arXiv:2305.03099v1 [cs.LG] 4 May 2023

一种用于快速监督学习的自助法算法

Michael A Kouritzin[1], Stephen Styles[2] 和 Beatrice-Helen Vritsiou[1]

[1]阿尔伯塔大学数学与统计科学系[2]加拿大统计局，ESMD/DMSE

摘要

神经网络（NN）的训练通常依赖某种曲线跟踪方法，例如梯度下降（GD）（以及随机梯度下降（SGD））、ADADELTA、ADAM或有限内存算法。这些算法的收敛通常需要大量观测值，才能达到较高的准确率；对于某些函数类，这些算法可能需要经过多个数据点训练轮次才能逐渐奏效。本文探讨了一种有望显著提高收敛速度的新技术，尤其适用于浅层网络：它不采用曲线跟踪，而是通过自助法、重采样和线性回归来解耦隐藏层并更新其加权连接。通过利用重采样的观测值，实验证明这一过程收敛速度极快，且所需数据点更少：具体而言，我们的实验表明，与传统神经网络训练方法相比，只需使用一小部分观测值，就能近似各种函数类。

1 引言

近几十年来，神经网络已被证明是解决模式识别类问题的出色工具，例如图像分类、手写分析、股票市场预测等（参见例如 [15], [5]）。针对特定应用构建合适的神经网络，首先需要选择适当的架构（包括层数以及每层的节点数），这可以依据经过验证的实践经验和理论来完成；然后使用带标签观测值/历史数据对其进行训练。最常见的做法是在训练阶段使用某种梯度下降（GD）来更新网络的（权重和偏置）参数。通常，人们希望频繁更新参数，每次只使用一部分训练数据，这便产生了批量梯度下降（BGD）或随机梯度下降（SGD）（当批次中的数据点很少，有时甚至只有一个数据点时）。此外，研究发现，SGD 方法的自适应变体有助于收敛：它们既能加快梯度搜索速度，也能确保搜索不会陷入参数空间的“鞍点”，或在最小值处过度越界。AdaGrad [4, 13], RMSProp [16], ADADELTA [17], ADAM [7], NADAM [3] 等自适应方法共同表明，SGD 可以更好地适应所处理的数据；如今，在大多数（但并非所有）应用中，它们已被视为首选方法，并且能够很好地应对许多不同的情形（例如梯度稀疏或参数数量极大）。

Kingma 和 Ba 提出的自适应矩估计算法（ADAM） [7], 是 SGD 最常用的改进算法之一，它结合了此前其他变体中的若干思路。从经验上看，它能够带来更快的收敛速度，而且无论不同问题的数据特征有何差异，或选择何种神经网络（NN）架构，都能表现良好。即便如此，使用 ADAM 时，仍可能需要进行许多次迭代，才能获得达到所需准确率水平的权重参数估计值。这或许是因为 ADAM 以及上文列出的其他算法都属于沿曲线搜索的算法，而沿曲线追踪至极值点（尤其是不使用高阶导数时）可能是一个相对缓慢的过程。另一方面，如果已知感兴趣的函数（在我们的设定中，通常是神经网络使用当前权重时预测的输出之间的均方误差（MSE）

并且正确输出）至少二次可微，那么我们可能可以使用牛顿法的改进方法（二阶方法）来加快收敛。实际上，由于与梯度方法相比，这些方法的内存需求和计算复杂度明显更高，而且神经网络本身就需要大量参数，因此通常会选择拟牛顿方法，例如有限内存 Broyden–Fletcher–Goldfarb–Shanno 算法（L-BFGS） [12], 。值得注意的是，在神经网络最标准的应用范围内，上述优化器中没有哪一种能明显优于所有其他算法，因此通常也会根据具体问题选择合适的方法。

在本文中，我们提出了一种截然不同的神经网络训练技术，它既不追踪曲线，也不需要了解导数：这是因为它省去了神经网络训练中的常规反向传播步骤。这种新方法称为自举学习算法（BLA），从目前对其进行测试的结果来看，至少对于单隐层神经网络（SHLNN），效果十分出色。我们认为，当可用的训练数据较少，或需要在极少的迭代内达到较高的准确率时，该方法会非常有用。

目前对宽型神经网络的浓厚兴趣（即隐藏层较少，有时只有一层，但每个此类层中可能包含许多节点）得到了若干理论结果的支持，这些结果建立在 Cybenko [1]和 Hornik [6]提出的开创性通用逼近结果之上并对其进行了改进：这些结果最基本的形式表明，任何输入来自 $\mathbb{R}^d$ 的紧子集、取值在 $\mathbb{R}^m$ 中的良态函数，都可以由足够宽的 SHLNN 以任意精度进行逼近。基于这一原因以及便于理解的考虑，我们首先聚焦于 SHLNN 情形，但随后将展示如何将该算法扩展到包含更多隐藏层的情形。

所提出的算法在某种意义上更接近 BGD 或梯度下降（GD），而非 SGD，因为它使用了规模较大的批次。然而，这里提出的方法并未采用梯度计算/近似，而是使用一种解耦的参数更新方案，该方案依赖于一种受 [9] 和 [11] 中数学结果启发的线性近似方法。为了说明这种方法，首先假设 $\{x_t\colon t=1,2,\dots\}$ 和 $\{y_t\colon t=1,2,\dots\}$ 分别是取值于 $\mathbb{R}^d$ 和 $\mathbb{R}$ 的随机过程，并且满足

$$y_t = x_t^T w + \epsilon_t, \quad t=1,2,\dots, \tag{1}$$

其中， w 是一个未知的 d维“权重”向量，而 ϵ_t 是一个随机噪声序列。我们可以将向量 x_t 视为一个仅有一组连接（即没有隐藏层）的“退化”神经网络的输入，将 y_t 视为其中一个输出节点的值，并将 w视为与该节点对应、需要准确确定的权重/连接向量。鉴于此，尝试最小化均方误差是合理的。

$$w \;\mapsto\; \lim_{N\to\infty} \frac{1}{N}\sum_{t=1}^{N} \mathbb{E}|y_t - x_t^T w|^2$$

（暂且假设输出序列 $(y_t)_t$ 大致正确）。在一些足够一般的条件下（例如，假设期望 $A := \lim_{N\to\infty} \frac{1}{N}\sum_{t=1}^{} \mathbb{E}(x_t x_t^T)$ 和$Nb := \lim_{N\to\infty} \frac{1}{N}\sum_{t=1}^{} \mathbb{E}(y_t x_t)$ 存在，且 A 为正定），存在唯一的最优 w，其表达式为 $w_0 := A^{-1}b$。然而，如果难以计算或近似其极限 A 和b，我们仍然可以通过运行以下线性近似算法，找到 w 的可用估计值。该算法从某个初始猜测 A_0, b_0 和某个初始权重 $r \geq 0$开始：令

$$\check{A}_0 = A_0, \qquad \check{A}_{n+1} = \check{A}_n + \frac{1}{n+1+r}(\underbrace{x_{n+1}x_{n+1}^T}_{A_{n+1}} - \check{A}_n), \qquad n=0,...,N-1 \tag{2}$$

$$\check{b}_0 = b_0, \qquad \check{b}_{n+1} = \check{b}_n + \frac{1}{n+1+r}(\underbrace{y_{n+1}x_{n+1}}_{b_{n+1}} - \check{b}_n), \qquad n=0,...,N-1, \tag{3}$$

