文章

解析式训练双层 ReLU 网络

Adrian Barbu [1]*

[1] 佛罗里达州立大学统计系，塔拉哈西，佛罗里达州 32306，美国；abarbu@fsu.edu

摘要： 神经网络通常采用梯度下降优化算法的不同变体进行训练，例如随机梯度下降或 Adam 优化器。近期的理论研究指出，采用平方损失的双层 ReLU 网络的临界点（即损失的梯度为零的点）并非全都是局部极小值。然而，在这项工作中，我们将探讨一种训练采用类 ReLU 激活函数和平方损失的双层神经网络的算法：该算法交替解析求出损失函数关于一层的临界点，同时固定另一层及神经元的激活模式。实验表明，与随机梯度下降或 Adam 优化器相比，这种简单算法能够找到更深的最优解，并且在评估的五个真实数据集中的四个上获得显著更低的训练损失值。此外，该方法比梯度下降方法更快，且几乎不需要调整参数。

关键词： 神经网络优化；临界点

arXiv:2304.02972v1 [cs.LG] 6 Apr 2023

check for updates

引用： Barbu, A. 解析式训练双层 ReLU 网络。预印本**2021**，*1*，0。 https://doi.org/

收稿日期:
接受日期:
发布日期:

出版商说明： MDPI 对已发表地图中的管辖权主张和机构隶属关系持中立立场。

1. 介绍

神经网络的训练通常使用不同的基于梯度下降的算法，例如随机梯度下降或 Adam 优化器 [1]。这种训练需要多次遍历整个数据集，通常约为100-300 次，因此速度非常慢。此外，这种训练的结果对许多调优参数都很敏感，例如学习率和小批量大小，以及在训练迭代次数期间更改这些参数的方式（例如学习率调度）。由于这些原因，通常会使用不同的参数和调度组合进行多次训练（通常为 10次或更多），并使用其中最成功的一次来获得最终模型。训练至少需要100 个训练轮次，而且通常至少需要进行 10 次训练才能找到良好的组合，这意味着要很好地训练一个神经网络，必须至少遍历数据 1000 次，这可能会带来高昂的计算成本。

这些计算方面的原因促使我们研究一种训练双层神经网络的新方法，该方法采用平方损失，只需对全部数据进行至多 10 次遍历。该方法能够获得比经过良好调参的梯度下降算法更小的损失，并且在输入数量较少时具有更好的泛化能力。

在 [2] 中提到：“实践者发现，窄神经网络难以得到良好的求解结果。”我们的工作通过实验验证了这一说法，至少对于输入数量较少的神经网络而言如此；同时也表明，所提出的方法能够找到损失更低的局部极小值。这项工作有望推动浅层神经网络在许多计算受限应用中的广泛使用。 [2] 的作者还指出，算法的测试误差可以分解为表示误差、优化误差和泛化误差。这项工作针对的是一种范围较窄的神经网络，即采用类 ReLU 激活函数和平方损失的两层神经网络，研究其优化误差。尽管研究范围有限，我们仍表明，在某些情况下，所提方法的优化能力远胜于标准梯度下降方法。

本工作的贡献如下：

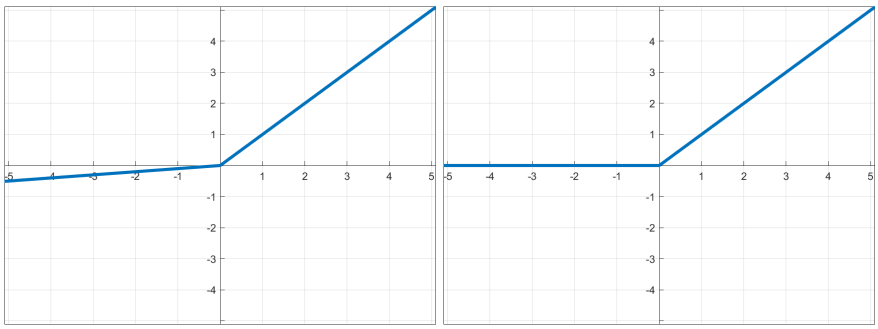


图1. 本文聚焦于采用类似 Leaky ReLU 的激活函数 $\sigma(x) = \alpha x + (1-\alpha)\max(0, x)$的两层神经网络，其中 $\alpha \in [0, 1)$。图中展示了 Leaky ReLU（$\alpha = 0.1$，左）和 ReLU（$\alpha = 0$，右）。

• 该方法通过在保持神经元激活模式固定的情况下，交替对每一层求解普通最小二乘法（OLS）方程，以解析方式训练具有Leaky ReLU类激活函数和平方损失的双层神经网络。神经元激活模式是一个二元矩阵，用于表示对于每个观测，哪些隐藏神经元被激活（具有非零激活值）。相关内容见第2.2节。

• 本文通过真实数据和模拟数据开展实验，以考察在什么条件下，所提出的方法优于标准的基于梯度的优化方法，例如标准随机梯度下降、Adam 优化器和 L-BFGS 算法。这些实验表明，当观测值数量不多或输入维度较小时，所提出的解析方法速度更快，并能找到损失更低的极小值。

本文结构安排如下：第 1.1 节概述相关工作。第 2 节介绍所提出的解析最小化方法，该方法在固定一层的同时拟合另一层。第3节展示真实数据和模拟数据上的实验结果以及消融研究。最后，第4节总结全文并讨论未来工作。

1.1. 相关工作

深度线性神经网络（无激活函数）在平方损失下的临界点已在 [3] 中得到研究。对于深度线性神经网络，作者证明了每个局部极小值同时也是全局极小值，而所有其他临界点都是鞍点。作者还研究了采用 ReLU 激活且使用平方损失的单隐藏层神经网络的临界点，但仅限于参数空间的某些区域。他们仅针对含有一个隐藏节点的神经网络，刻画了整个空间中的临界点。此外，他们并未尝试提供一种用于寻找临界点的显式算法。相比之下，本文提出了一种简单高效的算法，在保持神经元激活模式固定的同时，交替求解两层的临界点。

最近的工作 [4]表明，使用平方损失和随机梯度下降并进行适当正则化训练某些两层神经网络，可以找到损失函数的全局最优解。然而，这些网络必须使用平滑且有界的激活函数，例如 Sigmoid 函数或 Tanh 激活函数，而不能使用ReLU激活。这一证明使用了针对 Villani函数 [5,6],的庞加莱型不等式，该不等式已在 [6]中得到证明。

一些工作 [7,8] 放宽了对神经网络的限制，从而能够使用凸优化并获得全局解。第一项工作 [7] 针对卷积神经网络，其中共享滤波器被表示为低秩约束，并对其进行放宽以用于凸优化。第二项工作 [8] 专注于两层ReLU 网络，并利用凸对偶表明，两层 ReLU 网络可以通过二阶锥规划进行全局优化，其计算复杂度

方法	复杂度中的 N	d	h	Loss fn.	全局最优解保证
Adam [1]	N	d	h	any	No
随机梯度下降 [4]	N	d	h	ℓ_2	是（Sigmoid 函数和 Tanh 激活函数）
基于 ICA [10,11]	poly(N)	poly(d)	exp(h)	一些	No
凸对偶 [8]	N^d	exp(d)	-	ℓ_2	Yes
基于 TD [12]	poly(N)	poly(d)	poly(h)	-	No
所提出的方法	N	d^3	h^3	ℓ_2	No

表 1：相关工作的概述，包括时间复杂度和理论上的全局最优解保证。

$O\left(d^3r^3\binom{N}{r}^{3r}\right)$ 其中r 是输入数据矩阵的秩（对于较大的$r = d$，通常如此N）。该算法关于样本量N 是多项式时间，但关于输入维度d是指数时间，因此当N 和d较大时，计算开销可能难以承受。他们还证明，除非 P=NP，否则无法降低关于d 的指数复杂度。相比之下，本文提出的算法速度更快，复杂度为$O((dh)^3 + N(dh)^2)$，其中h 是隐藏节点的数量，但它不保证能找到全局最优解。文献 [9], 进一步改进了凸松弛的思路，基于门控 ReLU 模型提出了更快的近似算法。不过，该算法基于梯度，依赖加速近端梯度方法。该方法的优势在于可应用于任意损失函数，而不仅限于我们的方法所需的平方损失。

其他研究 [10,11] 尝试在隐藏节点的偏置项为 0 的假设下，开发能够在多项式时间内学习神经网络的算法。在 [10] 中，作者研究了具有 ReLU 激活函数的两层神经网络，并假设隐藏节点的真实权重矩阵满秩。他们提出了一种关于 N 和 d 为多项式、但关于隐藏节点数 h 为指数的算法；在特定假设下，该算法保证能够找到真实系数。然而，该算法基于独立成分分析，而非损失最小化。同样，我们的算法关于隐藏节点数 h为三次复杂度，但无法保证得到全局最优解。

在 [12], 中，在假设隐藏节点的权重矩阵具有线性无关列的条件下，放宽了零偏置假设。他们的算法基于张量分解，而非优化损失函数。该算法关于 N、h 和d 为多项式时间；给定足够多的样本后，它可以在误差不超过给定值 ϵ的范围内恢复真实系数（允许置换）。然而，最主要的缺点是该算法假设目标数据中没有噪声，因此无法在实践中使用。 [13], 也采用了这种无噪声假设；其中作者证明，仅凭网络输出就能恢复深度 ReLU 神经网络，包括其架构、权重和偏置（允许同构变换）。为此，他们利用了网络输出的分段线性区域，这些区域由神经元激活模式定义，而我们的工作也使用了这些模式。

