 为随机旋转矩阵（与上一节的 **D** 类似），**F** 为三角变换矩阵，**R** 随机选取和组合变换后数据矩阵的行。为使得数据矩阵的能量分布更加均匀，可以独立地产生多个 **FE**Π 对其进行多次变换。利用快速算法降低三角变换的复杂度后，这一采样方法的时间复杂度为 $\mathcal{O}(mnlogk)$。

以上的两种方法均对变换后的矩阵元素进行抽样，得到的抽样结果是原数据矩阵元素的线性组合。另外一种思路则是提取数据本身的统计特征后，再直接对数据矩阵进行抽取。相关的方法称为杠杆值抽样。

D. 杠杆值抽样

行杠杆值衡量数据矩阵每行包含的信息量。其定义如下 [6]：

定义 2 (行杠杆值)**.** 数据矩阵 **A** 的第 $i(1 \le i \le m)$ 行的杠杆值为

$$l_i = \|\mathbf{P_A}\delta_i\|_2 \tag{9}$$

其中 δ_i 为第 i 个标准单位向量，$\mathbf{P_A}$ 为投影到 **A** 列空间的投影矩阵。

通过对 **A** 进行分解，我们可以进一步解读杠杆值的含义。不妨假设 **A** 秩为 n，其列向量张成的空间的其中一组标准正交基为 **Q**：

$$\mathbf{A} = \mathbf{QB}, \mathbf{Q} \in \mathbb{R}^{m\times n} \tag{10}$$

则有 $\|\mathbf{P_A}\delta_i\|_2 = \delta_i^T\mathbf{QQ^TQQ^T}\delta_i = \delta_i^T\mathbf{QQ^T}\delta_i = \|\mathbf{Q^T}\delta_i\|_2$。对于如上的 **QB** 分解，可以认为 **B** 的行包含了构建 **A** 所需要的信息，而 $\|\mathbf{Q^T}\delta_i\|_2$ 则代表将 **B** 的各行组合成 **A** 的第 i 行时组合系数包含的能量。这一能量越大，表明 **A** 的第 i 行包含的信息越多，应该在采样时重点关注。

如果已经求得 **A** 的杠杆值，便可以根据杠杆值对数据矩阵进行重要度采样。通过对杠杆值归一化，求得 **A** 第 i 行被采样的概率为：

$$p_i = \frac{l_i}{\sum_i l_i} \tag{11}$$

根据这一概率分布，设计采样矩阵 **S**：**S** 的每行以概率 p_i 取 $\delta_i/\sqrt{p_ik}$，以抽取第 i 行数据和归一化，并将这一过程重复 k 次。为使这样得到的 **S** 为 **A** 精度为 ϵ 的嵌入，需要的采样规模为

$$k \ge 2\epsilon^{-2}nlog(2n) \tag{12}$$

与之相对，若对每一行均匀采样，则需要的采样规模增加到

$$k \ge 2\epsilon^{-2}\mu(\mathbf{A})log(2n) \tag{13}$$

其中 $\mu(\mathbf{A})$ 衡量 **A** 的杠杆值分布差异，在杠杆值分散的情况下，$\mu(\mathbf{A}) \gg n$。因此，在按行采样的情况下，按杠杆值进行重要度采样可以大幅降低所需的采样规模。[4]

然而，由于计算杠杆值通常涉及计算 **A** 的子空间，需要对其进行复杂度较高的分解，一般仅对 p_i 进行估计。记估计得到的分布为 $\{q_1, ..., q_m\}$，分布估计的质量 β 可通过下式衡量：

$$\beta = \min_i \frac{q_i}{p_i} \tag{14}$$

显然 $\beta \le 1$，当且仅当 $\beta = 1$ 时估计完全准确。以下定理保证了估计的质量：

定理 2. 给定采样规模 k，精度 ϵ 和估计质量 β，则按行抽样得到的采样矩阵 $\mathbf{S}$ 是 $\mathbf{A}$ 的一个精度为 ϵ 的子空间嵌入之概率不小于 *[6]*

$$1-\delta \geq 1-2n(\frac{e^{\epsilon}}{(1+\epsilon)^{(1+\epsilon)}})^{\beta k/n} \tag{15}$$

定理 3. 给定精度 ϵ 和估计质量 β，达到成功概率 $1-\delta$ 所需要的采样规模为

$$k > 144n\ ln(2n/\delta)/(\beta\epsilon^2) \tag{16}$$

估计分布 q_i 可以在较低的复杂度下获得 [5]。总的来说，通过估计杠杆值分布和进行按行的重要度采样，我们可以构建规模较小的子数据集而不影响数据的性质。

通过高斯采样、数据预处理和杠杆值采样等方法，我们获得了采样矩阵 $\mathbf{S}$ 和数据集降维后的低维“草图” $\mathbf{Y}=\mathbf{SA}$。其中，高斯采样的理论分析较为完备，但复杂度较高，通常使用稀疏符号采样作为替代；预处理方法拥有较低的复杂度，虽然欠缺理论分析，但实际效果与高斯采样相仿；杠杆值采样保留了原数据集的元素，具有较好的可解释性，在部分低秩逼近问题中较为实用，但精度往往较低 [4]。根据处理的问题不同，应选择不同的数据降维方式。为简洁起见，本文在之后的内容中，假设 $\mathbf{S}$ 已经通过某种方法得到，而不再对具体选择何种降维方法加以讨论。

