随机列文伯格-马夸尔特算法在神经网络中的应用

周宏宇

（数学系　指导教师：杨将）

[摘要]：列文伯格-马夸尔特（LM）算法最初是为求解最小二乘问题而提出，并已被广泛应用于神经网络训练中。然而，该方法由于在大规模的神经网络问题中计算成本极高，使其应用范围收到了限制。本文提出了随机列文伯格-马夸尔特（SLM）算法，在每次迭代中通过随机选择参数子集进行更新。收敛性分析表明，SLM 方法具有全局收敛性。实验结果进一步验证，SLM 能够在大幅降低训练时间的同时实现与传统 LM 算法相当的精度表现。

[关键词]：列文伯格算法，神经网络学习算法，最小二乘问题

[**ABSTRACT**]: The Levenberg–Marquardt (LM) algorithm was originally designed for solving least squares problems and has been widely applied to neural network training. However, its computational cost becomes prohibitively high, especially for large-scale neural network problems. In this paper, we propose the Stochastic Levenberg–Marquardt (SLM) algorithm, which updates parameters based on randomly selected parameter subsets at each iteration. Convergence analysis establishes that SLM possesses a global convergence property under some basic conditions. Experimental results further demonstrate that SLM achieves accuracy comparable to the traditional LM algorithm while significantly reducing training time.

[**Key words**]: Levenberg–Marquardt algorithm, neural network training algorithm, least square problem

Content

1. Introduction

Artificial neural network (ANN) has become an effective tool for solving complex problems in many different fields. Thanks to their flexibility, ANN has shown great success in both theory and real-world applications, including areas like chemistry, physics, biology, and finance [1, 2, 3, 4]. Furthermore, ANN like physics-informed neural networks has also been used for solving partial difference equations [5, 6, 7].

Despite their success, one of the main challenges in utilizing ANN lies in the optimization process. In 1986, Rumelhart et al. [8] introduced the back-propagation (BP) algorithm, enabling the training of ANN using standard optimization techniques. Among various strategies, the *least squares method* has been widely adopted due to its direct minimization of residual error. To address the difficulties in solving nonlinear least squares problems, numerous algorithms have been proposed. One of the methods is the Levenberg–Marquardt (LM) algorithm, originally proposed by Marquardt [9] and subsequently improved and applied in various contexts [10, 11, 12]. The LM algorithm is a self-adaptive second-order optimization method that combines the Gauss–Newton (GN) method and gradient descent (GD), aiming to leverage the strengths of both approaches. It has been shown to achieve global and even quadratic convergence under certain conditions [13]. Moreover, with appropriate modifications, LM demonstrates strong stability and robustness in training artificial neural networks (ANN) [14, 15].

In this paper, we propose a stochastic variant of the LM method, where each iteration relies only on a randomly selected subset of the full parameter space. This approach is motivated by the high computational cost of computing derivatives and performing matrix inversion in classical LM, especially for large-scale ANN. Our method is inspired by subsampled Newton-type approaches and techniques developed for probabilistic models [16, 17, 18, 19].

The structure of this paper is as follows. In Section 2, we provide a brief overview of artificial neural networks (ANN), as well as the gradient descent (GD) and Gauss–Newton (GN) methods. In Section 3, we introduce the proposed stochastic Levenberg–Marquardt method (SLM). Section 4 presents the global convergence analysis, and numerical experiments are provided in Section 5. Finally, conclusions are drawn in Section 6.

2. Artificial Neural Network

2.1. fully connected neural networks

In recent years, researchers have proposed a variety of neural network architectures to address different tasks in deep learning [20, 21, 22, 23]. Among them, neural networks are often viewed as universal function approximators. A neural network can generally be represented as a parameterized function:

$$\varphi(x; \theta) \tag{2.1}$$

where $x \in \mathbb{R}^d$ is the input and θ denotes the trainable parameters of the network (weights and bias).