可以认为，使用 Hessian 矩阵逆矩阵近似的 BFGS [14] 和 L-BFGS [15] 算法，在某种程度上是所提出方法的有力竞争者。然而，这些算法不会在每次迭代时冻结神经元激活模式，而是试图在原始参数空间的能量景观中搜索最优解。这些方法的问题在于，由于分段线性 ReLU 激活函数具有不可微性，能量景观中存在许多逆 Hessian 矩阵为无穷大的点。因此，BFGS算法和 L-BFGS 算法会在局部最优解附近发散，这一点可以通过实验轻易观察到。相比之下，在我们的工作中，通过冻结神经元激活模式，实验观察到解析解始终存在

对于某些数据（实验中的五个真实数据集中的三个）如此，对于其余数据通常也存在解析解。

表格 1 概述了上述工作，包括其计算复杂度、所优化的损失函数类型，以及是否提供全局最优解保证。

2. 材料与方法

本研究将聚焦于两层 ReLU 网络：

$$f(\mathbf{x}) = \sigma((1, \mathbf{x}^T)\mathbf{A})\mathbf{B} + \mathbf{b}^0 \in \mathbb{R}^c$$

其中$\mathbf{x} \in \mathbb{R}^d$是输入向量，$\mathrm{A} = (\mathrm{a}_1,...,\mathrm{a}_h)$是隐藏层权重的$(\mathrm{d}+1) \times \mathrm{h}$矩阵，$\mathrm{B} = (\mathrm{b}_1,...,\mathrm{b}_c)$是输出层权重的$\mathrm{h} \times \mathrm{c}$矩阵，而$\mathrm{b}^0 = (\mathrm{b}_1^0, ..., \mathrm{b}_c^0)$是 c 个输出的偏置。我们关注的激活函数是 LeakyReLU 类型 $\sigma(\mathrm{x}) = \alpha\mathrm{x} + (1-\alpha)\max(0,\mathrm{x})$，其中 $\alpha \in [0,1)$。在实验中，我们将研究 ReLU($\alpha = 0$) 和 Leaky ReLU ($\alpha = 0.1$)（图 1）。

给定观测值 $(\mathbf{x}_1, \mathbf{y}_1)$、...、$(\mathbf{x}_N$、$\mathbf{y}_N) \in \mathbb{R}^d \times \mathbb{R}^c$，这些网络的平方损失函数为：

$$L(\mathbf{A}, \mathbf{B}, \mathbf{b}^0) = \frac{1}{N}\sum_{k=1}^{c}\sum_{i=1}^{N} \|\sigma((1, \mathbf{x}_i^T)\mathbf{A})\mathbf{b}_k + b_k^0 - y_{ik}\|^2 + \lambda(\|\mathbf{b}^0\|_2^2 + \|\mathbf{B}\|_F^2 + \|\mathbf{A}\|_F^2)), \tag{1}$$

其中 $\|A\|_F^2 = \sum_i, {}_j A_{ij}^2$ 是弗罗贝尼乌斯范数。

我们将使用标准记号，用$\mathbf{X}$ 表示观测值矩阵

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & ... & x_{1d} \\ & ... & & \\ 1 & x_{N1} & ... & x_{Nd} \end{pmatrix}, \tag{2}$$

用$\mathbf{Y}$ 表示输出矩阵，其中$\mathbf{y}_i^T$为行，$\mathbf{y}_{\cdot j}$ 为列，

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y}_1^T \\ ... \\ \mathbf{y}_N^T \end{pmatrix} = \begin{pmatrix} y_{11} & ... & y_{1c} \\ & ... & \\ y_{N1} & ... & y_{Nc} \end{pmatrix} = (\mathbf{y}_{\cdot 1}, ..., \mathbf{y}_{\cdot c}), \tag{3}$$

其中 $\mathbf{y}_{\cdot j}$ 是第j个输出的目标向量。

我们将使用一种通过交替寻找损失（1）关于 $\mathbf{B}$ 和 $\mathbf{A}$ 的临界点来最小化损失的算法，其中对于关于 $\mathbf{A}$ 的临界点，神经元激活模式保持冻结。该过程在算法 1中进行了描述。

阈值 τ 控制线性系统 $(\mathbf{M} + \lambda\mathbf{I}_{(d+1)h})\mathbf{a} = \mathbf{C}$ 何时可以进行数值求解。在实践中，我们取 $\tau = -10,\ 000$。

2.1. 拟合输出层权重 $\mathbf{B}$

拟合输出层权重矩阵 $\mathbf{B}$ 是简单的普通最小二乘法，使用 $\mathbf{S} = (\sigma(\mathbf{XA}), \mathbf{1}_N)$ 作为输入。由于平方损失是各输出变量的平方损失之和，因此可以分别针对每个输出变量求解$\mathbf{B}$：

$$(\frac{1}{N}\mathbf{S}^t\mathbf{S} + \lambda\mathbf{I}_{h+1})\begin{pmatrix} \mathbf{b}_k \\ b_k^0 \end{pmatrix} = \mathbf{S}^T\mathbf{y}_{\cdot k}, k = 1, ..., c \tag{4}$$

算法 1 解析最小化（ANMIN）

输入： 训练数据 **X**,**Y**.
输出： 已训练的网络参数向量($\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{b}}_0$).

1：随机初始化**A**
2: 拟合 ($\mathbf{B}$,$\mathbf{b}^0$) 使用公式（4）。
3：计算损失 $l_0 = L(\mathbf{A},\mathbf{B},\mathbf{b}^0)$
4：**对于**$i = 1$到 E **执行**
5：　更新$\mathbf{F} = I(\mathbf{XA} > 0)$ 和 $\mathbf{G} = (1-\alpha)\mathbf{F} + \alpha\mathbf{1}$
6：　计算$\mathbf{M}$ 使用公式 (7) 和 $\mathbf{c}$ 使用公式 (8)
7：　**如果** $\ln|\mathbf{M} + \lambda\mathbf{I}_{(d+1)h}| > \tau$ **则**
8：　　求解$(\mathbf{M} + \lambda\mathbf{I}_{(d+1)h})\mathbf{a} = \mathbf{c}$
9：　**else**
10：　　通过奇异值分解求得 $\mathbf{U}$、$\mathbf{D}$、$\mathbf{V}$，使得
　　$\mathbf{UDV}^T = \mathbf{M} + \lambda\mathbf{I}_{(d+1)h}$
11：　　如果 $D_{ii} < 0.0001$，则令 $D_{ii} = 0.0001, i = 1, ..., dh$
12：　　求得$\mathbf{a} = \mathbf{UD}^{-1}\mathbf{V}^T\mathbf{c}$
13：　**结束如果**
14：　将向量 $\mathbf{a}$ 重塑为 $(d+1) \times h$ 矩阵 $\mathbf{A}$
15：　拟合 ($\mathbf{B}$,$\mathbf{b}^0$) 使用方程（4）。
16：　计算损失 $l_i = L(\mathbf{A},\mathbf{B},\mathbf{b}^0)$
17：　**若**$l_i < l_0$，**则**
18：　　令 $\hat{\mathbf{A}} = \mathbf{A}$，$\hat{\mathbf{B}} = \mathbf{B}$，$\hat{\mathbf{b}}^0 = \mathbf{b}^0$
19：　　设置$l_0 = l_i$
20：　**end if**
21: **结束循环，**
适用于

2.2. 拟合隐藏层权重 **A**

令 $\mathbf{F} = I(\mathbf{XA} > 0)$ 为 $N \times h$ 维二值激活模式矩阵，用于指示每个观测对应的哪些隐藏神经元被激活。此处 $I(x)$ 是逐元素指示算子。令$\mathbf{G} = (1-\alpha)\mathbf{F} + \alpha\mathbf{1}$，其中$\mathbf{1}$ 是所有元素均为 1 的矩阵。

然后，针对训练观测 ($\mathbf{x}_i$,$\mathbf{y}_i$)的神经网络响应为

$$\hat{y}_{ik} = b_k^0 + \sum_{j=1}^{h} b_{jk} G_{ij} \tilde{\mathbf{x}}_i \mathbf{a}_j = (G_i * (\tilde{\mathbf{x}}_i \mathbf{A}))\mathbf{b}_k + b_k^0, k = 1, ..., c$$

其中G_i 是i-**G**的第i行，“$*$”是逐元素乘法，而 $\tilde{\mathbf{x}}_i = (\mathbf{1}, \mathbf{x}_i^T)$ 是i-**X**的第i行。然后，所有响应都可以写成

$$\hat{\mathbf{Y}} = (\mathbf{G} * (\mathbf{XA}))\mathbf{B} + \mathbf{b}^0$$

将激活模式 $\mathbf{G}$ 添加到损失函数参数中，得到

$$L(\mathbf{A}, \mathbf{B}, \mathbf{b}_0, \mathbf{G}) = \frac{1}{N} \sum_{k=1}^{c} \sum_{i=1}^{N} \| \sum_{j=1}^{h} b_{jk} G_{ij} \tilde{\mathbf{x}}_i \mathbf{a}_j + b_k^0 - y_{ik} \|^2 + \lambda(\|\mathbf{b}^0\|^2 + \|\mathbf{B}\|_F^2 + \|\mathbf{A}\|_F^2)). \quad (5)$$

Let

$$\mathbf{U}_{jl} = \frac{1}{N} \sum_{i=1}^{N} G_{ij} G_{il} \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^T, \quad (6)$$