III. 最小二乘

最小二乘问题已在 (1) 中描述。为降低其求解复杂度，RandNLA 首先生成 $[\mathbf{A}\ \mathbf{b}]$ 的一个精度为 ϵ 的子空间嵌入 $\mathbf{S}$，而求解下列问题作为替代：

$$\tilde{\mathbf{x}}^* = \min_{\mathbf{x}} \|\mathbf{SAx}-\mathbf{Sb}\|_2 \tag{17}$$

精度 ϵ 足够小的情况下，问题的解 $\tilde{\mathbf{x}}^*$ 是原问题解的合理近似，其近似精度为 [7]

$$\|\mathbf{b}-\mathbf{A}\tilde{\mathbf{x}}^*\|_2 \leq (1+\epsilon)\|\mathbf{b}-\mathbf{Ax}^*\|_2 \tag{18}$$

$$\|\mathbf{x}^*-\tilde{\mathbf{x}}^*\|_2 \leq \sqrt{\epsilon} cond(\mathbf{A})\ \sqrt{\gamma^{-2}-1}\|\mathbf{x}^*\|_2 \tag{19}$$

其中 $cond(\mathbf{A})$ 是 $\mathbf{A}$ 的条件数，$\gamma=\frac{\|\mathbf{P_A}b\|_2}{\|\mathbf{b}\|_2}$ 是 $\mathbf{b}$ 落在 $\mathbf{A}$ 列空间中的能量比例。特别地，对于利用杠杆值采样求解最小二乘问题的误差分析见 [1]。通过单次采样求解最小二乘的复杂度为采样复杂度和计算 Moore-Penrose 逆复杂度的和 $\mathcal{O}(mnlog(k)+kn^2)$。

对于精度要求不太高的问题，单次采样求解最小二乘便可满足要求。然而，如第二节所述，要产生精度为 ϵ 的嵌入 $\mathbf{S}$，需要的采样规模 k 正比于 ϵ^{-2}。当 ϵ 接近机器精度时，需要的采样规模巨大，因此不可能通过单次采样获得最小二乘问题足够精确的解。以下两个方法可以有效地降低高精度要求时的求解复杂度。

A. 迭代采样法

我们可以通过迭代求解，使采样规模对精度的依赖下降至 $k=\mathcal{O}log(\epsilon)$，从而使高精度求解变得可行。迭代算法见算法 1 [4]：

Algorithm 1 迭代最小二乘

Input: $\mathbf{A}$，$\mathbf{b}$，嵌入精度为 ϵ_0，求解精度 η

Output: $\tilde{\mathbf{x}}^*$

1: $\mathbf{x}\leftarrow\mathbf{0}$，$\mathbf{r}\leftarrow\mathbf{b}$
2: **while** $\|\mathbf{r}\|_2 \geq \eta$ **do**
3: $\quad\mathbf{r}\leftarrow\mathbf{b}-\mathbf{Ax}$
4: $\quad$产生精度为 ϵ_0 的嵌入 $\mathbf{S}\in\mathbb{R}^{d\times m}$
5: $\quad\mathbf{x}\leftarrow\mathbf{x}+\min_{\mathbf{x}'}(\|\mathbf{SAx}\|_2-\langle\mathbf{A^T r},\mathbf{x}\rangle)$
6: **end while**
7: $\tilde{\mathbf{x}}^*\leftarrow\mathbf{x}$

其中第 5 步利用了性质 $\|\mathbf{SAx}\|_2\approx\|\mathbf{Ax}\|_2$。上述迭代的残差 $\mathbf{r}$ 线性收敛至原问题的残差 $\mathbf{b}-\mathbf{Ax}^*$，故要得到理想的精度 ϵ，所需的迭代次数为 $j=\mathcal{O}(log(1/\epsilon))$。若设定嵌入精度为常数 ϵ_0，则 $k=\mathcal{O}(nlogn)$，总的计算复杂度为 $\mathcal{O}((mn^2+n^3)log(n)log(1/\epsilon))$。