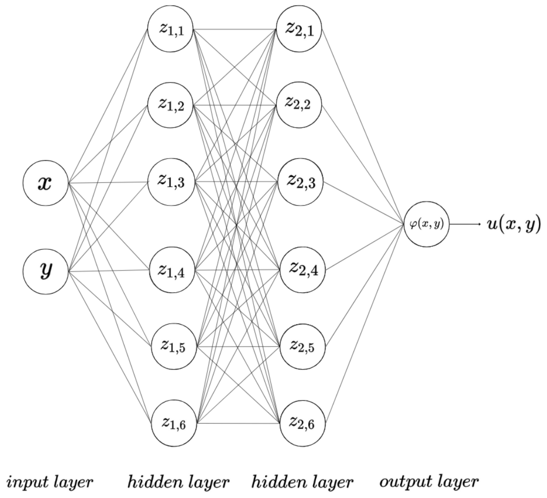


Figure 1 Fully connected neural network (FCNN) architecture with two hidden layers

One of the most widely used architectures is the *fully connected neural network* (FCNN). In an FCNN, each neuron in one layer is connected to every neuron in the next layer. The architecture typically consists of an input layer, one or more hidden layers, and an output layer, with each hidden layer followed by a nonlinear activation function such as the ReLU ($y(z) = \max(0, z)$) or the tanh function ($y(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$) to introduce nonlinearity and improve expressive power.

For example, to approximate a two-dimensional function $u(x, y) : (x, y) \in \mathbb{R}^2 \to \mathbb{R}$, a fully connected network with two hidden layers can be visualized as in Fig. 1. In such a network, the input layer accepts the coordinates (x, y), the hidden layers transform the representation through learned weights and nonlinearities, and the output layer produces a scalar value approximating $u(x, y)$.

2.2. the gradient descent method

The gradient descent (GD) method is one of the most widely used optimization algorithms in deep learning due to its simplicity and effectiveness [24]. At each iteration, GD updates the parameters based on the gradient of the loss function computed over the entire dataset:

$$\theta^{k+1} = \theta^k - \eta \nabla_\theta \left\| \varphi(\theta^k) - \overline{\varphi} \right\|^2, \tag{2.2}$$

where $\left\| \varphi(\theta^k) - \overline{\varphi} \right\|^2$ denotes the least squares loss, $\overline{\varphi}$ represents the target values, and η is the learning rate. Strategies for choosing and adapting η can be found in [25].

In practical applications, stochastic gradient descent (SGD) [26] is typically employed, where the gradient is approximated using a small mini-batch of data, leading to faster iterations and better scalability. Various improved variants of GD, such as momentum methods, RMSProp, and Adam, have also been proposed to accelerate convergence and improve training stability. A comprehensive overview of these variants is provided in [27].

2.3. the Gauss–Newton method

The Gauss–Newton (GN) method is a popular used technique for solving nonlinear least squares problems without explicitly computing the full Hessian matrix [28]. Based on the first-order Taylor expansion of $\varphi(\theta + h)$, the goal of each iteration is to find a direction h such that the gradient of the squared residual norm vanishes, i.e.,

$$\frac{\partial}{\partial h} \left\| \varphi(\theta + h) - \overline{\varphi} \right\|^2 = 0.$$

Using the linear approximation $\varphi(\theta + h) \approx \varphi(\theta) + J(\theta)h$, where $J(\theta)$ is the Jacobian of φ, we obtain the gradient:

$$\frac{\partial}{\partial h} \|\varphi(\theta + h) - \overline{\varphi}\|^2 \approx 2J(\theta)^\top (\varphi(\theta) + J(\theta)h - \overline{\varphi}) .$$

Setting the gradient to zero yields the linear system:

$$J(\theta)^\top J(\theta)h = J(\theta)^\top (\overline{\varphi} - \varphi(\theta)),$$

which defines the update direction h at each iteration. One of the main appeals of the Gauss–Newton method lies in its approximation of the Hessian matrix by $J(\theta)^\top J(\theta)$. Consequently, most of Newton-type methods differ primarily in how they approximate or regularize the Hessian.

3. Methodology

Notations. Let $\|\cdot\|$ denote the L_2 norm. Let $\|S\|$ denote the cardinality of a set S, i.e., the number of elements in S. Let $\mathcal{F} \subseteq \{1, \dots, d\}$ denote an index subset of the parameter space. For a vector $x \in \mathbb{R}^d$, we use $x_{\mathcal{F}}$ to denote the subvector of x indexed by $\mathcal{F}$, whose dimension is $|\mathcal{F}|$. Let $\mathbb{P}$ denote the probability measure defined on the basic probability space.