它们是 $(d+1)\times(d+1)$ 矩阵，可通过批次增量计算。令

$$\mathbf{M}=\begin{pmatrix}\mathbf{M}_{11} & \mathbf{M}_{12} & \ldots & \mathbf{M}_{1h}\\ \ldots & \ldots & \ldots & \ldots\\ \mathbf{M}_{h1} & \mathbf{M}_{h2} & \ldots & \mathbf{M}_{hh}\end{pmatrix} \tag{7}$$

为具有单元 $(d+1)h\times(d+1)h$ 的$\mathbf{M}_{jl}=\sum_{k=1}^{c} b^{jk}b_{lk}\mathbf{U}^{jl}$矩阵。

另外，令

$$\mathbf{c}=(\mathbf{v}_1^T,\ldots,\mathbf{v}_h^T)^T,\mathbf{v}_j=\frac{1}{N}\sum_{k=1}^{c}\sum_{i=1}^{N} b_{jk}(y_i-b_k^0)G_{ij}\mathbf{x}_i. \tag{8}$$

以下定理描述了为解析求出损失函数 $L(\mathbf{A},\mathbf{B},\mathbf{b}^0,\mathbf{G})$ 关于 $\mathbf{A}$ 的临界点而需要求解的线性方程组。

定理 1。 若矩阵 $\mathbf{G}$ （见公式 (5) ）固定，则损失函数 L 关于 $\mathbf{A}$ 的临界点$(\mathbf{A},\mathbf{B},\mathbf{b}^0,\mathbf{G})$（见公式 (5) ）是下列方程的解：

$$(\mathbf{M}+\lambda\mathbf{I}_{(d+1)h})\mathbf{a}=\mathbf{c}, \tag{9}$$

其中 $\mathbf{a}=(\mathbf{a}_1^T,\ldots,\mathbf{a}_h^T)^T$ 是矩阵$\mathbf{A}$ 展开后的形式。

证明。 损失（5）也可以写成

$$\begin{aligned}L(\mathbf{A},\mathbf{B},b_0,\mathbf{G})=&\frac{1}{N}\sum_{k=1}^{c}\sum_{j,l=1}^{h}\mathbf{a}_j^T b_{jk}b_{lk}\Big(\sum_{i=1}^{N}G_{ij}G_{il}\mathbf{x}_i\mathbf{x}_i^T\Big)\mathbf{a}_l+\frac{2}{N}\sum_{j=1}^{h}\sum_{k=1}^{c}\sum_{i=1}^{N}b_{jk}(b_k^0-y_i)G_{ij}\mathbf{x}_i^T\mathbf{a}_j\\&+\lambda(\mathbf{b}^{0T}\mathbf{b}^0+\sum_{k=1}^{c}\mathbf{b}_j^T\mathbf{b}_k+\mathbf{a}^T\mathbf{a}),\end{aligned} \tag{10}$$

因此

$$L(\mathbf{A},\mathbf{B},b_0,\mathbf{G})=\sum_{j,l=1}^{h}\mathbf{a}_j^T\Big(\sum_{k=1}^{c}b_{jk}b_{lk}\mathbf{U}_{jl}\Big)\mathbf{a}_l-2\sum_{j=1}^{h}\mathbf{v}_j^T\mathbf{a}_j+\lambda(\mathbf{b}^{0T}\mathbf{b}^0+\sum_{k=1}^{c}\mathbf{b}_j^T\mathbf{b}_k+\mathbf{a}^T\mathbf{a}). \tag{11}$$

然后，损失函数可利用公式（7）写为

$$L(\mathbf{a},\mathbf{B},b_0,\mathbf{G})=\mathbf{a}^T\mathbf{M}\mathbf{a}-2\mathbf{c}^T\mathbf{a}+\lambda(\mathbf{b}^{0T}\mathbf{b}^0+\sum_{k=1}^{c}\mathbf{b}_j^T\mathbf{b}_k+\mathbf{a}^T\mathbf{a}). \tag{12}$$

这是关于$\mathbf{a}$ 的二次函数，将其关于$\mathbf{a}$ 的梯度设为零，我们得到公式（9）。

□

2.3. 计算复杂度

算法 ANMIN 的计算复杂度（如算法 1 所述）可轻松计算如下。

计算用于求解 B 的输入S 的复杂度为 O(Ndh)，拟合B的复杂度为$O(h^3+ch^2)$。计算每个矩阵U^{ij} 的复杂度为$O(Nd^2)$，因此计算 M 的复杂度为$O(N(dh)^2)$。拟合A 的复杂度为$O((dh)^3+c(dh)^2)$。由于迭代次数是固定的，整个算法的复杂度为 $O((dh)^3+(N+c)(dh)^2)$，因此其关于 N 和 c 呈线性，但关于 d 和 h 呈三次方。

数据集	N	d	h
SDF 解码器	73,080	2	64
鲍鱼	4,177	7	64
共享单车	17,379	13	64
YearpredMSD	515,345	90	64
DAE	15,912	225	32

表 2：实验中使用的真实数据集及其观测值数量n 和特征维度d。还列出了实验中使用的神经网络隐藏节点数量h 。

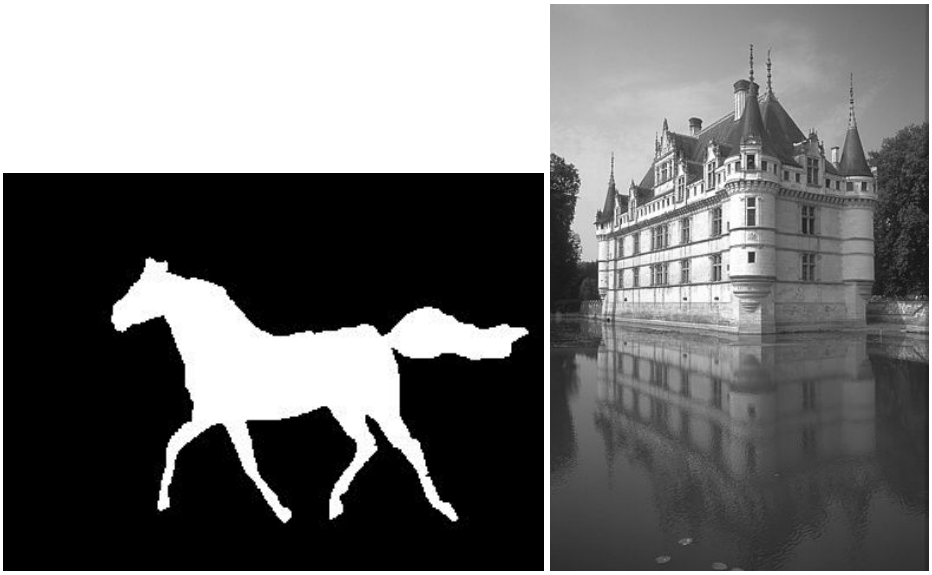


图 2。 左：用于训练形状解码器的有符号距离变换所对应的二值形状图像。右：用于训练去噪自编码器（DAE）的图像。

2.4. 使用的数据集

真实数据实验将在加州大学欧文分校机器学习仓库中的三个数据集上进行 [16] 以及专门为两个计算机视觉任务生成的两个数据集。这五个数据集总结于表2中。

使用了来自加州大学欧文分校机器学习资料库 [16] 的鲍鱼、共享单车和年份预测MSD数据集，这三个都是回归数据集。

鲍鱼数据集用于根据鲍鱼的身体测量值预测其年龄。它包含4177个观测值和7个特征，因此是一个低维数据集。

共享单车数据集用于预测共享单车租赁系统中的每小时和每日单车租赁数量。使用了hourlycount，其中包含17,379 个观测值和 14个特征。由于计算注册用户数量的特征与响应变量高度相关，因此将其移除，并使用剩余的 13 个特征进行预测。

年份预测MSD数据集旨在根据从歌曲中提取的音频特征预测歌曲的发行年份。它包含 515,345 个观测值和 90 个特征。

另一个数据集基于 [17]中的单形状深度 SDF 任务生成。该任务旨在根据像素坐标 (x,y)预测有符号距离变换（SDF）矩阵的值。输入形状使用了Weizmann 马数据集的第一张掩膜图像 [18]，（如图 2左侧所示），并据此计算 SDF。将图像的所有像素坐标作为输入，并将其对应的SDF 值作为目标，得到一个包含 73,080 个观测值和两个特征的数据集。

这些回归数据集都是用于预测单个输出。为了评估 ANMIN 预测多个输出的性能，使用神经网络作为去噪

自编码器（去噪自编码器）。在这种情况下，目标输出由自然图像块构成，而输入则是被噪声污染的相同图像块。这些图像块取自伯克利数据集中的一幅图像 [19], , 如图 2右侧所示。以 3 为步长，从该图像中提取了 15,912 个大小为 15×15 的重叠图像块。因此，数据共有 15,912 个观测值、225 个输入和 225 个输出。输出是原始图像块，输入则是加入了标准差为 $\sigma = 10$ 的高斯噪声的相同图像块。