B. 预处理法

迭代采样法的一个突出缺点是需要多次独立产生采样矩阵 $\mathbf{S}$ 并计算 $\mathbf{SA}$。与迭代采样法不同，预处理法直接对原问题进行求解，并且只需产生一次 $\mathbf{S}$ 用于对 $\mathbf{A}$ 进行预处理。其思路如下 [4], [6]：对采样得到的 $\mathbf{Y}$，有 $\mathbf{QR}$ 分解：

$$\mathbf{SA}=\mathbf{Y}=\mathbf{QR_Y} \tag{20}$$

选取合适的采样规模，可以认为 $\mathbf{Y^TY}\approx\mathbf{A^TA}$。对 $\mathbf{S}$ 和 $\mathbf{A}$ 分别进行 QR 分解得 $\mathbf{R_Y^TR_Y}\approx\mathbf{R_A^TR_A}$，进而 $\mathbf{R_Y}\approx\mathbf{R_A}$。因此可用 $\mathbf{R_Y}$ 对最小二乘问题进行预处理：

$$\tilde{\mathbf{x}}^* = \min_{\mathbf{x}} \|\mathbf{AR_Y^{-1}R_Yx}-\mathbf{b}\|_2 \tag{21}$$

预处理矩阵 $\mathbf{AR^{-1}}$ 近似列正交，因此使用共轭梯度法求解上述问题，每次迭代的复杂度为 $\mathcal{O}(mn)$，与迭代采样

法类似，同样迭代 $\mathcal{O}(log(1/\epsilon))$ 次。预处理法总的复杂度为采样、QR 分解和迭代复杂度的和：$\mathcal{O}(mnlog(n)+n^3log(n)+mnlog(1/\epsilon))$。

由于只需要进行一次采样，预处理法的效率高于迭代采样法，这也使其成为较常用的最小二乘求解方法[4]–[6]。

C. 应用：深层随机参数网络优化

深层随机参数网络（Deep Stochastic Configuration Networks, DeepSCN）是一种易于训练的网络架构，其每层的节点数量和层数由迭代算法决定。记输入数据为 $\mathbf{x}\in\mathbb{R}^n$，在第 m 层节点数量、参数 $\mathbf{W_m},\mathbf{b_m}$ 和层数 n 均确定的情况下，最终的输出 $\hat{\mathbf{t}}\in\mathbb{R}^c$ 由下式决定：

$$\hat{\mathbf{t}}=\mathbf{H}\boldsymbol{\beta} \tag{22}$$

其中 $\mathbf{H}=[\mathbf{h_1},\mathbf{h_2},...,\mathbf{h}_\sigma]$ 为所有 $\sigma=\sum_m L_m$ 个节点的输出总和。在决定层数时，最小二乘问题 $min\ \|\mathbf{H}\boldsymbol{\beta}-\mathbf{t}\|_2$ 被反复计算（其中 $\mathbf{t}$ 为数据标签），以决定参数 $\boldsymbol{\beta}$ 和是否应该增加层数。

DeepSCN 的训练流程如下：

Algorithm 2 DeepSCN 训练算法

Input: 数据集 $\mathbf{X}\in\mathbb{R}^{m\times n}$，标签 $\mathbf{T}\in\mathbb{R}^{m\times c}$，训练终止门限 η

Output: 网络层数 n，每节节点个数和参数 $L_n,\mathbf{b}_n,\mathbf{W}_n$

$\epsilon\leftarrow\inf, n\leftarrow 1$
while $\epsilon\geq\eta$ **do**
  $L_n,\mathbf{b}_n,\mathbf{W}_n=Config\_Decider(\mathbf{X},\mathbf{T})$
  $\mathbf{H}_{L_n}=Inference(\mathbf{X};\mathbf{b}_n,\mathbf{W}_n)$
  $\mathbf{H}=[\mathbf{H},\mathbf{H}_{L_n}]$
  $\boldsymbol{\beta}=LS\_Solver(\mathbf{H},\mathbf{T})$
  $\epsilon=\|\mathbf{H}\boldsymbol{\beta}-\mathbf{T}\|_2$
  $n\leftarrow n+1$
end while

由于 $\mathbf{H}$ 规模正比于网络参数规模，多次计算最小二乘严重影响训练效率。同时，$\mathbf{H}$ 本身的较小奇异值也将带来数值不稳定性。通过对其进行基于 SVD 分解的低秩逼近，降低最小二乘的复杂度，可以以牺牲少量网络性能为代价，将计算最小二乘的时间缩短 10 倍左右[8]（基于 SVD 分解的低秩逼近将在下一章说明）。