The general nonlinear least squares problem can be considered as follows:

$$\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{2}\|r(x)\|^2 = \frac{1}{2}\sum_{i=1}^{M} |r_i(x)|^2 \,, \tag{3.1}$$

where $r(x) : \mathbb{R}^d \to \mathbb{R}^M$ is a continuously differentiable residual function. Throughout this paper, we primarily consider the case where $r(x)$ represents the output of a neural network system.

3.1. LM algorithm

The core idea of the classical Levenberg–Marquardt (LM) method is to approximate the Hessian of the objective function by

$$\mathbf{H}(x) = \sum_{i=1}^{M} \nabla r_i(x) \nabla r_i(x)^\top = J(x)^\top J(x), \tag{3.2}$$

where $J(x)$ denotes the Jacobian matrix of the residual vector $r(x) \in \mathbb{R}^M$. At each iteration, the LM method solves the following linear system to compute an update direction:

$$h^k = -\left(\mathbf{H}(x^k) + \mu^k \mathbf{I}\right)^{-1} g(x^k), \tag{3.3}$$

where $g(x^k) = \sum_{i=1}^{M} \nabla r_i(x^k) r_i(x^k) = J(x^k)^\top r(x^k)$ is the gradient of the objective function and $\mu^k > 0$ is the regularization parameter. Therefore, it is easy to show that $\mathbf{H}(x^k) + \mu^k \mathbf{I}$ is always positive definite. As a result, the inverse matrix always exists. Intuitively, when μ^k becomes larger, the update step behaves more like gradient descent; when μ^k becomes smaller, it resembles a Gauss–Newton step. To interpret the update step from another perspective, we define the following quadratic model:

$$m_k(x^k + h) := f(x^k) + g(x^k)^\top h + \frac{1}{2}h^\top \mathbf{H}(x^k)h + \frac{1}{2}\mu^k\|h\|^2. \tag{3.4}$$

This model serves as a local approximation of the objective function around the current step. The effectiveness of the proposed step h^k is evaluated using the ratio

$$\rho_k = \frac{f(x^k) - f(x^k + h^k)}{m_k(x^k) - m_k(x^k + h^k)}, \tag{3.5}$$

which measures the agreement between the actual and predicted reductions. This ratio can be interpreted as a confidence indicator of the step's efficiency. When ρ_k is close to 1, the model is considered reliable, and the regularization parameter μ^k is decreased. Conversely, a small ρ_k suggests that the model is a poor approximation of the objective function, and μ^k is increased, following a strategy similar to that of the trust region method which can be found in [29, 30, 31]. From another perspective, the LM step h^k can be viewed as the solution to the following subproblem:

$$h^k = \arg\min_h m_k(x^k + h), \tag{3.6}$$

which balances descent direction quality and regularization to ensure stability and convergence. Then the Levenberg–Marquardt algorithm can be described as:

Algorithm 1 LM Algorithm

```
1: Initialize: $x^0 \in \mathbb{R}^n$, constants $\mu^0 > 0$, $\mu_{\text{min}} > 0$, $\gamma > 1$, $\eta_1 > 0$, $\eta_2 > 0$, iteration count $k = 0$
2: while not converged do
3:     Compute step $h^k$ and ratio $\rho_k$
4:     if $\rho_k \geq \eta_1$ and $\|g(x^k)\|^2 \geq \eta_2/\mu^k$ then
5:         $x^{k+1} \leftarrow x^k + h^k$
6:         $\mu^{k+1} \leftarrow \max\{\mu^k/\gamma, \mu_{\text{min}}\}$
7:     else
8:         $x^{k+1} \leftarrow x^k$
9:         $\mu^{k+1} \leftarrow \gamma\mu^k$
10:    end if
11:    $k \leftarrow k + 1$
12: end while
```

3.2. Stochastic LM algorithm

To introduce the proposed Stochastic Levenberg–Marquardt (SLM) method, we begin with a brief description of its core idea. In this algorithm, only the derivatives with respect to a randomly selected parameter subset are required to perform each update step. To ensure convergence, we define a parameter β, which represents the minimum size of the parameter subset that must be used at each iteration.