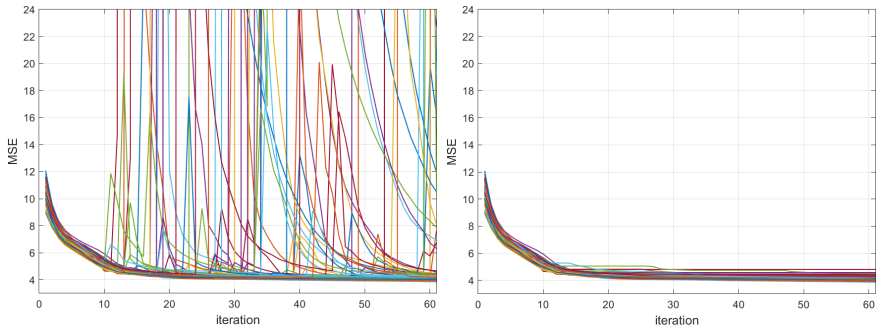


图 3. 在鲍鱼数据集的 100 个随机划分上使用 L-BFGS 算法得到的训练均方误差。左：训练均方误差中有许多位置会导致 L-BFGS 算法失稳。右：考虑每次迭代为止获得的最小训练均方误差，可以缓解这一问题。F

or all d数据集，一个 hundred rand$^{\text{om sp}}$ lits of the data i到一个 80% trai$^{\text{n 设置一个}}$ d 20% 生成了测试集。

一个具有h 个隐藏节点、采用 ReLU 或 Leaky ReLU 激活函数的两层神经网络，使用不同的训练算法在随机划分的训练部分上进行训练，然后在相应的测试集上进行测试。隐藏节点的数量为$h = 64$，但 DAE 数据除外，其数量为$h = 32$，以获得低维表示。

2.5. 方法比较

所提出的 ANMIN算法与随机梯度下降算法、Adam 优化器 [1] 以及 L-BFGS 算法 [15]进行了比较。主要关注SGD 和Adam，因为 L-BFGS 在优化过程中会多次发散到如此大的数值，正如图 3左侧所示，其中绘制了鲍鱼数据集的100个训练损失。出于这一原因，对于 L-BFGS 算法，在每次运行中都提取获得最小训练均方误差的模型，并将其用于测试。ANMIN 也采用了相同的方法（算法 1的第 17-20 行）。

ANMIN 和 Adam 分别使用两种激活函数进行实验：ReLU ($\alpha = 0$) 和 Leaky ReLU ($\alpha = 0.1$)；而 L-BFGS 和随机梯度下降仅使用 ReLU 进行实验。实验中，使用 Leaky ReLU 的方法分别称为 ANMIN-L 和 Adam-L。

2.6. 实现细节

所有算法均使用 PyTorch 实现。实验在一台配备 32 GB 内存和 GeForce 3080 GPU 的 Core I7 计算机上进行。代码可在https://github.com/barbua/ANMIN/获取。

随机梯度下降和 Adam 均训练了 300 个训练轮次。Adam 的初始学习率为 0.03，而随机梯度下降的学习率则针对每个数据集单独调节，以避免损失过度增大，并获得较小的最终损失值。每经过 100 个训练轮次，学习率就降低为原来的十分之一。ANMIN 算法运行了 30 次迭代。L-BFGS 以 0.01 的学习率运行了 60 次迭代。

对于 Adam 和随机梯度下降，使用了256的批量大小；对于年份预测MSD，则使用了 2048 的批量大小。对于 L-BFGS，使用了 100,000 的批量大小。对于 ANMIN，使用了 256 的批量大小。批量大小用于 ANMIN中累积矩阵 $\mathbf{U}^{jl}$

	均方误差训练					
数据集	SGD	Adam	Adam-L	L-BFGS	ANMIN	ANMIN-L
SDF 解码器	22.43(5.29)	12.35(4.48)	9.46(2.49)	25.71(62.08)	**3.53**(1.05)	**3.45**(1.15)
鲍鱼	4.20(.08)	4.00(.09)	3.99(.09)	4.17(.16)	**3.94**(.10)	**3.95**(.09)
共享单车	2422(100)	2465(181)	2477(193)	5710(4581)	**1457**(319)	**1555**(465)
YearPredMSD	**71.67**(0.16)	**70.98**(.15)	**71.01**(.14)	74.51(2.55)	72.87(.18)	73.08(.21)
DAE	200.8(3.1)	165.5(4.9)	163.4(4.5)	2990(9.2)	155.2(4.0)	**150.9**(2.7)
	MSE测试					
数据集	SGD	Adam	Adam-L	L-BFGS	ANMIN	ANMIN-L
SDF 解码器	22.36(5.18)	12.36(4.46)	9.48(2.49)	25.65(61.96)	**3.56**(1.06)	**3.48**(1.14)
鲍鱼	**4.45**(.41)	5.05(1.89)	4.99(1.71)	**4.50**(.46)	**4.64**(.55)	**4.66**(.95)
共享单车	2681(171)	2721(224)	2737(239)	5850(4280)	**1714**(385)	**1808**(493)
YearPredMSD	**76.51**(0.53)	**76.71**(.51)	**76.68**(.55)	77.17(1.29)	**76.88**(.54)	**76.82**(.49)
DAE	207.1(6.1)	170.8(7.1)	168.7(6.1)	2988(36.8)	160.4(5.6)	**156.1**(4.8)
	R^2 列车					
数据集	SGD	Adam	Adam-L	L-BFGS	ANMIN	ANMIN-L
SDF 解码器	0.968(.008)	0.982(.007)	0.986(.004)	0.963(.089)	**0.995**(.002)	**0.995**(.002)
鲍鱼	0.596(.005)	0.615(.005)	0.615(.006)	0.598(.014)	**0.621**(.007)	**0.620**(.007)
共享单车	0.926(.003)	0.925(.006)	0.925(.006)	0.826(.139)	**0.956**(.010)	**0.953**(.014)
YearPredMSD	0.400(.001)	**0.406**(.001)	**0.406**(.001)	0.376(.021)	0.390(.001)	0.388(.002)
DAE	0.933(.001)	0.945(.002)	0.945(.002)	0.000(.000)	0.948(.001)	**0.950**(.002)
	R^2 测试					
数据集	SGD	Adam	Adam-L	L-BFGS	ANMIN	ANMIN-L
SDF 解码器	0.968(.008)	0.982(.006)	0.986(.004)	0.963(.090)	**0.995**(.002)	**0.995**(.002)
鲍鱼	**0.573**(.028)	0.515(.178)	0.521(.164)	**0.568**(.035)	**0.555**(.049)	**0.554**(.079)
共享单车	0.919(.005)	0.918(.006)	0.917(.007)	0.822(.130)	**0.948**(.012)	**0.945**(.015)
YearPredMSD	**0.360**(.003)	0.359(.003)	0.359(.003)	0.355(.011)	0.357(.003)	0.358(.003)
DAE	0.931(.002)	0.943(.002)	0.944(.002)	0.000(.000)	0.946(.002)	**0.948**(.002)

表3：真实数据实验。在五个数据集上，训练和测试均方误差的最终平均值以及R^2（均附标准差）。所有结果均基于100次随机划分，每次划分为80%训练集和20%测试集。最佳结果以及与最佳结果相比差异不显著（$p > 0.01$）的结果以粗体显示。

根据式（6），因此不同的批量大小会得到相同的解，而且我们观察到，批量大小在 128 到 2048 之间时，计算时间大致相同。ANMIN 的收缩参数 λ 取值为 $\lambda = 0.001$。

3. 结果与讨论

我们在多个真实数据集上开展实验，以深入了解ANMIN方法相较于常规基于梯度的训练在何时表现良好，以及在何时表现不佳。我们将看到，当输入维度相对于隐藏神经元数量较小时，ANMIN 的优势更加明显。随后，我们将在一个非线性数据集上进行模拟，其中输入维度和观测值数量可以根据需要进行调整。

3.1. 真实数据实验

训练和测试均方误差以及训练和测试 R^2 汇总于表3中。最佳结果以及通过配对 t检验与最佳结果进行显著性比较后获得 $p > 0.01$ 的结果以粗体显示。

图 4、5、6 展示了在全部 100 次随机划分上，ANMIN 方法与 Adam、随机梯度下降和 L-BFGS 方法（均采用 ReLU 激活函数）的训练均方误差损失和平均 R^2 曲线。图中还绘出了测试均方误差均值和 R^2 曲线及其标准差。ANMIN 和 L-BFGS 的测试均方误差和 R^2 基于训练均方误差损失最小的模型。

查看图4、5、6以及表3中的均方误差损失值可以发现，在五个数据集中有四个数据集上，ANMIN 方法的训练损失显著低于 Adam、随机梯度下降和 L-BFGS。在四个数据集上，其测试均方误差显著低于 Adam 和 L-BFGS；在三个数据集上显著低于随机梯度下降。此外，在这五个数据集中的任何一个上，随机梯度下降、Adam 或 L-BFGS 在测试集上的表现都从未显著优于 ANMIN。L-BFGS 算法仅在鲍鱼数据集上表现出色，在年份预测MSD数据集上的表现尚可，在其他三个数据集上的表现则很差。

观察R^2 值（来自图4、5、6以及表3）可以看出，ANMIN 方法在五个数据集中的四个数据集上获得了显著高于随机梯度下降、Adam 和 L-BFGS 的训练R^2 值。在四个数据集上，其测试R^2 值显著高于 Adam 或L-BFGS，在三个数据集上显著高于随机梯度下降。在一个数据集上，Adam 和随机梯度下降的表现显著优于 ANMIN。