IV. 低秩逼近

A. 列空间基选取

假设矩阵 $\mathbf{A}$ 是低秩逼近的目标矩阵，且其秩约为 r。为使低秩逼近具有良好效果（式 (3) 中的逼近误差较小），应找到其列空间的 r 个基并将其投影到这些基张成的子空间上。如果对 $\mathbf{A}$ 的列进行线性组合：

$$\mathbf{Y}=\mathbf{AS} \tag{23}$$

则只要 $\mathbf{S}\in\mathbb{R}^{n\times l}$，取 $l\geq r$ 并使得 $\mathbf{S}$ 列满秩，得到的 $\mathbf{Y}$ 张成的列空间将是 $\mathbf{A}$ 列空间的一个至少 r 维的子空间。直接对 $\mathbf{Y}$ 进行 $\mathbf{QR}$ 分解即可得到这个子空间的基。选取的 l 通常稍大于 r，称 $p=l-r$ 为过采样参数。一般而言选取较小的 p（例如 5 或 10）即可，但 p 的选择也和 $\mathbf{A}$ 的性质有关：$\mathbf{A}$ 的规模较大，或奇异值下降的速度较大时，则需要的过采样程度 p 也更大。列组合矩阵 $\mathbf{S}$ 在这里起到数据降维作用，它和第二节的采样矩阵类似，因此同样可以通过高斯采样、快速 J-L 变换等方法产生。[4], [9]

更多情况下，矩阵 $\mathbf{A}$ 的秩难以确定，需要观察估计误差来确定预先选择的秩 r 是否足够近似 $\mathbf{A}$。若限定重建误差 $\|(\mathbf{I}-\mathbf{QQ^T})\mathbf{A}\|_2<\epsilon$，则在选择 l 个基向量 $\mathbf{Q}^{n\times l}$ 之后，可以通过下式估计重建误差 [9]：

$$\|(\mathbf{I}-\mathbf{QQ^T})\mathbf{A}\|_2\leq 10\sqrt{\frac{2}{\pi}}\max_{i=1,2,...,k}\|(\mathbf{I}-\mathbf{QQ^T})\mathbf{A}\boldsymbol{\omega}_i\|_2 \tag{24}$$

其中 $\boldsymbol{\omega}_i, 1\leq i\leq k$ 从标准高斯分布 $\mathcal{N}(\mathbf{0},\mathbf{I}_n)$ 中独立采样 k 次得到。式 (24) 的成立几率大于 $1-10^{-k}$，故只需取适当 k，(24) 几乎必然成立。

进一步，在对 $rank(\mathbf{A})$ 没有任何先验知识的情况下，可以选取足够大的采样规模 l，保证采样结果 $\mathbf{Y}$ 的秩等于 $rank(\mathbf{A})$；或是，在选取基时，迭代地增加 $\mathbf{Q}$ 的维数，并在每次增加维数后检验重建误差是否满足要求，在误差足够小时停止向 $\mathbf{Q}$ 中添加新的向量 [9]。算法如下：

B. 随机奇异值分解 (RSVD)

随机奇异值分解是较为常用的低秩逼近算法，基于列空间基选取实现。假设通过某种列空间基选取的手段，已经获得足够精确的 $\tilde{\mathbf{A}}=\mathbf{QQ^TA}$ 和 $\mathbf{Q}$。则存在如下的近似分解 [4]：

$$\mathbf{A}\approx\mathbf{QQ^TA}=\mathbf{Q}(\mathbf{Q^TA})=\mathbf{Q}\hat{\mathbf{U}}\boldsymbol{\Sigma}\mathbf{V}^*=\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^* \tag{25}$$

Algorithm 3 迭代基选取算法

Input: 待近似矩阵 $\mathbf{A}$，重建精度 ϵ，检验向量个数 k

Output: 基向量矩阵 $\mathbf{Q}$

从标准高斯分布中采样得 $\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_k$
$\mathbf{y}_i = \mathbf{A}\boldsymbol{\omega}_i$
$j \leftarrow 0$
$\mathbf{Q} \leftarrow []$
while $\max_{i=1,...,k} \|\mathbf{y}_i\|_2 \geq \epsilon$ **do**
  $j \leftarrow j + 1$
  $\mathbf{q}_j \leftarrow \mathbf{y}_j / \|\mathbf{y}_j\|_2$
  $\mathbf{Q} \leftarrow [\mathbf{Q},\ \mathbf{q}_j]$
  从标准高斯分布中采样得 $\boldsymbol{\omega}_{j+k}$
  $\mathbf{y}_{j+k} \leftarrow (\mathbf{I} - \mathbf{Q}\mathbf{Q}^{\mathbf{T}})\boldsymbol{\omega}_{j+k}$
  $\mathbf{y}_i = \mathbf{y}_i - \mathbf{q}_j \langle \mathbf{q}_j, \mathbf{y}_i \rangle, i = j+1, ..., j+k-1$
end while

Algorithm 4 RSVD

Input: 待近似矩阵 $\mathbf{A}$

Output: 近似的奇异值分解 $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*$

$\mathbf{Q} \leftarrow Rangefinder(\mathbf{A})$
$\mathbf{C} \leftarrow \mathbf{Q}^{\mathbf{T}}\mathbf{A}$
$\hat{\mathbf{U}}, \boldsymbol{\Sigma}, \mathbf{V}^* \leftarrow svd(\mathbf{C})$
$\mathbf{U} \leftarrow \mathbf{Q}\hat{\mathbf{U}}$