然后 fo　　取某个较小的正常数，或一个缓慢递减的增益序列 μ_t，持续设置　　g

$$w_{t+1} = w_t + \mu_t(\check{b}_N - \check{A}_N w_t)$$

直到 w_t 基本收敛。简而言之，我们计算加权平均

$$\check{A}_N = \frac{r}{r+N}A_0 + \frac{1}{r+N}\sum_{n=1}^{N} A_n, \quad \check{b}_N = \frac{r}{r+N}b_0 + \frac{1}{r+N}\sum_{n=1}^{N} b_n \tag{4}$$

对于某个较大的 N，然后以数值稳定的方式求解 $w_N := \check{}A_N^{-1}\check{}b_N$。

备注。为明确/澄清下文将使用的记号：我们也将 y 视为一个向量，即神经网络某一层所有节点值组成的向量；在这种情况下，式 (1) 将替换为 $y_t^T = x_t^T w + \epsilon_t^T$，其中 $w \in \mathbb{R}^{d\times m}$ 现在是矩阵值的（而 ϵ_t 是一个随机"噪声"向量）。此时，式 (3) 中的 b_{n+1} 将改为定义为 $x_{n+1}y_{n+1}^T$，而 $\check{}\, b_N$ 将是 $\mathbb{R}^{d\times m}$值的。

除非另有说明，我们在实验中将选择 $r = N$，这意味着

$$\check{A}_N = \frac{1}{2}A_0 + \frac{1}{2N}\sum_{n=1}^{N} A_n, \quad \check{b}_N = \frac{1}{2}b_0 + \frac{1}{2N}\sum_{n=1}^{N} b_n. \tag{5}$$

换句话说，初始估计/"猜测"（它将从算法中使用的上一个数据批次沿用而来）将与仅根据当前批次数据得出的估计赋予相同的权重。这样做旨在产生加权平均，使其以几何速率 1 2 忘却较早的估计。

–

迄今为止，我们完全忽略了一个关键事实：要近似一般的函数类，我们需要使用至少有一个隐藏层的神经网络，并且如式 (1) 所示，'内部'节点对输入的依赖不能是线性的。相反，还要应用激活函数（有时每一层使用不同的激活函数），使这种依赖变为非线性。为了将这一事实与上述线性近似方案结合起来，我们在此提出将神经网络'解耦'，并针对每一对相邻层同时应用线性近似过程。对于 SHLNN，第一个过程用于估计输入层与隐藏层之间的突触权重，第二个过程则用于估计隐藏层与最终输出层之间的权重。自然，这会带来一个新问题：我们必须确保这两个过程的结果始终兼容。这引出了所提方法中或许最关键、最具创新性的特点：以两种不同的方式利用训练数据——既用于更新，也用于跟踪算法在任一给定时刻生成的两组权重之间的依赖关系（如何实现这一点的细节将在下一个小节中简要介绍，并在第 2 节中完整阐述）。

记号。我们研究只有一个隐藏层的全连接神经网络（在实验中，我们选择在这一层设置约 $m = 100$ 个节点，但在今后的应用中可以根据需要进行调整）。为简单起见，我们只考虑标量输出，因为可以将多维输出视为多个独立函数，并对每个函数应用我们的方法。我们重点研究回归问题，并采用下图所示的网络架构。

我们将从输入到输出对层进行计数，并使用上标和下标来区分层以及层内相应的节点（或根据上下文区分时间）。 β 将用于表示偏置（以区别于上文引入的线性近似算法中使用的向量（矩阵）b），不过在大多数情况下，我们会将其作为权重向量 w 的 w_0分量包含在内（然后我们还会设置 $x_0 = -1$，使标量积记号 $w^T x$ 包含减去偏置的操作；根据上下文应该可以明确每次偏置项是否包含在向量（矩阵） w 中）。使用向量记号、下图以及其中所隐含的权重和激活函数 σ_1， σ_2 （它们按分量作用），可以得到

$$\widehat{y} = \sigma_2(w^2\sigma_1(w^1 x - \beta^1) - \beta^2), \tag{6}$$

根据通用逼近定理，该函数具有足够的通用性，能够在 $\mathbb{R}^d$ 的任意紧子集上逼近任意连续标量函数。现在将 w 中的偏置吸收进去，我们可以写成

$$\widehat{y} = \sigma_2(w^2 h), \quad h = \sigma_1(w^1 x), \tag{7}$$

在后续内容中，跟踪每一层的预激活值也至关重要：

$$z^2 = w^2 h, \quad z^1 = w^1 x. \tag{8}$$

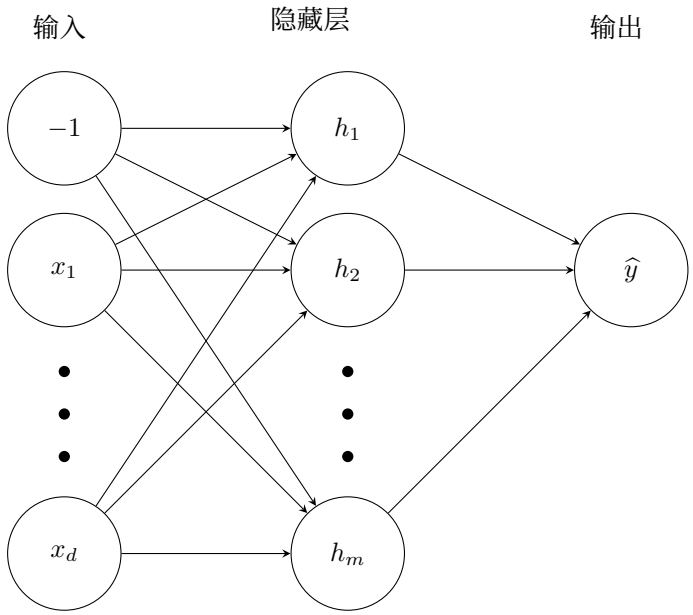


图 1：神经网络架构

1.1 自举粒子

暂且假设我们能够获得（准确求得的）隐藏值 h。（当然，我们的工作并未作此假设。）那么，内部变量 h_n （对应于函数 f 在给定输入 x_n 时的取值；我们想要近似该函数，其中 $f(x_n) = y_n$ 是输入 x_n 时的实际输出）可以视为第 2 层的输入，而它们的预激活值 z_n^1 （当激活函数 σ_1 恰好可逆时，该值将等于 $\sigma_1^{-1}(h_n)$）则成为第 1 层的输出。请注意， x_n （神经网络的实际输入）与 z_n^1 满足线性关系，我们希望用上文介绍的线性近似算法对其进行优化。类似地， h_n 与 z_n^2 也满足另一种线性关系（如果存在 σ_2^{-1}，后者的值就等于 $\sigma_2^{-1}(y_n)$；在我们的实验中，主要将 σ_2 设为恒等函数，这也是线性回归中的标准选择）。这样，我们便将 SHLNN 解耦为两个“退化”的神经网络，从而可以应用前文的线性近似方案：

$$\text{Layer 1 to 2:} \quad \begin{aligned} A_n^1 &= x_n {x_n}^T \\ b_n^1 &= x_n (z_n^1)^T \end{aligned}$$

$$\text{Layer 2 to 3:} \quad \begin{aligned} A_n^2 &= h_n {h_n}^T \\ b_n^2 &= h_n (\sigma_2^{-1}(y_n))^T \end{aligned} .$$