ANMIN 方法优于Adam 和随机梯度下降的数据集有什么共同点？可以注意到，相关的单输出数据集的输入维度都较低，$d \leq 13$。因此，真实数据实验似乎支持这样一个假设：当数据的输入维度（从而隐藏层中的参数数量）较小时，ANMIN 相较于标准梯度优化具有优势。该假设将在模拟部分3.2中进一步探讨。在这种情况下，SGD 和 Adam似乎难以探索低维参数空间并找到更深的局部最优解。事实上，在输入维度最小的数据集（即 SDF 解码器）上，ANMIN的均方误差约为SGD和 Adam 的三分之一。在鲍鱼和共享单车这另外两个特征数量较少的数据集上，ANMIN 在训练集上的表现再次显著优于 Adam和 SGD，但在鲍鱼数据集的测试集上，SGD 不知为何表现出了更好的泛化能力。

除鲍鱼数据集外，L-BFGS 算法在所有数据集上的表现都不佳；几乎在所有情况下，其训练均方误差都很高，是所评估方法中表现最差的。

数据集	随机梯度下降	Adam	Adam-L	L-BFGS	ANMIN	ANMIN-L
SDF 解码器	153	223	224	79.6	**12.8**	19.3
鲍鱼	9.1	9.1	9.1	16.4	1.3	**0.9**
共享单车	40.2	36.7	36.0	35.5	4.7	**3.2**
YearpredMSD	1089	1246	1084	493	**203**	208
DAE	31.4	31.1	31.4	30.7	**18.1**	**18.1**

表4：表中实验的计算时间（秒）3。

表4展示了所评估方法的计算时间。比较计算时间可从图4 看出，在三个低维数据集上，ANMIN 用时仅为 Adam 的 1/5 到 1/10，便能达到与 Adam 最终损失相当的训练损失值；增加迭代次数还能进一步改善损失值。即使比较总计算时间（ANMIN 迭代 30 次，Adam 迭代 300 次），ANMIN 方法也具有计算优势：对于去噪自编码器（ DAE ），速度约快 2 倍；对于三个低维数据集，速度则快 7 倍以上。

这些实验表明，ANMIN 方法在某些情况下可能具有优势，例如输入维度d 较小时。下一节将进一步研究这一假设。

3.2 仿真

本节将通过模拟来检验这一假设：当输入维度较小时，ANMIN 相较于 Adam 的优势更加明显。

模拟数据包含从 $\mathcal{N}(0, \mathbf{I}_d)$ 中采样得到的$\mathbf{x} \in \mathbb{R}^d$，以及$y = \sin(\|\mathbf{x}\|_2^2)$。维度$d$取自$d \in \{1, 3, 10, 30\}$。采样了数量为$N \in \{1000, 10000\}$的观测值用于训练，并采样了相同数量的观测值用于测试。

该模型是一个具有$h = 64$ 个隐藏节点并采用 ReLU 激活函数的两层神经网络。值得注意的是，当 d 较大时，该模型无法很好地拟合数据。

使用 ANMIN（算法 1）对模型进行训练，共迭代 30 次。为作比较，还使用 Adam 优化器进行了 300 次迭代，[1] 初始学习率为$\eta = 0.03$，并且每 100 次迭代将学习率降低 10 倍。批量大小为 256。

图 7展示了不同 n 和 d 组合下，连续运行 100 次所得的均方误差随迭代次数变化的曲线。为便于观察，图 8 绘出了两种方法在 100 次运行中得到的训练和测试均方误差；对于每种方法，运行结果均按测试均方误差降序独立排列。这种展示方式有助于观察每种方法的训练均方误差与测试均方误差之间是否存在趋势，也有助于更清楚地了解测试均方误差的分布。

可以看出，对于较小的 $d \in \{1, 3\}$，ANMIN 得到的训练和测试损失都小于 Adam。对于 $d \in \{10, 30\}$，Adam 得到的训练损失较小，但过拟合程度高于 ANMIN。在所有情况下，ANMIN 方法的测试均方误差都较小。

3.3. 消融研究

本节研究为算法1所做的设计决策如何影响其性能。将考察以下设计决策：

- 是否使用迭代过程中出现的最小损失值，而不是算法的最终损失。
- 当线性系统（9）退化时，是停止、随机重新初始化 **A**，还是使用伪逆继续计算。

消融实验在共享单车数据集上进行，该数据集的对数行列式值较小，因此更难优化。结果汇总于表 5。

实验	最小	pinv	训练均方误差	测试均方误差
1	-	-	8919 (4095)	9010 (4056)
2	+	-	1826 (660)	2092 (662)
3	-	+	2199（1972）	2450（1955）
4 (算法1)	+	+	1457（319）	1714（385）

表 5：消融实验。ANMIN 算法在共享单车数据集上获得的平均均方误差（含标准差）1 其中移除了不同部分。

在实验 1 中，当对数行列式首次小于 $-10{,}000$ 时，停止 ANMIN 算法并报告最终损失。

在实验 2 中，每次迭代时，只要对数行列式小于 $-10{,}000$，就用随机值重新初始化矩阵 **A**。报告所有迭代中训练损失最低的模型及其训练损失和测试损失。

在实验 3 中，按照算法 1所述运行算法，但报告最终模型，而非迭代过程中损失最低的模型。

实验 4 完全按照算法 1所述进行。

从表 5 可以看出，迭代次数中的最小值以及使用伪逆而非随机重启，都有助于获得较小的最终损失。

图 9展示了 SDF 解码器和 bike- sharing 数据集在每次迭代时的均方误差训练损失值（左）以及迄今为止得到的最小均方误差（右）。

可以看到，每次迭代时的均方误差损失有时会突然增大（通常是在对数行列式较小或 $-\infty$ 时）。不过，这些跳变远小于 L-BFGS 方法中的跳变。这是因为每次更新矩阵 **A**（可能导致跳变）后，也会更新矩阵 **B** 以使均方误差最小化。

3.4. 诊断 *ANMIN* 算法

与基于梯度的算法不同，ANMIN 方法提供了一些能力，可在优化过程中诊断能量景观。为此，可以使用 $\mathrm{M} + \lambda \mathrm{I}(\mathrm{d}+1)\mathrm{h}$ 的条件数，或其对数行列式 $\ln|\mathrm{M} + \lambda \mathrm{I}(\mathrm{d}+1)\mathrm{h}|$，来判断 ANMIN 在拟合隐藏节点 A 时所达到的局部最优解的退化程度。图 10 展示了对随机划分进行的 100 次 ANMIN 运行中，优化过程遇到的对数行列式值 $\ln|\mathrm{M} + \lambda \mathrm{I}(\mathrm{d}+1)\mathrm{h}|$。一些曲线出现中断，因为在某些迭代次数下，对数行列式为 $-\infty$。可以看到，对于单输出数据集，对数行列式有下降趋势；而对于多输出 DAE 问题，对数行列式则随着迭代次数增加而增大。对于低维数据集（鲍鱼和共享单车），对数行列式会降至非常小的值，使矩阵 $\mathrm{M} + \lambda \mathrm{I}(\mathrm{d}+1)\mathrm{h}$ 接近奇异，从而导致式（9）的解稳定性较差。

4. 结论

本文介绍了一种训练双层神经网络的方法：使用平方损失，通过在固定另一层以及神经元的激活模式的同时，交替拟合每一层的系数，来训练使用 ReLU 或类似 Leaky ReLU 的激活函数的双层神经网络。

实验表明，在某些情况下（输入维度较小时），该方法能够找到超越梯度下降优化所能达到水平的损失函数强局部极小值。它所需的迭代次数相对较少（10–30 次），这意味着它能够处理无法装入计算机内存的大型数据集。

这项工作本质上是实验性的：我们提出一种算法，并通过实验观察到它效果良好，但没有提供任何证明或理论保证来说明它在某些假设下始终有效。

这项工作有望让浅层神经网络应用于许多计算敏感型应用，为浅层神经网络提供优于梯度下降算法的优化能力。

众所周知，树集成（如提升树和随机森林）可以映射为双层神经网络 [20,21]。因此，这项工作有望为通过损失最小化来训练此类深度集成带来更好的算法，使其更善于最小化损失，且计算效率更高。

然而，该方法存在一定局限，因为它需要对一个大小为 $(d+1)h \times (d+1)h$ 的矩阵求逆。当输入维度和神经元数量都很大时（例如 $dh > 30,000$），该方法的适用性可能会受到限制。稀疏神经元连接（并非所有神经元都与所有输入相连）或使用多个图形处理器的大规模实现，或可解决这一缺点。

未来，我们计划将 ANMIN 方法应用于使用神经网络进行形状分析，以及大规模目标检测应用。

资助： 本研究未获得外部资助。

数据可用性声明： 实验中使用了以下公开数据集：来自加州大学欧文分校机器学习知识库的鲍鱼、共享单车和年份预测MSD [16],魏茨曼马匹数据集 [18], 以及伯克利数据集 [19]。去噪自编码器和SDF 解码器数据可在以下地址找到： https://github.com/barbua/ANMIN/tree/main/data。

利益冲突： 作者声明不存在利益冲突。

缩略语

本文使用了以下缩略语：

ANMIN 解析最小化算法 BFGS Broyden–Fletcher–Goldfarb–Shanno算法 CNN 卷积神经网络 DAE 去噪自编码器 LBFGS 有限内存BFGS算法 MSE 均方误差 NN 神经网络 OLS 普通最小二乘法 SGD 随机梯度下降 ReLU 修正线性单元