故 RSVD 可以通过如下的方法实现：

其中，Rangefinder() 表示任意的基选取算法。RSVD 已经被用于加速深度学习领域中的网络优化，上节中 DeepSCN 的加速算法就基于 RSVD [8]；另外，RSVD 也被用于快速计算二阶优化方法中的 Fisher 信息矩阵 [10]（将在本章最后介绍）。

C. 插值分解 (ID)

对于数据矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$，奇异值分解是重要的数据特征提取手段，例如，通过对采集到个体核苷酸数据主成分的分析，可以建立地域与基因间的相关关系 [2]。然而，经过分解得到的特征向量和特征值并没有明确的物理意义。考虑到若在数据矩阵中抽出少量的代表数据作为线性空间的基底，这些基底将不仅具有明确的物理含义，而且张成的子空间也可能包含数据集中的大部分数据，用这些数据进行原矩阵 $\mathbf{A}$ 的低秩逼近可能可解释性上均更具优势。由此，发展出插值分类方法。

插值分类方法的表示如下：

1) 列插值：$\mathbf{A} = \mathbf{CZ}$

2) 行插值：$\mathbf{A} = \mathbf{XR}$

其中的 $\mathbf{C}$ 和 $\mathbf{R}$ 分别为原矩阵 $\mathbf{A}$ 的部分列和部分行组成的矩阵。记 $\mathbf{C}$ 和 $\mathbf{R}$ 在原矩阵中对应的列和行索引集合为 $\mathcal{I}, \mathcal{J}$，则 $\mathbf{C} = \mathbf{A}[:, \mathcal{I}], \mathbf{R} = \mathbf{A}[\mathcal{J}, :]$。$\mathbf{Z}$ 和 $\mathbf{X}$ 分别用于插值。由于原理类似，这里主要介绍列 ID 分解。

ID 分解的主要任务是找到列索引集合 $\mathcal{I}$。对于规模较小的矩阵 $\mathbf{A}$，设定列矩阵 $\mathbf{C}$ 的规模 $|\mathcal{I}| = k$，用基于 Gram-Schmidt 分解的贪婪算法可以很好地找到合适的 $\mathcal{I}$。[4] 首先，找到 $\mathbf{A}$ 中模最大的一列，作为 G-S 分解中列正交矩阵 $\mathbf{Q}$ 的第一列；随后的每轮迭代中，将余下的矩阵投影到 $\mathbf{Q}$ 的列空间上，将残量模值最大的列索引加入 $\mathcal{I}$ 的同时，将残量归一化后并入 $\mathbf{Q}$，直到 $\mathcal{I}$ 的规模满足要求。此时有如下等式成立：

$$\mathbf{A}\boldsymbol{\Pi} = \mathbf{QS} + \mathbf{E} \tag{26}$$

贪婪算法的每次迭代需要进行列选取，这使得矩阵 $\mathbf{A}$ 在被 G-S 算法 $\mathbf{QR}$ 分解之前，列向量被重新排列，用矩阵 $\boldsymbol{\Pi}$ 表示。$\mathbf{S}$ 对应 $\mathbf{QR}$ 分解中的上三角矩阵 $\mathbf{R}$，$\mathbf{E}$ 则为矩阵 $\mathbf{A}$ 向 $\mathbf{Q}$ 列空间投影产生的残量。

在贪婪算法的迭代选取过程中，列索引集合 $\mathcal{I}$ 可以直接求得。为推导 ID 分解的具体形式，将 (26) 式进一步展开：由贪婪算法的过程容易得出，若将 $\mathbf{S}$ 表示为分块矩阵

$$\mathbf{S} = [\mathbf{S}_{11}, \mathbf{S}_{12}]\ (\mathbf{S}_{11} \in \mathbb{R}^{k \times k}) \tag{27}$$

则 $\mathbf{QS}_{11}$ 恰为选取的列矩阵 $\mathbf{C}$。代入 (26) 得

$$\mathbf{A} = \mathbf{QS}_{11}[\mathbf{I}_k\ \mathbf{S}_{11}^{-1}\mathbf{S}_{12}]\boldsymbol{\Pi}^T + \mathbf{E}\boldsymbol{\Pi}^T \tag{28}$$

若记 $[\mathbf{I}_k\ \mathbf{S}_{11}^{-1}\mathbf{S}_{12}]\boldsymbol{\Pi}^T = \mathbf{Z}$，则得到 $\mathbf{A}$ 的 ID 分解形式 $\mathbf{A} \approx \mathbf{CZ}$，$\tilde{\mathbf{E}} = \mathbf{E}\boldsymbol{\Pi}^T$ 为逼近误差。