然而，这预设我们能够获得对 h_n 和z_n^1至少相当准确的近似值，但实际上我们无法保证这一点。

我们通过创建自举粒子来解决这个问题。给定当前的权重（和偏置）值，对每个输入（来自一个批次的

大小为 N 的数据）运行网络，并跟踪内部值和最终输出，即可创建自举粒子：令 $\{(x_n, z_n^1, h_n, z_n^2, \hat{y}_n)\}_{n=1}^N$ 表示通过这种方式得到的五元组／粒子（注意，通常情况下，$\hat{y}_n \neq y_n$是函数 f 对输入 x_n的实际输出）。为使记号清晰，我们将粒子$\{(x_n, z_n^1, h_n, z_n^2, \hat{y}_n)\}_{n=1}^N$重新标记为 $\{(\hat{x}_k, \hat{z}_k^1, \hat{h}_k, \hat{z}_k^2, \hat{y}_k)\}_{k=1}^N$：这些粒子将帮助我们选取合适的内部值 $\hat{h}_{s_n}$ 和 $\hat{z}_{s_n}^1$，再将它们与真实数据 $\{(x_n, y_n)\}_{n=1}^N$ 结合，以应用线性近似方案。具体而言，对于每个 (x_n, y_n)，我们选取距离最近的 δ 个粒子（即 $\ell_{n\to k} = \|(x_n, y_n) - (\hat{x}_k, \hat{y}_k)\|$, $k \in \{1, ..., N\}$ 中最小的 δ个距离），并使用一个当 (x_n, y_n) 接近 $(\hat{x}_k, \hat{y}_k)$ 时取值较大的（评分）函数 s。按照该评分函数所给出的概率，某个自举粒子 $(\hat{x}_{k_j}, \hat{z}_{k_j}^1, \hat{h}_{k_j}, \hat{z}_{k_j}^2, \hat{y}_{k_j})$, $1 \le j \le \delta$ 会被选中，作为真实数据对 (x_n, y_n) 的匹配粒子，由此我们可以构造“混合”粒子

$$\left\{\left(x_n,\, \hat{z}_{k_{j_n}}^1,\, \hat{h}_{k_{j_n}},\, z_n^2 = \sigma_2^{-1}(y_n),\, y_n\right)\right\}_{n=1}^N.$$

^^备注。 1. 为了简化记号，从现在起，我们将用 h_n 代替 $h_{k_{j_n}}$（类似地，用 $\hat{z}_n^1$ 代替 $\hat{z}_{k_{j_n}}^1$）；特别地，记号 $\hat{h}$ 和 $\hat{z}$ 表示这些节点值来自针对真实数据对 (x_n, y_n) 抽取的自助法粒子，而通常并不等于将 x_n 输入神经网络（使用当前权重）时所得到的内部值。

2. 如前所述，在我们的实验中，我们令 σ_2 为恒等函数，因此可以安全地设置 $z_n^2 = \sigma_2^{-1}(y_n)$。另一方面，如果在某些应用中，我们必须使用最后一层的非可逆激活函数，则可以改为按如下方式选择 z_n^2：对于每个 $(x_n,\ y_n)$，我们可以按照上述相同的方式独立于第一个自助法粒子 k_{j_n}，再抽取第二个自助法粒子（粒子 k_{u_n}），并设置 $z_n^2 = \hat{z}_n^2 = \hat{z}_{k_{u_n}}^2$。

现在，对于 SHLNN 的每一对连续层，我们可以放心地应用前文所述的线性近似方案:

$$\text{Layer 1 to 2:}\quad \begin{array}{rcl} A_n^1 & = & x_n x_n^T \\ b_n^1 & = & x_n(\hat{z}_n^1)^T \end{array} \quad \rightsquigarrow \quad w^1 = \left[A_N^1\right]^{-1} b_N^1$$

$$\text{Layer 2 to 3:}\quad \begin{array}{rcl} A_n^2 & = & \hat{h}_n \hat{h}_n^T \\ b_n^2 & = & \hat{h}_n(z_n^2)^T \end{array} \quad \rightsquigarrow \quad w^2 = \left[A_N^2\right]^{-1} b_N^2.$$

关于方法论的备注。我们主要使用平方 l_2 距离：$\ell_{n\to k} = |x_n - \hat{x}_k|^2 + |y_n - \hat{y}_k|^2$，以及评分函数 $s(l) = e^{-l^2}$。在实验中，我们首先将其设为 $\delta = \delta_m = 40$，随着已处理的数据批次数量 m 增加，逐步将其降至 8（当 δ 达到 8 后，我们将其保持不变）。

原则上，我们预计允许 δ 随着训练的进行而降低（甚至一直降至 1）可以提升性能，同时也有助于建立理论结果。然而，在本工作中，我们没有探究应选择哪种递减函数 $m \to \delta_m$ 才是最佳方案。

本文其余部分的组织如下：第 2 节详细介绍该方法（以及一些理论方面的考量），第 3 节和第 4 节则描述我们的实验，并给出与其他方法的比较（我们比较的方法包括梯度下降（ GD ）、ADAM 和有限内存 Broyden–Fletcher–Goldfarb–Shanno 算法（L-BFGS））。

致谢。 本研究基于第二位作者在阿尔伯塔大学攻读硕士学位期间及之后担任研究助理时提出的一些想法。

2 方法

如前所述，本文主要关注仅包含一个隐藏层的网络。对于这种架构，我们需要选择两个激活函数 σ_1 和 σ_2，其中 σ_1 是

输入层与隐藏层之间的激活函数，而 σ_2 是隐藏层与输出层之间的激活函数。通常，对于回归，会选择 σ_2 作为恒等函数，这与通用逼近定理一致。在本文中，我们主要将双曲正切用作第一个激活函数 σ_1。不过，在更合适的情况下，也会使用 ReLU 和 Leaky ReLU。

我们考虑一组包含 $\mathcal{N}$ 个输入-输出观测值的训练数据，可以（并且打算）重复使用多次，每次遍历数据称为一个训练轮次。每个训练轮次 e 将被拆分成一个或多个（互不重叠的）批次，使用从低到高连续的数字标记为 $M_l(e)\ldots, M_h(e)$，因此 $M_l(1) = 1$，且对于 $e \geq 1$，$M_l(e+1) = M_h(e) + 1$。批次 m 的大小记为 $N_m \leq \mathcal{N}$；若无歧义，也可简记为 N。其大小可以因训练轮次而异（甚至同一训练轮次内不同批次的大小也可能不同）。

以下是该方法对每个数据批次所执行的步骤。

权重初始化：为了初始化算法，我们首先为网络生成随机权重。实现方式有很多，但我们只采用一种标准方法：生成相互独立的权重，其值来自以零为中心、方差较小的正态分布（在我们的例子中，方差为 0.5）。关键在于不要将权重设为零。

自举粒子提议：在每个批次开始时，批次中的每个输入都会被前向传播到网络中，我们会跟踪这些输入、隐藏节点值以及估计输出。我们用 $\{(\hat{}x_i,\ \hat{}z_i^1,\ \hat{}h_i,\ \hat{}z_i^2,\ \hat{}y_i)\}_{i=1}^N$ 表示这些值。当然，它们取决于网络当前的权重（和偏置）。

评分：每次处理一个数据点（取自当前批次），我们使用范数函数，将其输入值与真实输出值组成的数对（$x,\ y$）与所有（$\hat{}x_i,\ \hat{}z_i^1,\ \hat{}h_i,\ \hat{}z_i^2,\ \hat{}y_i$）、$1 \leq i \leq N$ 中的对应分量进行比较。（对于本文，我们对单输入网络使用（平方） ℓ_2 范数，对多个输入使用 ℓ_∞ 范数。）然后，我们将该范数距离输入评分函数 s，该函数为范数较小的单元分配更高的分数。

概率分布构建：对于每个真实数据点，我们选择得分 δ 最高的自举粒子提议 $\{(\hat{}x_i,\ \hat{}z_i^1,\ \hat{}h_i,\ \hat{}z_i^2,\ \hat{}y_i)\}_{i=1}^{\delta}$，并使用得分作为相对权重来构建概率分布。请注意，每个真实数据点都会被分配一个专属的概率分布。