参考文献

1. Kingma, D.P.; Ba, J. Adam：一种随机优化方法。*arXiv*预印本 *arXiv:1412.6980* **2014**。2. Sun, R.; Li, D.; Liang, S.; Ding, T.; Srikant, R. 神经网络的全局景观：综述。*IEEE*信号处理杂志 **2020**，*37*，95–108。3. Zhou, Y.; Liang, Y. 线性神经网络的临界点：解析形式与景观性质。ICLR，2018。4. Gopalani, P.; Mukherjee, A. 双层神经网络中随机梯度下降的全局收敛。*arXiv*预印本 *arXiv:2210.11452* **2022**。5. Villani, C. 次强制性。*arXiv*预印本 *math/0609050* **2006**。6. Shi, B.; Su, W.J.; Jordan, M.I. 关于学习率与薛定谔算子。*arXiv*预印本 *arXiv:2004.06977* **2020**。7. Zhang, Y.; Liang, P.; Wainwright, M.J. 凸化卷积神经网络。国际机器学习会议。PMLR，2017，第 4044–4053 页。8. Pilanci, M.; Ergen, T. 神经网络是凸正则化器：适用于双层网络的精确多项式时间凸优化公式。ICML，2020，第 7695–7705 页。9. Mishkin, A.; Sahiner, A.; Pilanci, M. 双层ReLU网络的快速凸优化：等价模型类与锥分解。国际机器学习会议。PMLR，2022，第 15770–15816 页。10. Bakshi, A.; Jayaram, R.; Woodruff, D.P. 在多项式时间内学习双层修正神经网络。学习理论会议。PMLR，2019，第 195–268 页。11. Goel, S.; Klivans, A.R. 在多项式时间内学习具有两个非线性层的神经网络。学习理论会议。PMLR，2019，第 1470–1499 页。12. Awasthi, P.; Tang, A.; Vijayaraghavan, A. 学习具有通用ReLU激活的深度为2的神经网络的高效算法。神经信息处理系统进展 **2021**，*34*，13485–13496。13. Rolnick, D.; Kording, K. 深度ReLU网络逆向工程。国际机器学习会议。PMLR，2020，第 8178–8187 页。14. Fletcher, R. 优化实用方法；约翰·威立父子出版公司，2013。15. Liu, D.C.; Nocedal, J. 关于用于大规模优化的有限内存BFGS方法。数学规划 **1989**，*45*，503–528。16. Dua, D.; Graff, C. UCI机器学习资料库，2017。17. Park, J.J.; Florence, P.; Straub, J.; Newcombe, R.; Lovegrove, S. DeepSDF：学习用于形状表示的连续有符号距离函数。CVPR，2019，第 165–174 页。18. Borenstein, E.; Sharon, E.; Ullman, S. 结合自顶向下与自底向上分割。CVPR研讨会。IEEE，2004，第 46–46 页。19. Martin, D.; Fowlkes, C.; Tal, D.; Malik, J. 人工分割的自然图像数据库及其在评估分割算法和测量生态统计量中的应用。ICCV，2001，第 2 卷，第 416–423 页。20. Sethi, I.K. 使用人工神经网络实现来提升决策树性能。载于机器智能与模式识别；Elsevier，1991；第 11 卷，第 71–88 页。21. Welbl, J. 将随机森林转化为人工神经网络（并从中获益）。德国模式识别会议。Springer，2014，第 765–771 页。

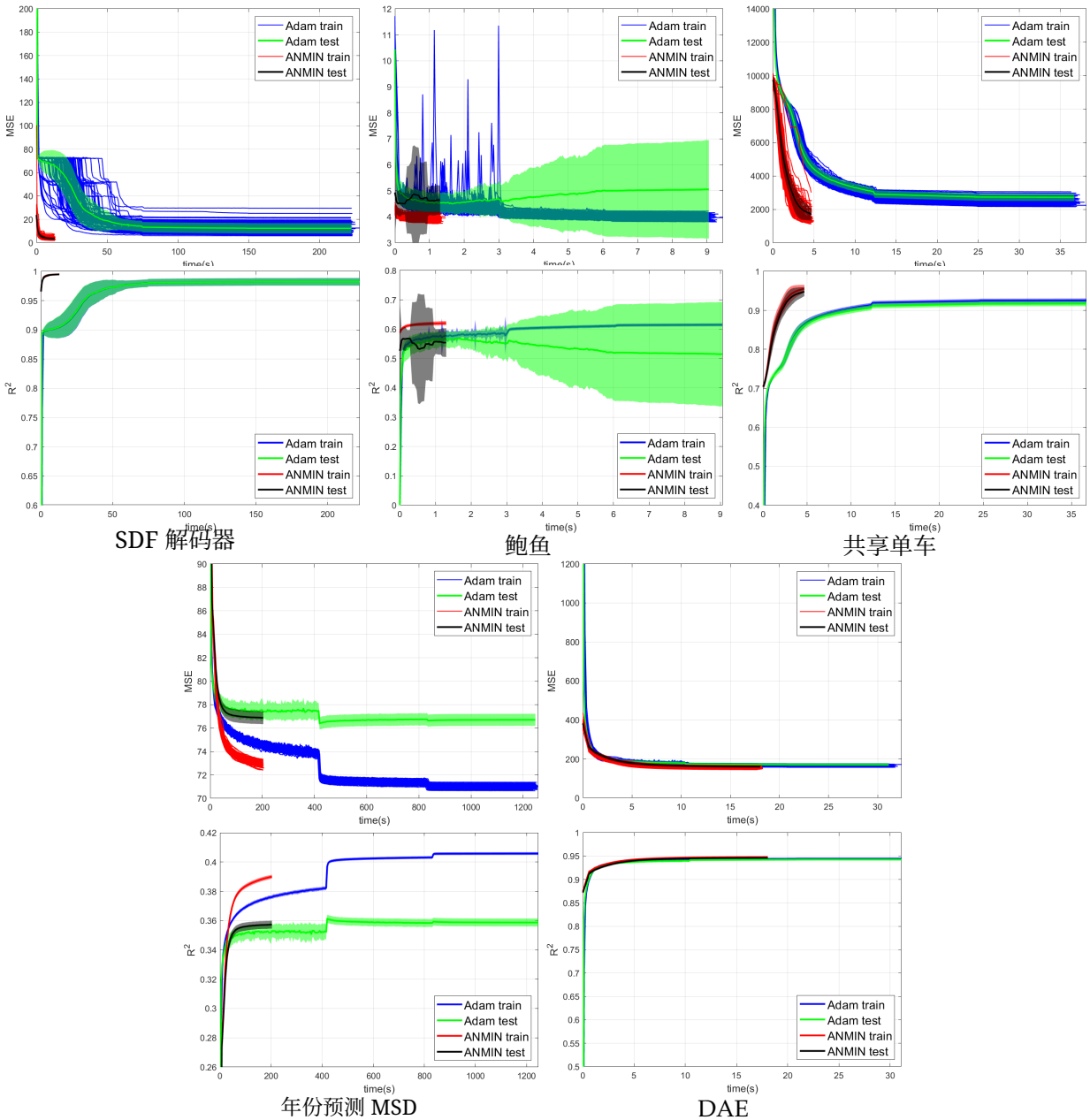


图4。 使用 Adam 和 ANMIN 优化器训练带有 ReLU 激活的神经网络 100 次时，均方误差和平均 R^2 随时间（秒）的变化。还绘制了测试均方误差的均值和 R^2 及其标准差。