对于规模更大的矩阵 $\mathbf{A}$，则需要采用随机方法进行预处理，降低 ID 的复杂度。与列空间基选取过程类似，这里需要对 $\mathbf{A}$ 的行进行线性组合 $\mathbf{Y} = \mathbf{SA}$；为保持 $\mathbf{A}$ 的列空间在随机采样后大致不变，需要选取 $\mathbf{S}$ 的列数 l 稍大于预先确定的秩 k。这样，可以认为 $\mathbf{A}$ 的各行均位于 $\mathbf{Y}$ 的行空间中：

$$\mathbf{A} = \mathbf{FY} \tag{29}$$

若对 $\mathbf{Y}$ 存在 ID

$$\mathbf{Y} = \mathbf{CX} \tag{30}$$

则有 $\mathbf{A} = \mathbf{FY} = \mathbf{FCX}$。由于 $\mathbf{A} = \mathbf{FY}$，而 $\mathbf{C}$ 又是 $\mathbf{Y}$ 的部分列，容易得到 $\mathbf{FC} = \mathbf{C_A}$ 也是 $\mathbf{A}$ 的部分列。这意味

着，$\mathbf{A}$ 和 $\mathbf{Y}$ 的 *ID* 具有完全相同的列索引，因此通过对规模较小的 $\mathbf{Y}$ 进行 ID，可以直接得到原矩阵 $\mathbf{A}$ 的 ID [4]。

ID 分解也可以通过杠杆值采样获得。通过杠杆值采样，可以获得原矩阵 $\mathbf{A}$ 的一组列 $\mathbf{C}$。在第二节中提到，只要选取采样规模 $k = \mathcal{O}(nlog(2n)/\epsilon^2)$，则 $\mathbf{C}$ 为 $\mathbf{A}$ 精度为 ϵ 的嵌入。推导可得

$$\|\mathbf{A} - \mathbf{P}_{\mathbf{C_k}}\mathbf{X}\|_F \leq (1+\epsilon)\|\mathbf{A} - \mathbf{P}\mathbf{U_k}\mathbf{A}\|_F \tag{31}$$

其中 $\mathbf{C}_k$ 和 $\mathbf{U}_k$ 分别为矩阵 $\mathbf{C}$ 和 $\mathbf{A}$ 前 k 个左特征向量组成的矩阵。鉴于 $\|\mathbf{A} - \mathbf{P}\mathbf{U_k}\mathbf{A}\|_F$ 为对 $\mathbf{A}$ 进行秩 k 逼近的误差下界，通过增大采样规模减小 ϵ 可以获得精度可靠的低秩逼近。

D. 应用：快速 K-FAC 优化方法

K-FAC (Kronecker-Factored Approximate Curvature) 是针对自然梯度方法 (Natural Gradient) 提出的近似算法。自然梯度的形式如下：

$$\nabla_{NG} f(\boldsymbol{\theta}) = \mathbf{F}^{-1}\nabla f(\boldsymbol{\theta}) \tag{32}$$

其中 $\mathbf{F}$ 为 Fisher 矩阵，$\boldsymbol{\theta} = [\boldsymbol{\theta}_1^T, \boldsymbol{\theta}_2^T, ..., \boldsymbol{\theta}_n^T]^T$ 为按层排列的网络参数，$f(\cdot)$ 为损失函数。通过将 Fisher 矩阵的逆作用于梯度向量，自然梯度法可以提高优化效率。

由于 Fisher 矩阵的规模正比于网络参数的数量，计算其逆是十分困难的。K-FAC 将 Fisher 矩阵的结构做简化近似，仅考虑 Fisher 矩阵的层内部分，而将层间部分的影响置零。由于参数 $\boldsymbol{\theta}$ 是按层排列的，这对应于将 Fisher 矩阵的非对角块置零。进一步推导 Fisher 矩阵，得到其 Kronecker 分解形式：

$$\mathbf{F}_{K-FAC} = blockdiag\{\mathcal{A}_n \otimes \boldsymbol{\Gamma}_n\} \tag{33}$$

其中 $\mathcal{A}_n$，$\boldsymbol{\Gamma}_n$，分别为第 n 层的输入自相关矩阵和对第 n 层参数的梯度自相关矩阵。通过将 Fisher 矩阵近似为块对角矩阵，求逆操作得到简化：

$$\mathbf{F}_{K-FAC}^{-1} = blockdiag\{\mathcal{A}_n^{-1} \otimes \boldsymbol{\Gamma}_n^{-1}\} \tag{34}$$

将上式代入 $\nabla_{NG} f(\boldsymbol{\theta})$。对第 k 层的参数 $\boldsymbol{\theta}_k$，自然梯度为

$$\nabla_{NG} f(\boldsymbol{\theta}_k) = vec(\boldsymbol{\Gamma}_k^{-1}\ Mat(\nabla f(\boldsymbol{\theta}_k))\mathcal{A}_k^{-1}) \tag{35}$$