自助法粒子接受： 现在，我们从与实际输入-输出对 (x_n, y_n) 对应的每个概率分布中采样 $1 \leq n \leq N$，并记录采样所得自助法粒子的**内部值**，再将这些值与数值 x_n 和 y_n组合起来，为下文的回归计算构造 (x_n, y_n) 的新代理数据点。更准确地说，我们用“混合”粒子替换 (x_n, y_n)

$$(x_n, \hat{z}_n^1, \hat{h}_n, \sigma_2^{-1}(y_n), y_n),$$

其中，$\hat{}z_n^1$ 和$\hat{}\ h_n$ 是所采样的自助法粒子对应的内部值。（还要注意，此处的 $\sigma_2 =$ 是恒等函数，因此 $\sigma_2^{-1}(y_n) = y_n$。）由于此前已完成范数归一化和评分，我们创建了一个代理数据点，其内部变量与实际输入输出数据（$x_n,\ y_n$）高度一致的可能性很大。

回归权重更新：现在我们有了代理值 $\{(x_n,\ \hat{}z_n^1,\ \hat{}h_n,\ z_n^2,\ y_n)\}_{n=1}^N$，便可以使用第 1 节中的批次方法更新权重和偏置。也就是说，我们使用当前批次的代理数据点更新 $\check{}\check{}A_n$ 和 b_n，然后求解权重（见下方方程）。

算法批次模式自助助推学习

For epoch $e = 1$ to E /* w_1^1 and w_1^2 were already randomly set */

For batch $m = M_l(e)$ to $M_u(e)$

Let $N = N_m$ be size of batch m.

For $n = 1$ to N_m

$\hat{z}_n^1 = w^1(m)\hat{x}_n - \beta^1$, $\hat{h}_n = \sigma_1(\hat{z}_n^1)$ /* Feed x forward to determine Bootstrap proposals */

$\hat{z}_n^2 = w^2(m)\hat{h}_n - \beta^2$, $\widehat{y}_n = \sigma_2(\hat{z}_n^2)$

For $n = 1$ to N_m /* Find Bootstrap target particles */

For $i = 1$ to N_m do /* Find distances from each datum */

$\ell_{n\to i} = ||(x_n, y_n) - (x_i, \widehat{y}_i)||$

Sample and create $(x_n, \hat{\mathbf{z}}_\mathbf{n}^\mathbf{1}, \hat{\mathbf{h}}_\mathbf{n}, z_n^2, y_n)$ with $\{s(\ell_{n\to i})\}_{i=1}^{N_m}$ as weights for $\{(\hat{x}_i, \hat{z}_i^1, \hat{h}_i, \hat{z}_i^2, \hat{y}_i)\}_{i=1}^{N_m}$

If $e = 1$ then $r = 0$, else $r = N_m$ and set

$$\check{A}_N^1(m) = \tfrac{r}{r+N_m}\check{A}_N^1(m-1) + \tfrac{1}{r+N_m}\sum_{n=1}^{N_m} x_n x_n^T,$$

$$\check{b}_N^1(m) = \tfrac{r}{r+N_m}\check{b}_N^1(m-1) + \tfrac{1}{r+N_m}\sum_{n=1}^{N_m} x_n(\hat{z}_n^1)^T,$$

$$\check{A}_N^2(m) = \tfrac{r}{r+N_m}\check{A}_N^2(m-1) + \tfrac{1}{r+N_m}\sum_{n=1}^{N_m} \hat{h}_n\hat{h}_n^T,$$

$$\check{b}_N^2(m) = \tfrac{r}{r+N_m}\check{b}_N^2(m-1) + \tfrac{1}{r+N_m}\sum_{n=1}^{N_m} \hat{h}_n(z_n^2)^T$$

Set $\mu_t^1 = 2/[\max\{\lambda(\check{A}_N^1(m))\} + \min\{\lambda(\check{A}_N^1(m))\}]$

Set $\mu_t^2 = 2/[\max\{\lambda(\check{A}_N^2(m))\} + \min\{\lambda(\check{A}_N^2(m))\}]$

/* Get weights and biases for new epoch/batch within epoch */

Set $t = 0$ and $w^i(m) = \begin{cases} 0 & m = 1 \\ w^i(m-1) & m > 1 \end{cases}$ for $i = 1, 2$

While still converging do

$w^1(m) = w^1(m) + \mu_t^1(\check{b}_N^1(m) - \check{A}_N^1(m)w^1(m))$

$w^2(m) = w^2(m) + \mu_t^2(\check{b}_N^2(m) - \check{A}_N^2(m)w^2(m))$

Shuffle the data for the next epoch.

^^备注。 1. 大多数量都隐式地依赖于批次编号 m，例如 $h_n = h_n(m)$ 和 $\delta = \delta_m$；或依赖于迭代编号 t，例如 $w^1(m) = w_t^1(m)$。不过，为避免算法中的记号过于复杂，大多数这类依赖关系都被省略了。

2. 应当注意（另请参见下面的引理），为保证 $w^i(m)$ $i = 1,2$ 收敛（其极限等于 $[\check{}A_N^i(m)]^{-1}\check{}b_N^i(m)$），我们需要确保$I - \mu_t^2\check{}A_N^i(m)$的特征值位于单位球内。这等价于以下两个条件：

$$\mu_t^i < \frac{2}{\max\{\lambda(\check{A}_N^i(m))\}} \quad \text{and} \quad \mu_t^i \min\{\lambda(\check{A}_N^i(m))\} > 0$$

但是，如果 $\mu_t^i > 0$ and $\check{}\, A_N^i(m)$ 非奇异，则第二个条件始终成立（只要每个数据批次的规模足够大，后者就以很高的概率成立）。不过，为使极端特征值位于单位圆盘的中心（并且根据实验观察，采用下述选择可获得更快的收敛速度），实际中我们取

$$\mu_t^i = 1.95/[\max\{\lambda(\check{A}_N^i(m))\} + \min\{\lambda(\check{A}_N^i(m))\}].$$

还要注意，我们在实验中运行的所有线性过程似乎都收敛得相对较快，即使在我们观察到矩阵 $\check{}\, A_N^i(m)$ 条件不佳的情况下也是如此。从理论角度来看，这一点也值得进一步探究。

目前，鉴于我们的矩阵规模最大仅为 100×100（该规模由根据隐藏节点构建的矩阵达到），我们可以使用默认 R 软件包高效地计算 $\lambda(\check{A}_N^i)$，但对于更大的数据集（即更多隐藏节点），应探索更复杂的方法。或者，也可以将 $\mu_t^1(t) \searrow 0$ 设置得较慢，以稍微降低收敛速度为代价（另请参见以下引理）。例如，我们可以取 $\mu_t^i(t) = \frac{1}{\ln(1+t)}$。

现在，我们为每个训练轮次最后一步中“同时仍在收敛”这一部分提供一些理论依据。

引理 1。假设 $[I - \mu_t^1 \check{A}_N^1(m)]$ 和 $[I - \mu_t^2 \check{A}_N^2(m)]$ 的所有特征值都（最终）严格位于单位球内，并且 μ_t^1、$\mu_t^2 > 0$ 对于 $i = 1, 2$ 满足 $\sum \mu_t^i = \infty$。那么，这些直线$t=1$