Our method are presented as follows:

Algorithm 2 Stochastic LM Algorithm

```
1: **Initialize:** $x^0 \in \mathbb{R}^n$, constants $\mu^0 > 0$, $\mu_{\text{min}} > 0$, $\gamma > 1$, $\eta_1 > 0$, $\eta_2 > 0$, iteration count $k = 0$
2: while not converged do
3:     Randomly choose parameter subset $\mathcal{F}$ with $|\mathcal{F}| \geq \beta$
4:     Compute step $h^k_{\mathcal{F}}$ and ratio $\rho_k$
5:     if $\rho_k \geq \eta_1$ and $\|g(x^k_{\mathcal{F}})\|^2 \geq \eta_2/\mu^k$ then
6:         $x^{k+1} \leftarrow x^k + h^k_{\mathcal{F}}$
7:         $\mu^{k+1} \leftarrow \max\left\{\mu^k/\gamma, \mu_{\text{min}}\right\}$
8:     else
9:         $x^{k+1} \leftarrow x^k$
10:        $\mu^{k+1} \leftarrow \gamma\mu^k$
11:    end if
12:    $k \leftarrow k + 1$
13: end while
```

At the same time, we provide a strategy for computing the parameter β at each iteration, with details given in Section 4 and Section 5.

3.3. update the Jacobian matrix

Among the operations in the Levenberg–Marquardt algorithm (LM), one of the most computationally expensive steps is obtaining the Jacobian matrix, especially for large-scale models or models without explicit derivatives. In classical neural networks, the Jacobian with respect to weights and biases is typically computed using the backpropagation technique. In this section, we introduce a numerical method for approximating the Jacobian matrix which proposed by Broyden [32] and we extend it to the LM algorithm (LMB) and stochastic LM algorithm (SLMB). Briefly, the update step is given by:

$$J(x^k) = J(x^{k-1}) + \frac{r(x^k) - r(x^{k-1}) - J(x^{k-1})h^{k-1}}{\|h^{k-1}\|^2}(h^{k-1})^\top \tag{3.7}$$

Thus, the Jacobian matrix at each step can be approximated based solely on the residual outputs, the previous Jacobian matrix, and the update direction derived from Eq. 3.3. Furthermore, for the stochastic LM algorithm, the Jacobian update can be performed as follows:

$$J(x^k_{\mathcal{F}}) = J(x^{k-1}_{\mathcal{F}}) + \frac{r(x^k) - r(x^{k-1}) - J(x^{k-1}_{\mathcal{F}})h^{k-1}_{\mathcal{F}}}{\|h^{k-1}_{\mathcal{F}}\|^2}(h^{k-1}_{\mathcal{F}})^\top \tag{3.8}$$

However, after a few iteration, the approximation does not work. So in practice, we recommend recomputing the exact Jacobian matrix whenever an iteration is rejected.

4. Convergence Analysis

In this section, we give our convergence proof of the proposed method. Initially, we present the relevant assumptions necessary for establishing the result.

For clarity, we repeat some of the notation introduced earlier. To solve the nonlinear problem, we consider the following quadratic model:

$$m_k(x_{\mathcal{F}}^k + h) := f(x_{\mathcal{F}}^k) + g^\top(x_{\mathcal{F}}^k)h + \frac{1}{2}h^\top \mathbf{H}(x_{\mathcal{F}}^k)h + \frac{1}{2}\mu^k\|h\|^2 \tag{4.1}$$

The ratio used in evaluating the success step is defined as:

$$\rho_k = \frac{f(x^k) - f(x^k + h^k)}{m_k(x^k) - m_k(x^k + h^k)}, \tag{4.2}$$

4.1. assumption

Assumption 4.1. *Suppose τ^k is the solution of* $\arg\min_h m_k(x_{\mathcal{F}}^k + h)$*, then the following inequality holds:*

$$m_k(x_{\mathcal{F}}^k) - m_k(x_{\mathcal{F}}^k + \tau^k) \geq \frac{1}{4}\|g(x_{\mathcal{F}}^k)\|^2 \min\left\{\frac{1}{\mu^k}, \frac{1}{\|\mathbf{H}(x_{\mathcal{F}}^k)\|}\right\} \tag{4.3}$$

and

$$\|\tau^k\| \leq \frac{2\|g(x_{\mathcal{F}}^k)\|}{\mu^k} \tag{4.4}$$

We find that assumption 4.1 characterizes the decrease and stability of the model m_k when optimizing over a random parameter subset. It parallels the classical Cauchy step condition in trust-region methods, which ensures a minimum model reduction along the steepest descent direction.