其中 $vec(\cdot)$ 和 $Mat(\cdot)$ 分别将矩阵拉伸为向量/将向量组织为矩阵。考虑到自相关矩阵 $\mathcal{A}$，$\boldsymbol{\Gamma}$ 分别有 SVD 分解 $\mathbf{U}_{\boldsymbol{\Gamma}}\boldsymbol{\Sigma}_{\boldsymbol{\Gamma}}\mathbf{V}_{\boldsymbol{\Gamma}}^T$ 和 $\mathbf{U}_{\mathcal{A}}\boldsymbol{\Sigma}_{\mathcal{A}}\mathbf{V}_{\mathcal{A}}^T$，自然梯度可以借助 SVD 分解进行：

$$\begin{aligned}\nabla_{NG} f(\boldsymbol{\theta}_k) = vec(&\mathbf{V}_{\boldsymbol{\Gamma}}(\boldsymbol{\Sigma}_{\boldsymbol{\Gamma}} + \lambda\mathbf{I})^{-1}\mathbf{U}_{\boldsymbol{\Gamma}}^T\ Mat(\nabla f(\boldsymbol{\theta}_k))\\ &\mathbf{V}_{\mathcal{A}}(\boldsymbol{\Sigma}_{\mathcal{A}} + \lambda\mathbf{I})^{-1}\mathbf{U}_{\mathcal{A}}^T)\end{aligned} \tag{36}$$

Puiu 在 [10] 中指出，$\boldsymbol{\Gamma}_k^{-1}$ 和 $\mathcal{A}_k^{-1}$ 均可以进行秩 r 的低秩逼近，从而降低 (36) 中矩阵相乘的复杂度。据此，[10] 提出了一种基于 RSVD 的计算加速方法，用于近似 (34) 中自相关矩阵的逆：

$$\begin{aligned}\nabla_{NG} f(\boldsymbol{\theta}_k) \approx vec(&\mathbf{V}_{\boldsymbol{\Gamma}}^{(r)}(\boldsymbol{\Sigma}_{\boldsymbol{\Gamma}}^{(r)} + \lambda\mathbf{I})^{-1}\mathbf{U}_{\boldsymbol{\Gamma}}^{(r)T}\ Mat(\nabla f(\boldsymbol{\theta}_k))\\ &\mathbf{V}_{\mathcal{A}}^{(r)}(\boldsymbol{\Sigma}_{\mathcal{A}}^{(r)} + \lambda\mathbf{I})^{-1}\mathbf{U}_{\mathcal{A}}^{(r)T})\end{aligned} \tag{37}$$

其中上标 (r) 表示取最大的 r 个特征值和特征向量。通过进行低秩逼近，在不影响训练效果的前提下可将训练时间缩短至原先的约 1/3 [10]。

V. 矩阵相乘

以最小二乘和低秩逼近为代表的矩阵计算采用的随机方法，利用随机生成的单个降维矩阵 $\mathbf{S}$ 对原矩阵 $\mathbf{A}$ 进行预处理并应用于后续计算，这相当于用单次随机试验的结果估计问题的精确解。与这些方法不同，在矩阵相乘和矩阵函数估计等问题中，广泛采用蒙特卡洛方法，算法思路为：

1) 从某一简单分布 $\mathcal{P}$ 中采样得到随机矩阵 $\mathbf{X}$；
2) 以原矩阵 $\mathbf{A}$ 和随机矩阵 $\mathbf{X}$ 为自变量，构建合适的随机变量 $F = f(\mathbf{A}, \mathbf{X})$，使得 $\mathbb{E}_{\mathcal{P}}(F) = F^*$，其中 F^* 为问题的精确解；
3) 独立对 F 进行 k 轮采样，取平均值得到 F^* 的估计。
4) 另外，还需计算 $Var(F)$ 用于决定合适的采样轮次 k。

问题的关键在于如何构建合适的随机变量 F^*。本节以矩阵相乘和矩阵函数估计为例，给出这些问题中随机变量的构造方法。

A. 矩阵相乘

给定 $\mathbf{A} \in \mathbb{R}^{m\times n}$ 和 $\mathbf{B} \in \mathbb{R}^{n\times p}$，通过对 $\mathbf{A}$ 和 $\mathbf{B}$ 分别进行列采样和行采样，可以得到矩阵相乘的大致结果。记采样矩阵为 $\mathbf{S}$，算法如下 [3]：可以证明如此构造的随机变量 $\mathbf{F}$ 之期望恰为 $\mathbf{AB}$：

Algorithm 5 矩阵乘法估计

Input: $\mathbf{A}$，$\mathbf{B}$，采样规模 c，分布列 $\mathcal{P}=\{p_1,...,p_n\}$

Output: 矩阵乘积估计值 $\mathbf{F}\approx\mathbf{AB}$

1: $\mathbf{S}\leftarrow[],\mathbf{D}=\mathbf{0}\in\mathbb{R}^{c\times c}$
2: **for** i=1,2,...,c **do**
3: 随机选取 $\mathbf{s}=\boldsymbol{\delta}_{i_t},Pr\{i_t=k\}=p_k$
4: $\mathbf{S}\leftarrow[\mathbf{S},\mathbf{s}]$
5: $\mathbf{D}_{ii}=\frac{1}{\sqrt{cp_{i_t}}}$
6: **end for**
7: $\mathbf{F}=\mathbf{ASDD^TS^TB}$