当仍在收敛时执行
$$w_{t+1}^1(m) = w_t^1(m) + \mu_t^1(\check{b}_N^1(m) - \check{A}_N^1(m) w_t^1(m))$$
$$w_{t+1}^2(m) = w_t^2(m) + \mu_t^2(\check{b}_N^2(m) - \check{A}_N^2(m) w_t^2(m))$$
真正求解 $w^1(m) = \left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m)$ 和 $w^2(m) = \left[\check{A}_N^2(m)\right]^{-1} \check{b}_N^2(m)$ 作为 $t \to \infty$。

证明。只需考虑第一个方程，因为第二个方程的证明完全相同。于是有

$$w_{t+1}^1(m) = w_t^1(m) + \mu_t^1(\check{b}_N^1(m) - \check{A}_N^1(m) w_t^1(m)). \tag{9}$$

因此，如果从等式两边减去 $\left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m)$ 并进行化简，我们得到

$$\begin{aligned} w_{t+1}^1(m) - \left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m) &= w_t^1(m) - \left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m) \\ &\quad + \mu_t^1 \check{A}_N^1(m)(\left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m) - w_t^1(m)) \\ &= (I - \mu_t^1 \check{A}_N^1(m)) \left(w_t^1(m) - \left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m) \right). \end{aligned} \tag{10}$$

现在，利用递推以及标准方法（见 [8], [9], [11]），我们可以验证

$$\begin{aligned} \|w_{t+1}^1(m) - \left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m)\| &= \left\| \prod_{k=1}^{t} (I - \mu_k^1 \check{A}_N^1(m)) \left(w_1^1(m) - \left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m) \right) \right\| \\ &\leq \prod_{k=1}^{t} \|I - \mu_k^1 \check{A}_N^1(m)\| \cdot \left\| w_1^1(m) - \left[\check{A}_N^1(m)\right]^{-1} \check{b}_N^1(m) \right\| \\ &\leq C \prod_{k=1}^{t} \exp\left(-\mu_k^1 \lambda_m \vee (\mu_k^1 \lambda_M - 2)\right) \\ &\longrightarrow 0 \text{ as } t \to \infty. \end{aligned}$$

这里，$C > 0$ 是某个绝对常数，而 λ_m，λ_M 分别是 $\check{A}_N^1(m)$ 的最小和最大特征值。

□

备注。1. 上述证明还表明，当 μ_t^1，μ_t^2在（t中）保持不变时，收敛速度呈几何级数增长。

2. 由于该引理，在实践中，我们可以将伪代码中的“while still converging”一行替换为关于 $w^i(m)$ 连续两次更新之间的距离小于某个预先选定容差的条件。为便于说明，在我们的实验中，对于批次 m 对应的每个线性过程，我们只运行固定次数的迭代（100，000 次迭代）。

训练一个包含两个隐藏层的神经网络

增加另一个隐藏层时，我们只需要做一个关键改动。现在有两种形式为 h^i 的隐藏变量：h^1 更靠近输入层，h^2更靠近输出层。此外，还有三种（未知的）预激活变量类型z^1，z^2，{v10>；当激活函数可逆时，我们也可以分别将它们写作 $\sigma_1^{-1}(h^1)$ $\sigma_2^{-1}(h^2)$ $\sigma_3^{-1}(\hat{y})$，其中激活

函数 σ_1，σ_2，σ_3 可逆。对于该算法，假设我们选取一个可逆的 σ_3，则应设置 $z_n^3 = \sigma_3^{-1}(y_n)$，其中 y_n 是输入 x_n时的实际输出。对于其他内部变量，现在我们独立地抽取两个自助法粒子

$$\left(\hat{x}_{k_{j_n}}, \hat{z}^1_{k_{j_n}}, \hat{h}^1_{k_{j_n}}, \hat{z}^2_{k_{j_n}}, \hat{h}^2_{k_{j_n}}, \hat{z}^3_{k_{j_n}}, \hat{y}_{k_{j_n}}\right) \quad \text{and} \quad \left(\hat{x}_{k_{u_n}}, \hat{z}^1_{k_{u_n}}, \hat{h}^1_{k_{u_n}}, \hat{z}^2_{k_{u_n}}, \hat{h}^2_{k_{u_n}}, \hat{z}^3_{k_{u_n}}, \hat{y}_{k_{u_n}}\right),$$

并创建“混合”粒子

$$\left(x_n,\ \hat{z}^1_{k_{j_n}}, \hat{h}^1_{k_{j_n}},\ \hat{z}^2_{k_{u_n}}, \hat{h}^2_{k_{u_n}},\ z^3_n,\ y_n\right),$$

我们用它来学习第一组、第二组和第三组突触权重，以替代直接将 x_n 输入神经网络（使用当前权重）时得到的内部值。换句话说，我们有：

$$\begin{array}{lrcl}
\text{Layer 1 to 2:} & A^1_n & = & x_n(x_n)^T \\
& b^1_n & = & x_n(\hat{z}^1_{k_{j_n}})^T \\
\text{Layer 2 to 3:} & A^2_n & = & \hat{h}^1_{k_{j_n}}(\hat{h}^1_{k_{j_n}})^T \\
& b^2_n & = & \hat{h}^1_{k_{j_n}}(\hat{z}^2_{k_{u_n}})^T \\
\text{Layer 3 to 4:} & A^3_n & = & \hat{h}^2_{k_{u_n}}(\hat{h}^2_{k_{u_n}})^T \\
& b^3_n & = & \hat{h}^2_{k_{u_n}}(z^3_n)^T
\end{array} .$$

除此之外，算法基本保持不变。

3 回归比较

鉴于权重是每次根据多个样本进行更新的，我们的方法可视为（批次）梯度下降的对应方法。然而，BLA 的优势在于它不会沿曲线前进；这似乎有助于该方法避免陷入“鞍点”，并在回归应用中取得远优于梯度下降的结果（正如我们将在实验中看到的那样），因此我们还将其与 ADAM 等在线实时更新方法进行了比较。事实上，BLA 似乎只需少量训练数据点就能达到较高的准确率，因此非常适用于在线实时更新。我们比较了本方法与另外三种流行算法的收敛速度：梯度下降（GD）、ADAM 和有限内存 Broyden–Fletcher–Goldfarb–Shanno 算法（L-BFGS）。

我们使用同一数据集训练这四种算法（包括 BLA），随后在同一个独立生成的验证集上计算均方误差。所有实验的训练集均包含 $\mathcal{N} = 6000$ 个数据点，验证集则包含 1000 个数据点。每种算法都测试了 1000 次，并对结果取平均，以避免任何离群值的影响。全部 6，000，000 个数据点均通过以下方式生成：每次都令 x 在相关函数 f 的定义域上服从均匀分布，且与先前生成的 x’ 相互独立，然后设定 $y = f(x)$。对于所有方法，我们都使用了 Python [14] 中的 scikit-learn 的 MLPRegressor 和 MLPClassifier 软件包。对于梯度下降，我们使用“SGD”选项，批次大小设为 6000。对于 ADAM 和 L-BFGS，则使用默认设置。

本文重点研究单隐藏层中含有 $m = 100$ 个节点的 SHLNN。我们始终取 $\sigma_2(x) = x$ 和 $\sigma_1(x) = \tanh(x)$；但下面关于随机网络的实验除外，该网络的图预计会有许多类似角点的区域，因此使用 $\sigma_1(x)$ =ReLU(x) 更为合适。