Assumption 4.2. *Assume that the residual functions $r_i(x)$ are continuously differentiable, and their gradients $\nabla r_i(x)$ are Lipschitz continuous with constant $L > 0$, i.e.,*

$$\|\nabla r_i(x) - \nabla r_i(y)\| \leq L\|x - y\|, \quad \textit{for all } x, y \in \mathbb{R}^n.$$

Furthermore, suppose that the objective function $f(x)$ is bounded and the approximate Hessian $\mathbf{H}(x)$ is uniformly bounded above by a constant $c \geq 0$, i.e.,

$$\|\mathbf{H}(x)\| \leq c, \quad \textit{for all } x \in \mathbb{R}^n. \tag{4.5}$$

4.2. first order global convergence

The following lemma states that if the algorithm has iterated for a sufficient number of steps. Then the parameters μ^k will become almost infinitely large.

Lemma 4.1. *if Assumption 4.1 and 4.2 holds. Then almost surely $\mu^k \geq \kappa$ for any $\kappa \geq 0$.*

Proof. We proceed by contradiction to establish the lemma.For any constant $\kappa \geq 0$. Assume the set $\left\{k|\mu^k \leq \kappa\right\}$ is infinite with a positive probability α, also can conclude

$$\mathbb{P}(\left\{k|\mu^k \leq \kappa\right\} = \infty) = \alpha > 0 \tag{4.6}$$

Define the set $\mathcal{S} = \{k \mid \text{the } k\text{-th iteration is successful}\}$. As described in Algorithm 2, Eq. (4.6) implies that the parameter is decreased infinitely often with probability α, corresponding to infinitely many successful iterations.

According to the definition of ρ_k and the condition

$$\rho_k \geq \eta_1 \tag{4.7}$$

holds whenever the step is accepted. Applying Assumption 4.1, we can deduce that

$$\begin{aligned}\sum_{k\in\mathcal{S}} f(x_{\mathcal{F}}^k) - f(x_{\mathcal{F}}^k + h_{\mathcal{F}}^k) &\geq \eta_1 \sum_{k\in\mathcal{S}} m_k(x_{\mathcal{F}}^k) - m_k(x_{\mathcal{F}}^k + h_{\mathcal{F}}^k) \\ &\geq \sum_{k\in\mathcal{S}} \frac{\eta_1}{4} \left\|g(x_F^k)\right\|^2 \min\left\{\frac{1}{\mu^{k'}}, \frac{1}{\left\|\mathbf{H}\left(x_F^k\right)\right\|}\right\} \\ &\geq \sum_{k\in\mathcal{S}} \frac{\eta_1\eta_2}{4\mu^k} \min\left\{\frac{1}{\mu^{k'}}, \frac{1}{\left\|\mathbf{H}\left(x_F^k\right)\right\|}\right\} \\ &\geq \sum_{k\in\mathcal{S}} \frac{\eta_1\eta_2}{4\kappa} \min\left\{\frac{1}{\kappa}, \frac{1}{c}\right\}\end{aligned} \tag{4.8}$$

where we successively use accept step condition, Assumption 4.2, Eq. (4.6). Notice that the expectation

$$\mathbb{E}\left[\sum_{k\in\mathcal{S}}(f(x_{\mathcal{F}}^k)-f(x_{\mathcal{F}}^k+h_{\mathcal{F}}^k))\right]\geq|\mathcal{S}|\frac{\eta_1\eta_2}{4\kappa}\min\left\{\frac{1}{\kappa},\frac{1}{c}\right\} \tag{4.9}$$

and the right-hand side diverges to infinity. Hence,

$$\infty=\mathbb{E}\left[\sum_{k\in\mathcal{S}}\left(f(x_{\mathcal{F}}^k)-f(x_{\mathcal{F}}^k+h_{\mathcal{F}}^k)\right)\right]\leq\mathbb{E}\left[2f(x_{\mathcal{F}}^k)\right]. \tag{4.10}$$