证明. 由 $\mathbf{F}_{kl}=\sum_{i=1}^{c}\frac{1}{cp_{i_t}}\mathbf{A}_{ki_t}\mathbf{B}_{i_tl}$ 得:

$$\mathbf{F}_{kl}=\sum_{i=1}^{c}X_{i_t}$$

其中 $X_{i_t}=\frac{1}{cp_{i_t}}\mathbf{A}_{ki_t}\mathbf{B}_{i_tl}$。而

$$\mathbb{E}(X_t)=\sum_{i=1}^{n}p_i\frac{1}{cp_i}\mathbf{A}_{ki}\mathbf{B}_{il}=[\mathbf{AB}]_{kl}/c$$

故

$$\mathbb{E}(\mathbf{F}_{kl})=[\mathbf{AB}]_{kl}$$

进而 $\mathbb{E}(\mathbf{F})=\mathbf{AB}$。 □

产生多个 $\mathbf{F}$ 的样本并求平均，即得到矩阵乘积 $\mathbf{AB}$ 的估计值。

$\mathbf{F}$ 的方差随预先给定的分布列 $\mathcal{P}$ 变化。可以进一步证明，当 $\mathcal{P}$ 的值取得较为合理时，估计误差的量级大致为 $\epsilon=\|\mathbf{AB}-\mathbf{F}\|_F=\mathcal{O}(\|\mathbf{A}\|_F\|\mathbf{B}\|_F/c)$。

一个更加简单的方法是，在以上方法中取 c=1。基于类似的证明过程，构建随机变量 $\mathbf{X}=p_{i_t}\mathbf{A}_{i_t}\mathbf{B}_{i_t}^T$，则 $\mathbb{E}(\mathbf{X})=\mathbf{AB}$，多次采样求平均同样可以估计 $\mathbf{AB}$ [4]。

在第一种方法中，每次采样随机变量 $\mathbf{F}$ 的复杂度较高，但方差较少，因此需要的采样次数较少；第二种方法中，采样复杂度较低，但需要大量样本才能得到可信的估计值。然而，对这两种方法，若要使以 Frobenius 范数计算的估计精度小于 ϵ，计算复杂度均正比于 $1/\epsilon^2$，这意味着很难通过随机方法计算矩阵相乘的精确结果。

VI. 总结

本文中，我们简述了 RandNLA 实现数据降维，从而降低矩阵计算复杂度的原理，针对最小二乘、低秩逼近和矩阵乘法等具体问题，对相应算法的复杂度和精确度进行了分析，在最小二乘和低秩逼近领域，也介绍了其在深度学习领域的部分最新应用。在精确度要求较低的场合，通过数据降维，RandNLA 可以明显降低矩阵计算的复杂度；同时，大部分 RandNLA 算法中精确度与采样规模成平方反比关系，这使得 RandNLA 难以完成非常精确的计算，可能需要考虑采用残差迭代或蒙特卡洛等方法减小计算误差。

参考文献

[1] P. Drineas, M. W. Mahoney and S. Muthukrishnan, "Sampling algorithms for regression and applications," in Proc. of the 17th annual ACM-SIAM symp. on Discrete algorithm, 2006, pp. 1127-1136.

[2] P. Drineas and M. W. Mahoney, "RandNLA: Randomized numerical linear algebra," Commun. of the ACM, vol. 59, no. 6, pp. 80-90, 2016.

[3] P. Drineas, R. Kannanand M. W. Mahoney, "Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication," SIAM J.on Computing, vol. 36, no. 1, pp. 132-157, 2006.

[4] P. G. Martinsson and J. A. Tropp, "Randomized numerical linear algebra: Foundations & algorithms," arXiv:2002.01387v3 [math.NA],15 Mar 20213. https://arxiv.org/pdf/2002.01387.

[5] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," Foundations and Trends in Theoretical Computer Science, vol. 10, no. 1-2, pp. 1-157, 2014.

[6] R.Murray, et al., "Randomized numerical linear algebra: A perspective on the field with an eye to software," arXiv preprint arXiv:2302.11474, 2023.

[7] M. W. Mahoney, "Randomized algorithms for matrices and data," Foundations and Trends in Machine Learning, vol. 3, no. 2, pp. 123-224, 2010.

[8] C. Subramani, et al., "Enhancing deep stochastic configuration networks Efficient training via low-rank matrix approximation," Information Sciences, vol. 690, no. 121519, 2025.

[9] N. Halko, P. G. Martinsson and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," SIAM review, vol. 53, no. 2, pp. 217–288, 2011.

[10] C.O.Puiu, Natural Gradient Algorithms for Training Deep Neural Networks, Ph. D. thesis, University of Oxford, 2023.