我们的目的并非针对每个问题优化 BLA，以期可能为其带来某些优势。相反，我们在实现 BLA 时做了几项相当随意的决定。我们始终

第一个批次采用 $r = 0$，其余批次采用 $r = N_m$，以便让先前的估计与新一轮迭代得出的结果具有相同权重。我们始终将第一个训练轮次分为 10 个大小相同的批次，第二个训练轮次分为 5 个批次，第三个分为 3 个，第四个分为 2 个，其余所有训练轮次都只包含 1 个批次。（实验结果和直觉表明，将一个训练轮次内的批次设为不同大小或许效果更好，但我们将此留待未来工作探索。）此外，为简单起见，我们始终在距离公式中使用（平方）ℓ_2 范数（多输入实验中使用 ℓ_∞ 范数），并将评分函数设为 $s(x) = \exp(-x^2)$。如前所述，对于每个实际数据点在采样时所考虑的粒子数量 $\delta = \delta_m$，我们从 $\delta_1 = 40$ 开始，此后每个批次都将其减 1，直到 δ_m 等于 8（之后保持不变）。

备注。 对于输入 x 或输出 y（或输入 x的某个组成部分）比其他值大几个数量级的函数，建议先将其缩放到相同的数值范围，以避免对应输入或输出之间的距离在数量级上远大于其他距离。我们的函数均不需要进行此类处理。

3.1 单输入单输出回归

在这些实验中，我们考虑对连续函数 $f : K \to \mathbf{R}$ 进行 SHLNN回归，该函数从紧子集 K 中接收单个输入，并输出一个实数。具体而言，我们首先在以下三个函数上比较这四种算法：

$$f_1(x) = x^3 - 2x^2 + 5x - 1, \text{ where } x \in [-3, 3] \tag{11}$$
$$f_2(x) = \sin(x^2) - 0.03x^5, \text{ where } x \in [-3, 3] \tag{12}$$
$$f_3(x) = -(x-2)^3(x+1)^2(x-4)/8, \text{ where } x \in [-1, 4]. \tag{13}$$

以下图显示了某次试验在训练期内的均方误差：

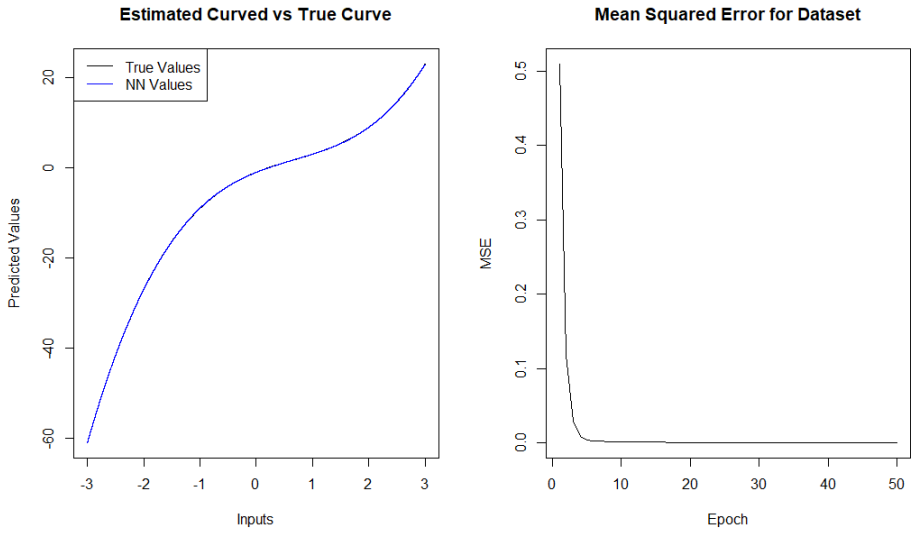


图2： 训练集的 $x^3 - 2x^2 + 5x - 1$ 近似值和均方误差，最多进行 50 个训练轮次。

四种算法在最多 50 个训练轮次内的平均均方误差比较，以及函数 f_1、f_2 和 f_3 的比较结果，分别见以下三个表：

	$x^3 - 2x^2 + 5x - 1$，其中 $x \in [-3, 3]$			
训练轮次	BLA	GD	ADAM	L-BFGS
1	0.1819	406.372	374.767	314.949
5	0.0099	359.012	202.644	73.127
10	0.0065	303.739	75.947	53.362
15	0.0050	251.660	58.347	45.512
25	0.0035	166.631	40.492	30.194
50	0.0019	87.064	15.262	11.001

	$\sin(x^2) - 0.03x^5$，其中 $x \in [-3, 3]$			
训练轮次	BLA	GD	ADAM	L-BFGS
1	0.1104	5.333	2.686	3.049
5	0.0886	4.884	2.022	2.036
10	0.0869	4.410	1.946	2.015
15	0.0857	4.016	1.800	1.987
25	0.0838	3.420	1.198	1.642
50	0.0805	2.616	0.785	1.053

	$-(x-2)^3(x+1)^2(x-4)/8$，其中 $x \in [-1, 4]$			
训练轮次	BLA	GD	ADAM	L-BFGS
1	0.060	5.997	3.822	4.271
5	0.038	5.740	1.670	1.938
10	0.035	5.468	1.493	1.696
15	0.033	5.240	1.241	1.607
25	0.028	4.875	0.832	1.157
50	0.022	4.247	0.467	0.654

在最后两种情况下，人们认为 BLA 可能正在接近最优下界（对于我们的 100 个隐藏状态神经网络），从而导致其改进速度放缓。

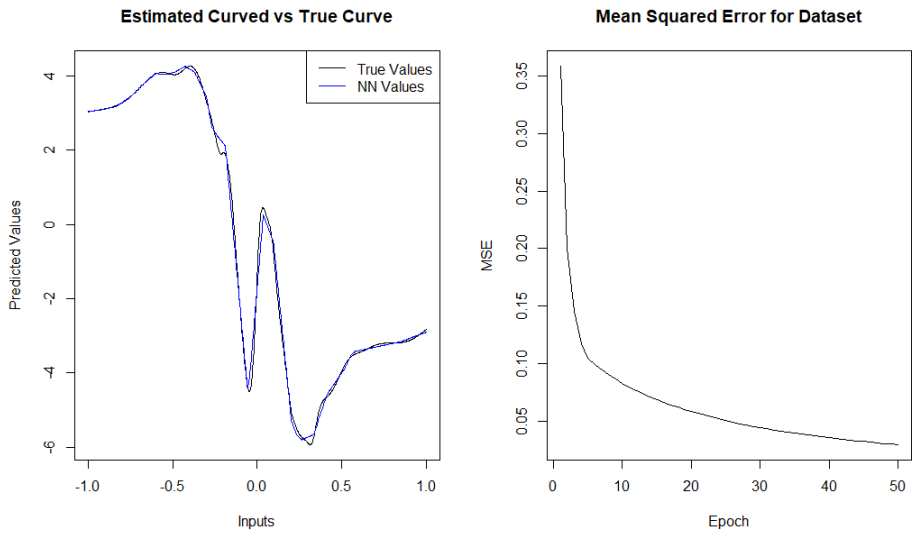
3.2 随机权重神经网络

前面的示例处理的是平滑函数，因此可能更容易逼近。为了处理复杂得多的函数，我们使用一个由神经网络本身定义的函数生成了一个数据集。具体来说，我们另外构建了一个单输入单输出神经网络（记作 NN_0 ），其架构与前面的示例相同（即包含一个有 $m = 100$ 个节点的隐藏层）。第一层的权重和偏置通过均值为 5、方差为 3 的正态随机变量赋值。类似地，第二层的权重和偏置通过均值为 0、方差为 0.5 的正态随机变量赋值。所有赋值彼此独立。

对于神经网络 NN_0，的第一个激活函数，我们选择了双曲正切激活函数，而第二个激活函数则是恒等函数。设置好 NN_0，后，我们从 $[-5, 5]$ 中均匀生成输入，并将其传入网络。由此得到的输入值和输出值对构成了训练集和验证集，我们用它们进行了实验。