However, Assumption 4.2 states that $f(x)$ is bounded, which leads to a contradiction with Eq. (4.10). In conclusion, we can almost surely $\mu^k\geq\kappa$ for any $\kappa\geq 0$. □

Lemma 4.2. *By Lemma 4.1, the parameter μ^k is almost surely sufficiently large after a finite number of iterations. In particular, if*

$$\mu^k\geq\max\left\{c,\frac{8(L+2c)}{1-\eta_1}\right\} \tag{4.11}$$

then the ratio satisfies $\rho_k>\eta_1$.

Proof. We begin by applying Assumption 4.2, which states that f is L-smooth. This yields the well-known Descent Lemma:

$$f(x_{\mathcal{F}}^k+h_{\mathcal{F}}^k)\leq f(x_{\mathcal{F}}^k)+g(x_{\mathcal{F}}^k)^\top h_{\mathcal{F}}^k+\frac{L}{2}\|h_{\mathcal{F}}^k\|^2. \tag{4.12}$$

Since $f(x_{\mathcal{F}}^k)=m_k(x_{\mathcal{F}}^k)$, we substitute this into Eq. (4.12) and rearrange the terms. Combining Assumption 4.2 and Assumption 4.1, we obtain:

$$\begin{aligned}f(x_{\mathcal{F}}^k+h_{\mathcal{F}}^k)-m_k(x_{\mathcal{F}}^k)-g(x_{\mathcal{F}}^k)^\top h_{\mathcal{F}}^k&=f(x_{\mathcal{F}}^k+h_{\mathcal{F}}^k)-m_k(x_{\mathcal{F}}^k+h_{\mathcal{F}}^k)\\&\quad+\frac{1}{2}{h_{\mathcal{F}}^k}^\top\mathbf{H}(x_{\mathcal{F}}^k)h_{\mathcal{F}}^k+\frac{1}{2}\mu^k\|h_{\mathcal{F}}^k\|^2.\end{aligned} \tag{4.13}$$

Rearranging the inequality, we have:

$$\begin{aligned}f(x_{\mathcal{F}}^k+h_{\mathcal{F}}^k)-m_k(x_{\mathcal{F}}^k+h_{\mathcal{F}}^k)&\leq\frac{L}{2}\|h_{\mathcal{F}}^k\|^2-\frac{1}{2}{h_{\mathcal{F}}^k}^\top\mathbf{H}(x_{\mathcal{F}}^k)h_{\mathcal{F}}^k-\frac{1}{2}\mu^k\|h_{\mathcal{F}}^k\|^2\\&\leq\frac{L+2c}{2}\|h_{\mathcal{F}}^k\|^2\\&\leq 2(L+2c)\cdot\frac{\|g(x_{\mathcal{F}}^k)\|^2}{(\mu^k)^2},\end{aligned} \tag{4.14}$$

where the last inequality follows from the Cauchy step bound. Now, using the definition of ρ_k, we have:

$$\begin{aligned}
1-\rho_k &= \frac{f(x^k_{\mathcal{F}}+h^k_{\mathcal{F}})-m_k(x^k_{\mathcal{F}}+h^k_{\mathcal{F}})}{m_k(x^k_{\mathcal{F}})-m_k(x^k_{\mathcal{F}}+h^k_{\mathcal{F}})} \\
&\le \frac{2(L+2c)\cdot\frac{\|g(x^k_{\mathcal{F}})\|^2}{(\mu^k)^2}}{\frac{1}{4}\|g(x^k_{\mathcal{F}})\|^2\min\left\{\frac{1}{\mu^k},\frac{1}{\|\mathbf{H}(x^k_{\mathcal{F}})\|}\right\}} \qquad (4.15)\\
&\le \frac{8(L+2c)}{\mu^k}.
\end{aligned}$$