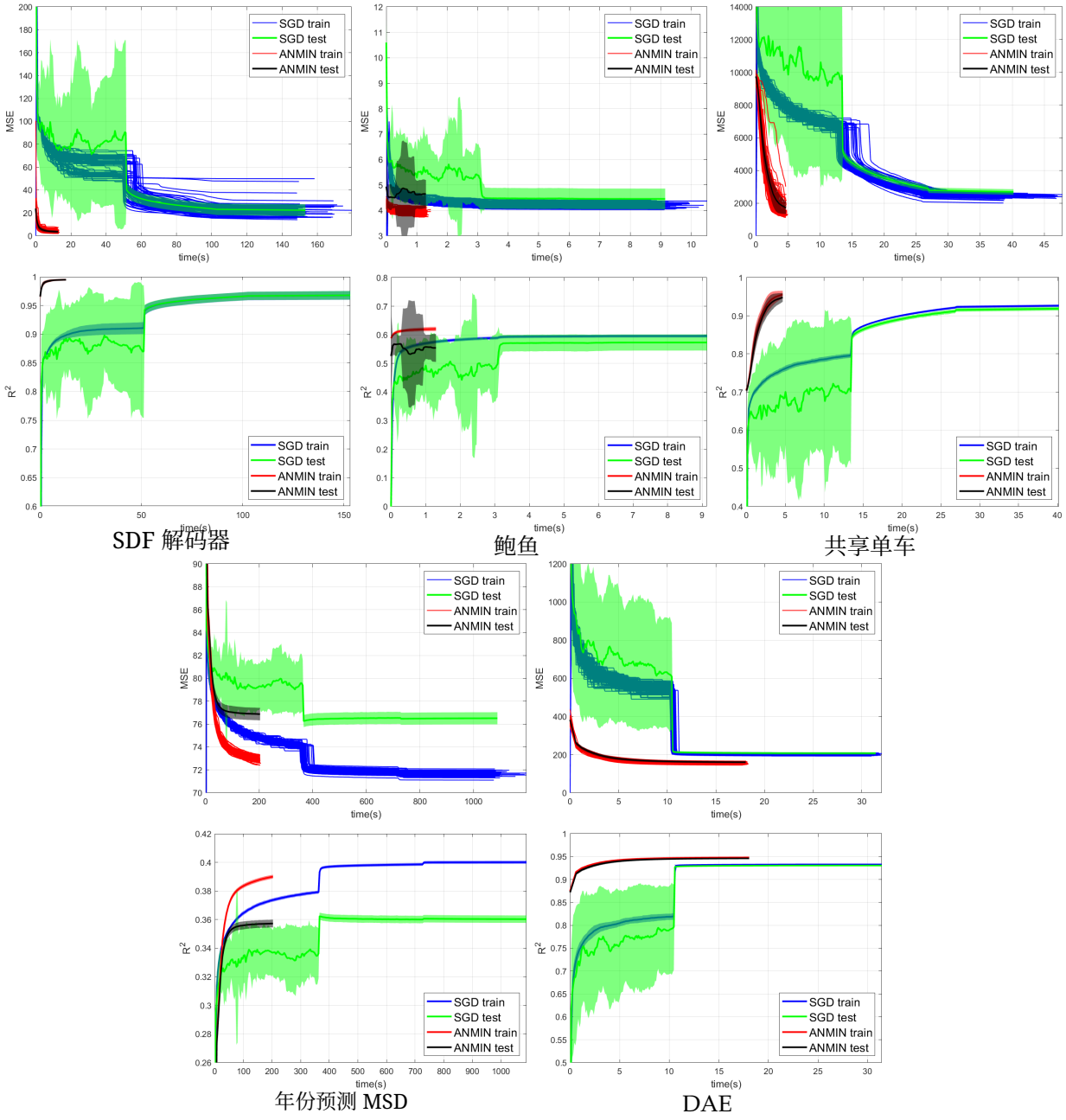


图 5. 使用随机梯度下降和 ANMIN 优化器，通过 ReLU 激活函数训练神经网络 100 次所得的均方误差和平均 R^2 随时间（秒）的变化。还绘制了测试均方误差和 R^2 的均值及标准差。

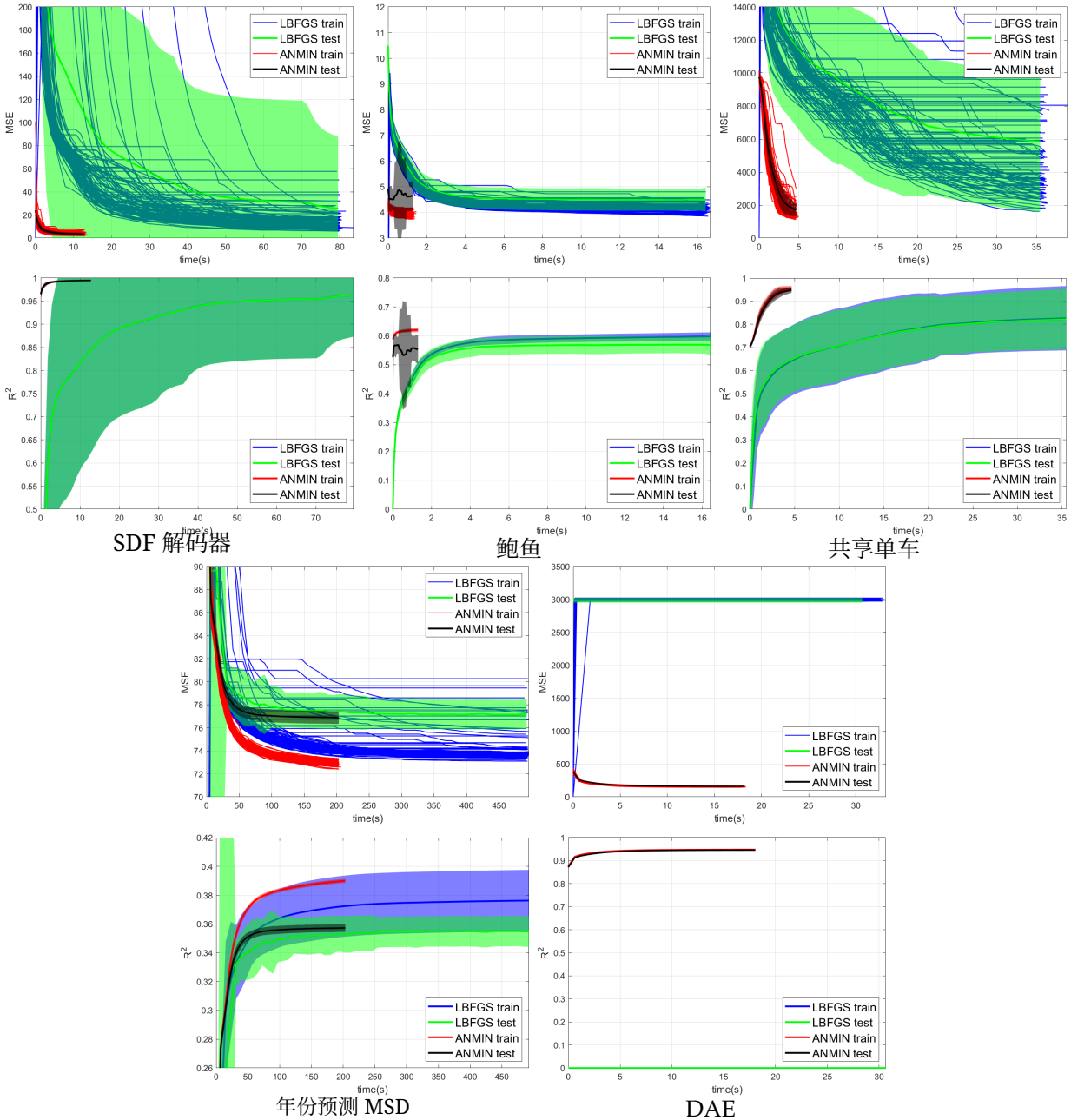
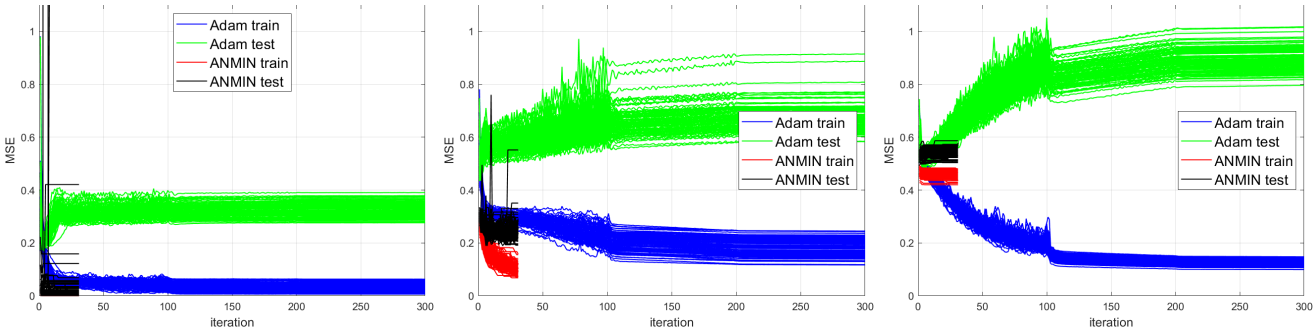
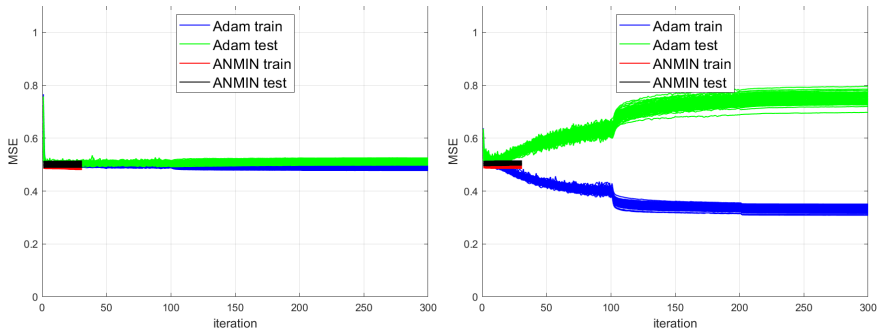


图 6。 使用 L-BFGS 和 ANMIN 优化器，以 ReLU 激活训练神经网络进行 100 次运行的均方误差和平均 R^2 随时间（秒）的变化。还绘制了平均测试均方误差和 R^2 及其标准差。