现在，关于我们希望训练以近似 NN_0 （作为函数）的网络 NN_1 ，我们需要说明以下几点：由于通过神经网络（例如 NN_0, ）定义的上述函数，其图形看起来完全不平滑，因此，对于此处测试的所有方法而言，通常最好选择一个自身也具有拐角的激活函数用于 NN_1 。因此，我们没有像之前的实验中那样选择 tanh 作为第一个激活函数（有些奇怪的是，NN_0 本身的定义中也采用了 tanh），而是使用了 ReLU（NN_1 的第二个激活函数仍为恒等函数）。

下图展示了一个训练集示例。



Figu图 3：BLA 在随机生成的神经网络上运行 50 个训练轮次的示例。

均方误差的比较结果如下：

	随机网络			
训练轮次	BLA	GD	ADAM	L-BFGS
1	0.203	39.749	29.645	16.802
5	0.088	31.891	6.749	4.648
10	0.073	24.894	2.516	3.176
15	0.066	20.051	1.304	2.214
25	0.056	14.402	0.722	1.269
50	0.045	9.518	0.306	0.628

与先前的实验类似，BLA 能够迅速为这个随机生成的函数找到良好的估计值。同样，我们推测，BLA 收敛速度减慢的现象，是因为它正在接近理论下界。

3.3 多输入单输出

随着问题维度的增加，我们应预期需要更多数据才能获得合理的估计。我们也应预期，在选择 $r = N_m$ 和 $\delta_1 = 40$ 等（超）参数时，需要做出更多自适应选择。不过，为了说明 BLA 是一种稳健方法，我们希望在全文中使用一套通用参数。

我们在此近似的函数为：

$$f(x_1, x_2, x_3) = 2{x_1}^2 x_2 - 6x_1 x_3 \text{ where } x_1 \in [-5, 5],\ x_2 \in [-2, 2],\ x_3 \in [0, 4]$$

and BLA 均方误差在一个训练轮次后约为 200，并从此处迅速收敛：

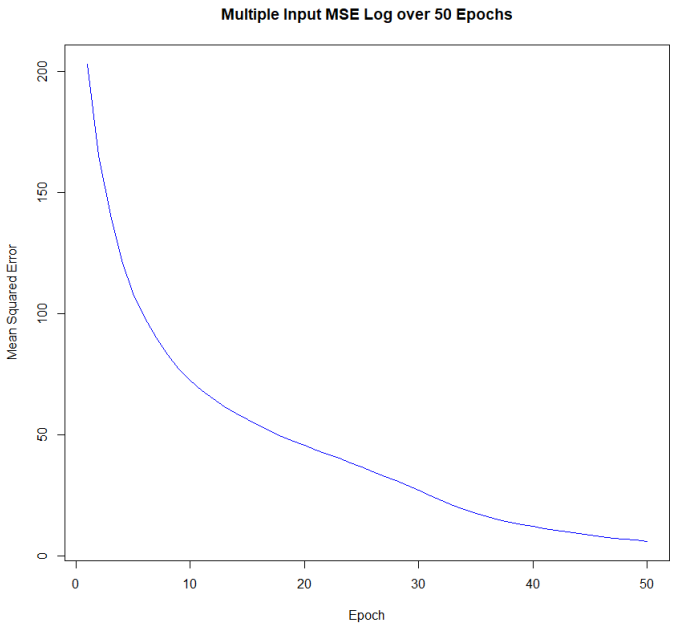


图 4：BLA 在三维输入数据集上的一次运行示例的均方误差。

与其他方法的比较如下：

	多输入			
训练轮次	BLA	GD	ADAM	L-BFGS
1	203.044	2185.198	2135.537	1987.212
5	108.205	1860.787	1641.746	873.113
10	72.536	1492.288	1153.364	732.618
15	56.370	1247.324	922.918	648.826
25	36.803	1053.292	671.781	529.465
50	6.123	840.075	463.275	390.943

显然，BLA 在这里并未提前“完成”。它相较于其他方法的优势似乎还在加速扩大，这可能是由于多输入问题的规模所致。

4 二元分类

我们曾指出，BLA 是一种监督学习方法，但迄今为止，我们只考虑了回归问题。因此，我们现在要展示 BLA 如何应用于另一类问题，特别是二元分类问题。通常，这类问题会通过逻辑回归之类的方法解决：拟合一条（Sigmoid）曲线，为每个点分配概率，以预测某个观测值属于哪个类别。当然，我们知道，也可以训练神经网络来拟合这样的曲线，不过我们应该预期，最终得到的曲线图会更加复杂。因此，我们可以用下面的示例测试 BLA。

一如既往，我们使用由 6000 个观测值组成的训练集和包含 1000 个观测值的验证集。首先，我们考虑一个简单的数据集，其中只有一个预测变量，并有两个预先定义的类别。这些类别的数据由四个相互独立的伯努利分布生成，每个分布都定义在不同的区间上。具体而言，我们从整个定义域（包含全部四个区间）中生成一个随机值，然后根据该值将其分配给四个伯努利分布之一，并将其分类为 0 或 1。这样可以生成区分明显且包含少量噪声的数据。

在这个示例中，我们假设预测器定义在 $[0,1]$ 上。对于生成的两个类别，我们希望更接近 $x=0$ 的值以较高概率被分配到类别 $y=0$，而更接近

$x = 1$ 的值则应更常被映射到 $y = 1$。因此，我们使用以下函数生成数据集：

伯努利(0.05)，对应 $x < 0.3$伯努利(0.25)，
对应 $0.3 \leq x < 0.6$伯努利(0.75)，对应
$0.6 \leq x < 0.8$ 伯努利(0.95)，对应
$0.8 \leq x$

例如，如果我们生成一个随机样本，得到 $x_1 = 0.5$，那么该样本对应的 y值将是伯努利(0.25)的结果。从训练集中的 6000 个数据点里取出 300 个后，数据如下所示：

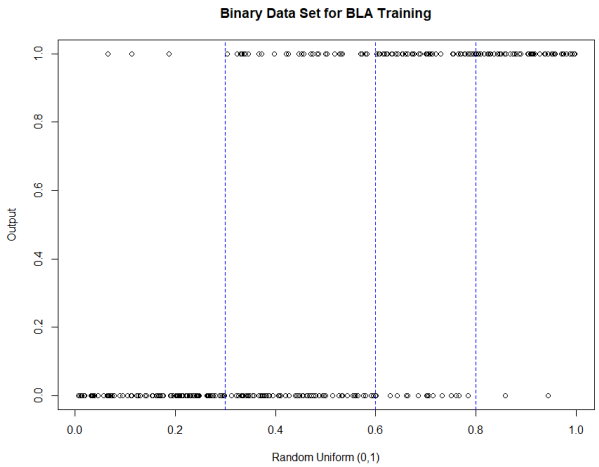


图 5：根据我们的概率分布生成的包含 300 个单元的训练集示例。

为了预测类别，我们首先使用这些数据和 BLA，按照上述所有标准设置训练一个 SHLNN。这个 SHLNN 函数 $g(x)$ 将生成集中在 $[0, 1]$ for$x \in [0, 1]$的实数。接下来，我们选取一个截断值 $c = 0.5$，使得任何 $g(x) \geq 0.5$ 都会被预测为类别 1，否则预测为类别 0。

为了证明 BLA 的收敛基本上在一个训练轮次内完成，我们使用该截断值进行了 1000 次试验，每次训练 1 个训练轮次。BLA 的平均准确率达到 84.364%。