It follows from Eq. (4.11) that

$$\rho_k \ge 1-\frac{8(L+2c)}{\mu^k}\ge \eta_1,$$

which concludes the proof. □

This lemma shows that for sufficiently large μ^k , the condition $\rho_k\ge\eta_1$ is always satisfied. Hence, the acceptance of the step just depends on the value of gradients $\|g(x^k_{\mathcal{F}})\|^2$ .We now establish the final lemma required for our convergence analysis. To guarantee convergence of the algorithm, we define the parameter β according to the following rule:

$$\beta := \begin{cases} d\sqrt{1-\frac{1}{2(\mu^k)^2\|g(x^k)\|^4}} & \text{if } \frac{1}{2(\mu^k)^2\|g(x^k)\|^4}\le 1, \\ 0 & \text{otherwise.}\end{cases} \qquad (4.16)$$

Lemma 4.3. *If the set $\mathcal{F}$ satisfies $|\mathcal{F}|\ge\beta$, then the event*

$$I_k := \left\{\left|\|g(x^k_{\mathcal{F}})\|^2-\frac{|\mathcal{F}|}{d}\|g(x^k)\|^2\right|<\frac{1}{\mu^k}\right\} \qquad (4.17)$$

occurs with probability $\mathbb{P}(I_k)>\frac{1}{2}$.

Proof. Define the set

$$\mathcal{F}_\delta := \{\mathcal{F}_\ell\,|\,\mathcal{F}_\ell\subseteq\{1,\dots,d\},\ |\mathcal{F}_\ell|=\delta\},$$

i.e., the collection of all subsets of size δ from the set $\{1,\dots,d\}$. It is clear that $|\mathcal{F}_\delta|=\binom{d}{\delta}$. Let $\mathcal{F}$ be a subset drawn uniformly at random from $\mathcal{F}_\delta$. We now compute the expectation and variance of $\|g(x^k_{\mathcal{F}})\|^2$.

Expectation.

$$
\begin{aligned}
\mathbb{E}\left[\|g(x^k_{\mathcal{F}})\|^2\right] &= \mathbb{E}\left[\sum_{i\in\mathcal{F}}(g_i(x^k))^2\right] \\
&= \sum_{\mathcal{F}_\ell\in\mathcal{F}_\delta}\left(\sum_{i\in\mathcal{F}_\ell}(g_i(x^k))^2\right)\cdot\mathbb{P}(\mathcal{F}=\mathcal{F}_\ell) \\
&= \sum_{\mathcal{F}_\ell\in\mathcal{F}_\delta}\left(\sum_{i\in\mathcal{F}_\ell}(g_i(x^k))^2\right)\cdot\frac{1}{\binom{d}{\delta}} \\
&= \frac{\delta}{d}\cdot\|g(x^k)\|^2.
\end{aligned}
\tag{4.18}
$$

Variance.

$$
\begin{aligned}
\text{Var}\left[\|g(x^k_{\mathcal{F}})\|^2\right] &= \mathbb{E}\left[\|g(x^k_{\mathcal{F}})\|^4\right] - \left(\mathbb{E}\left[\|g(x^k_{\mathcal{F}})\|^2\right]\right)^2 \\
&= \sum_{\mathcal{F}_\ell\in\mathcal{F}_\delta}\left(\sum_{i\in\mathcal{F}_\ell}(g_i(x^k))^2\right)^2\cdot\frac{1}{\binom{d}{\delta}} - \left(\frac{\delta}{d}\|g(x^k)\|^2\right)^2 \\
&\le \|g(x^k)\|^4 - \frac{\delta^2}{d^2}\|g(x^k)\|^4 \\
&= \left(1-\frac{\delta^2}{d^2}\right)\|g(x^k)\|^4 \\
&\le \left(1-\frac{\beta^2}{d^2}\right)\|g(x^k)\|^4.
\end{aligned}
\tag{4.19}
$$

It follows from Eq. (4.16),(4.18),(4.19) and Chebyshev's inequality that

$$
\begin{aligned}
\mathbb{P}\left(\left|\|g(x^k_{\mathcal{F}})\|^2 - \frac{\delta}{d}\|g(x^k)\|^2\right| \le \frac{1}{\mu^k}\right) &\ge 1-(\mu^k)^2\cdot\text{Var}\left[\|g(x^k_{\mathcal{F}})\|^2\right] \\
&\ge 1-(\mu^k)^2\left(1-\frac{\beta^2}{d^2}\right)\|g(x^k)\|^4 \\
&\ge \frac{1}{2}
\end{aligned}
\tag{4.20}
$$