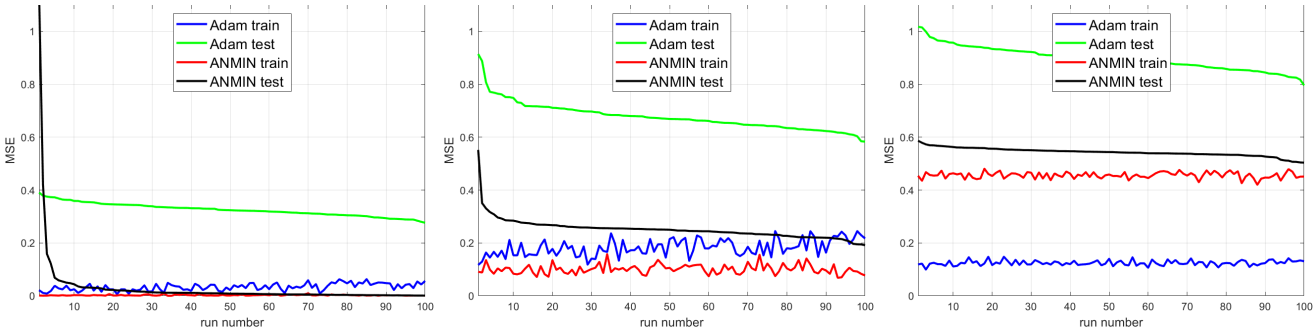


$N = 1{,}000, d = 1$ $N = 1{,}000, d = 3$ $N = 1{,}000, d = 10$

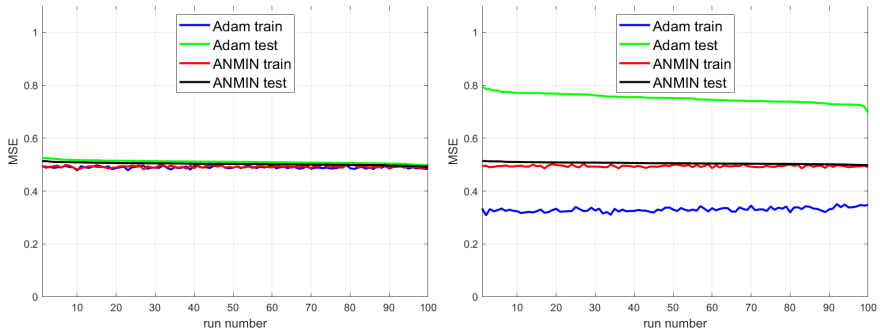


$N = 10{,}000, d = 10$ $N = 10{,}000, d = 30$

图 7。 仿真实验。使用 Adam 优化器的 300 个训练轮次或 ANMIN的 30 次迭代次数，训练一个具有64 个隐藏节点的神经网络，进行 100 次运行所得的均方误差（算法 1）。

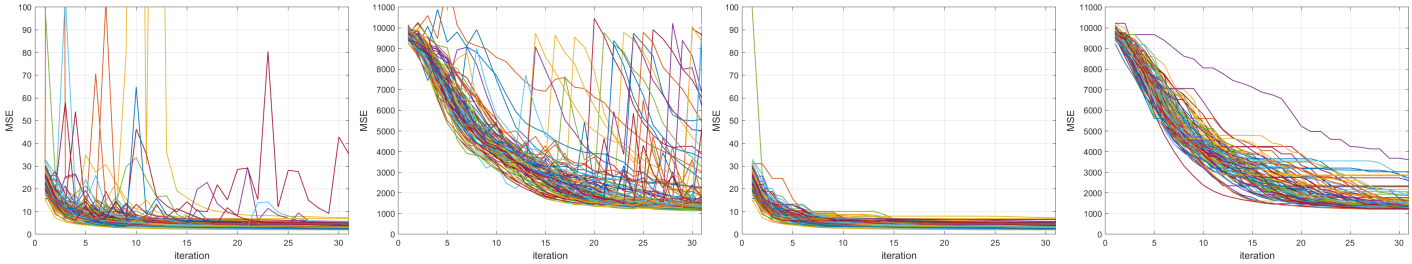


$N = 1{,}000, d = 1$ $N = 1{,}000, d = 3$ $N = 1{,}000, d = 10$



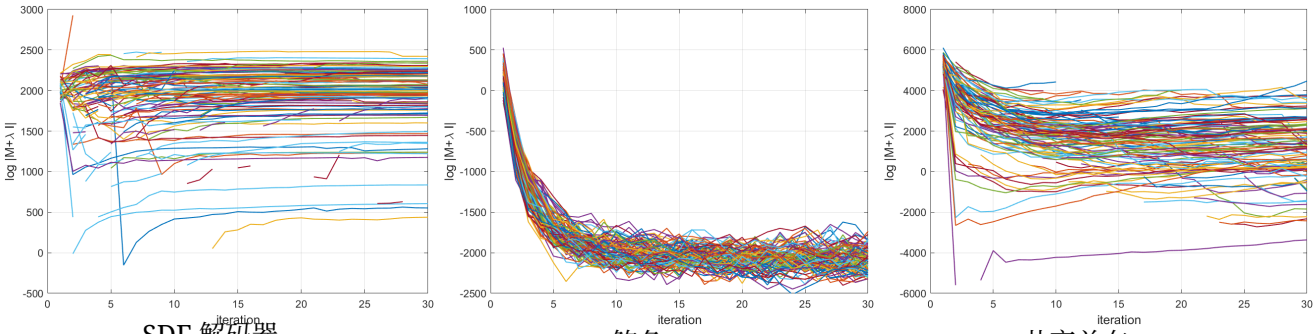
$N = 10{,}000, d = 10$ $N = 10{,}000, d = 30$

图8. 仿真实验。Adam 和 ANMIN 在 100 次运行中得到的最终训练和测试均方误差，按测试均方误差降序排列。

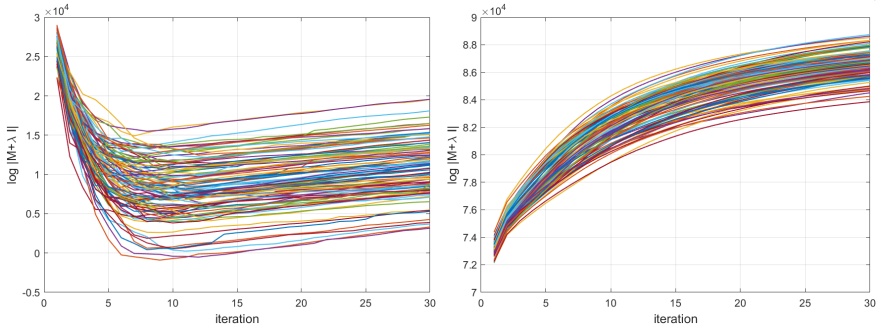


SDF 解码器　　共享单车　　SDF 解码器　　共享单车

图 9。使用 ANMIN（算法 1）进行 30 次迭代训练一个包含 64 个隐藏节点的神经网络时，100 次训练运行的均方误差随时间（秒）变化的图。左侧两个图：每次迭代中实际遇到的均方误差值。右侧两个图：截至每次迭代所获得的最小均方误差。



SDF 解码器　　鲍鱼　　共享单车



年份预测 MSD　　DAE

图 10. ln 的图 $|\mathbf{M} + \lambda \mathbf{I}(d+1)h|$ 100 次 ANMIN（算法 1）运行中神经网络（NN）使用 ReLU 激活函数时的迭代次数对比。

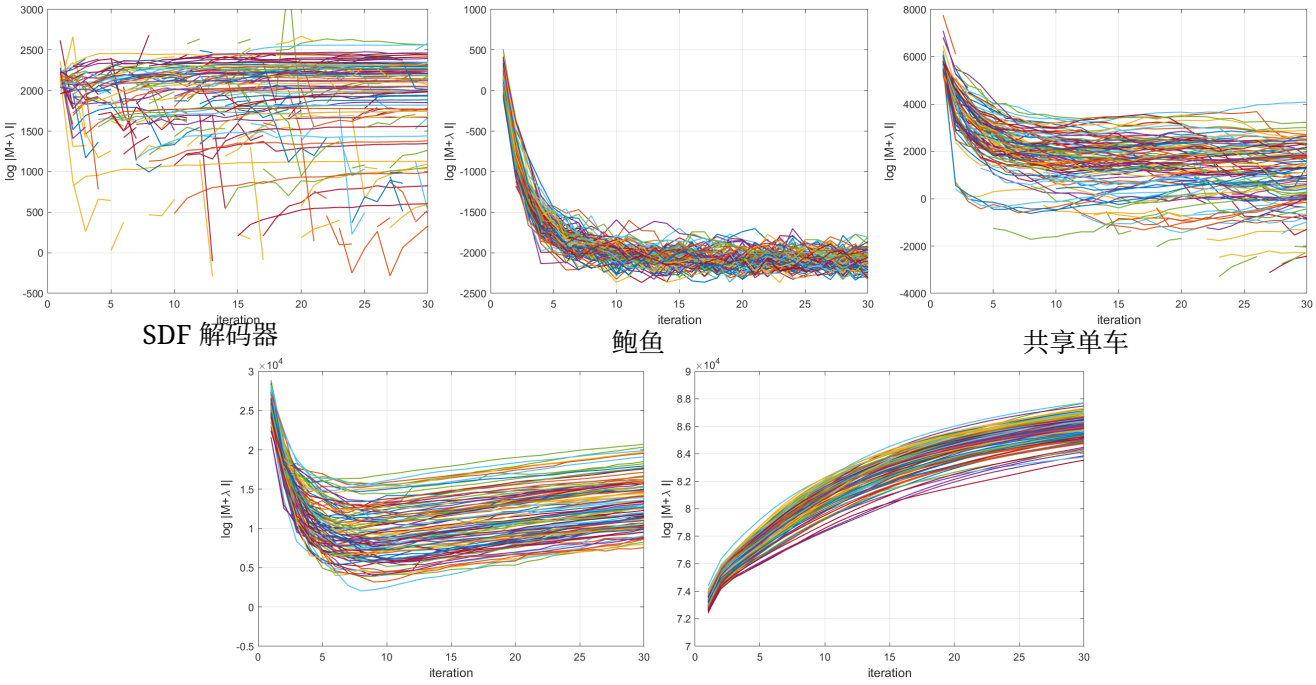


图11。 $\ln|\mathbf{M}+\lambda\mathbf{I}_{(d+1)h}|$ 与迭代次数的关系，针对使用Leaky ReLU 激活的神经网络，运行100次 ANMIN（算法 1）。