由于在这种情况下，我们知道生成该数据集的底层公式，因此可以计算准确率的上限（这是因为总会存在一些“噪声”，换句话说，也就是随机性）。如果我们将 $x \geq 0.6$ 赋值为 1，其他值赋值为 0，并对数据进行数千次模拟，就可以计算出预测准确率的上限为 85%。因此，仅经过 1 个训练轮次后，BLA 的准确率就只比这个上限低 0.636%。

对于第二个数据集，我们希望构造一个二元数据集，使定义域中的多个（互不相交）子区域都能以较高概率被赋予类别 1。为此，我们将取自 a$[0, 2\pi]$-均匀分布的值传入一个特定的 $[0, 1]$ 值函数，该函数随后按分量指定不同的伯努利分布。因此，我们的输出是一族伯努利随机变量，其概率为 $p(x) = (\cos(x) + 1)/2$，适用于 $x \in [0, 2\pi]$。这个函数使定义域两端形成两个子区域，其中的点以较高概率被分类为 1；中间则有一个子区域，其中的点以较高概率被分类为 0。下面是一个包含 300 个观测值的训练集示例：

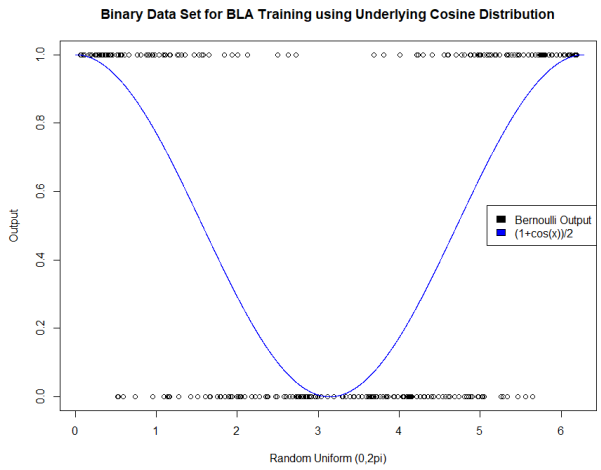


图 6: 使用 (1+cos(x))/2 概率分布生成的包含 300 个单元的示例训练集。 n.

与之前类似，我们模拟了该数据集准确率的上限。为此，我们生成了数千个数据集；如果某个单元在给定概率函数下的概率大于 0.5，就将其赋值为 1，否则赋值为 0。采用 $c = 0.5$ 的截断值后，我们估计平均准确率可达 81.83%。对 100 种不同的初始化各运行一次方法，仅训练一个训练轮次后，我们就达到了 81.114% 的准确率。与 ADAM 相比，若将该数据集运行 100 次，ADAM 在一个训练轮次后只能达到 60% 的准确率（至少对于 ADAM，后续准确率确实会逐渐接近这一水平；在我们的实验中，经过 100 个训练轮次后，平均准确率达到 81.6%）。下面列出了四种方法的详细结果（每种方法都以双曲正切作为首个激活函数）。

	第二个分类示例			
训练轮次	BLA	GD	ADAM	L-BFGS
1	81.114%	50.284%	60.045%	56.842%
5	80。98%	50。590%	78.575%	64。053%
10	81.122%	51.585%	80.874%	70。595%
15	81。387%	51。985%	81.212%	70。449&
25	81。584%	55。766%	81.461%	75。591%
50	81。763%	58。106%	81.583%	79。966%

5 结论与未来工作

本文介绍了自举学习算法（BLA），并提出将其用于监督学习，尤其是宽神经网络的训练。我们在若干回归和二元分类问题中测试了该方法。

对于宽神经网络，与训练轮次数量相近的其他常见机器学习技术相比，这种自举学习方法能够快速给出准确的估计，其精度（以均方误差衡量）可高出 100 倍以上。

从适用性、收敛性和使用效率的角度来看，该方法还需要进一步研究。例如，我们需要调整当前实现，以便将其应用于并测试于更具挑战性的问题，例如长短期记忆网络的学习。我们还应研究该方法应用于深度神经网络时的收敛性（以及收敛速度），这也将有望拓展其在多种其他应用中的用途。

我们预计可以开发出当前批次模式实现的合适在线版本。鉴于该方法已显示出快速收敛的特点，而且仅需少量训练数据即可实现，这一版本也将大有裨益。

最后，需要确立 BLA 的数学基础。

参考文献

[1] G. Cybenko (
1989 *Mathematics of control signals*)“通
and systems **2 4** 303 314
过叠加 sigmoid 函数进行逼近”，， （），第 - 页。[2] B. Delyon（2000），“递减增益随机逼近：收敛与渐近理论”，未发表的讲义，http://perso.univ-rennes1.fr/bernard.delyon/as cours.ps。[3] T. Dozat (
2016 *Proceedings of 4th International Conference*),
on Learning Representations Workshop Track
“将 Nesterov 动量融入 Adam”，，。[4] J. C. Duchi、E. Hazan 和 Y. Singer（2011），“用于在线学习和随机优化的自适应次梯度方法”， *Journal of Machine Learning Research*，12，第 2121-2159 页。[5] S. Espa˜na-Boquera、M. J. Castro-Bleda、J. Gorbe-Moya 和 F. Zamora-Martinez（2011），“使用混合 HMM/ANN 模型改进离线手写文本识别”， *IEEE Trans. Pattern Anal. Mach. Intell.*，33 4 767 779 10 1109 2010 141（），第 - 页。doi：。/TPAMI. . .[6] K. Hornik（1991），“多层前馈网络的逼近能力”， *Neural networks* **4**(2)，第 251-257 页。[7] D. P. Kingma 和 J. Ba（
2015 *Proceedings of 3rd International* ），
Conference on Learning Representations
“Adam：一种随机优化方法”，。[8] M.A. Kouritzin（
1994 *r* *Stochastics* ），
and Stochastics Reports **51** 241 266
“自适应滤波中归纳方法与 th-均值收敛速率”，，第 - 页。[9] M.A. Kouritzin（
1996 *IEEE Trans. Inform.* ），
Theory **42 4** 1305 1309
“关于线性随机逼近过程的收敛性”，IT-（），第 - 页。[10] M.A. Kouritzin（1996），“某些随机自适应算法与随机变量部分和的几乎必然不变原理之间的相互关系”， *J. Theoretical Probability* **9**(4)，第 811-840 页。[11] M. A. Kouritzin 和 S. Sadeghi（2015），“线性随机逼近算法的收敛速率与解耦”， *SIAM J. Control Optim.* **53**(3)，第 1484-1508 页。[12] Liu, D. C., Nocedal, J.（
1989 *Mathematical* ）“
Programming **45**, 503 528
大规模优化的有限记忆方法”，B. 第 - 页。[13] H. B. McMahan 和 M. J. Streeter（2010），“在线凸优化的自适应界优化”， *Proceedings of the 23rd Annual Conference On Learning Theory*，第 244-256 页。[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.（2011），“Scikit-learn：Python 机器学习”， *Journal of Machine Learning Research*，12，第 2825-2830 页。[15] Selvamuthu, D., Kumar, V. 和 Mishra（2019），“A. 使用逐笔数据和人工神经网络进行印度股市预测”。 *Financ Innov*，5(16)。https://doi.org/10.1186/s40854-019-0131-7。[16] T. Tieleman 和 G. Hinton（2012），“RMSProp：用近期幅值的运行平均值除梯度”， *COURSERA: Neural Networks for Machine Learning*，讲座 6.5。[17] M. D. Zeiler（2012），“ ADADELTA：自适应学习率方法”， *CoRR*。