□

Now the convergence proof of Algorithm 2 can be given by the following Theorem:

Theorem 4.1 (Global Convergence of Stochastic LM Algorithm)**.** *Under Assumptions 4.1 and 4.2, the sequence of parameters $\{x^k\}$ generated by Algorithm 2 converges almost surely* *to a stationary point. That is,*

$$\liminf_{k\to\infty} \|g(x^k)\| = 0 \tag{4.21}$$

Proof. We prove this theorem by contradiction using a submartingale argument. Suppose that for any integer $k_0 > 0$, there exists $\varepsilon > 0$ such that

$$\|g(x^k)\|^2 \geq \frac{d}{\delta}\varepsilon \quad \text{for all } k \geq k_0. \tag{4.22}$$

According to Lemma 4.1, for any integer $k_1 > 0$, there exists $k > k_1$ such that

$$\mu^k > \chi := \max\left\{\frac{2}{\varepsilon},\ \frac{2\eta_2}{\varepsilon},\ c,\ \frac{8(L+2c)}{1-\eta_1},\ \gamma\mu_{\min}\right\}. \tag{4.23}$$

Define

$$R_k := \log_\gamma\left(\frac{\chi}{\mu^k}\right).$$

By construction, we have $R_k \leq 0$ for all $k > \max(k_0, k_1)$. Since $\mu^k > \max\left\{c, \frac{8(L+2c)}{1-\eta_1}\right\}$, it follows that $\rho_k \geq \eta_1$, and hence the success of the iteration depends solely on $\|g(x^k_{\mathcal{F}})\|^2$. From Lemma 4.3, we know that with probability $v > \frac{1}{2}$,

$$\left|\|g(x^k_{\mathcal{F}})\|^2 - \frac{\delta}{d}\|g(x^k)\|^2\right| \leq \frac{1}{\mu^k}.$$

Using the assumption Eq.(4.22), we obtain

$$\|g(x^k_{\mathcal{F}})\|^2 \geq \frac{\varepsilon}{2}.$$

Moreover, since Eq.(4.23) implies $\frac{\eta_2}{\mu^k} < \frac{\varepsilon}{2}$, it follows that

$$\|g(x^k_{\mathcal{F}})\|^2 > \frac{\eta_2}{\mu^k},$$

which ensures that the iteration is successful. Now, consider the expected value of R_{k+1}:

$$
\begin{aligned}
\mathbb{E}[R_{k+1}] &= v \cdot \log_\gamma \left(\frac{\chi\gamma}{\mu^k} \right) + (1-v) \cdot \log_\gamma \left(\frac{\chi}{\mu^k \gamma} \right) \\
&= v \left(\log_\gamma \left(\frac{\chi}{\mu^k} \right) + 1 \right) + (1-v) \left(\log_\gamma \left(\frac{\chi}{\mu^k} \right) - 1 \right) \\
&= \log_\gamma \left(\frac{\chi}{\mu^k} \right) + (2v-1) = R_k + (2v-1) > R_k.
\end{aligned}
$$

In addition, we note that regardless of whether the iteration is successful or not, the quantity $|R_{k+1} - R_k| \geq 1$ due to the multiplicative update rule of μ^k by a factor of γ. Therefore, we conclude that

$$
\mathbb{P} \left(\limsup_{k \to \infty} R_k > 0 \right) = 1,
$$

which contradicts the fact that $R_k \leq 0$ for all $k > \max(k_0, k_1)$. This contradiction implies that our assumption Eq.(4.22) must be false. Hence, we conclude that

$$
\liminf_{k \to \infty} \|g(x^k)\| = 0 \quad \text{almost surely.}
$$

$\square$

5. Numerical Results

In this section, we preform our algorithm on some specific least square problem using simple fully connected neural network $\varphi(\theta)$ (θ is the trainable parameter) . Throughout all experiments, the algorithms were implemented in PyTorch and executed on an NVIDIA A100-SXM4-80GB GPU.

In the stochastic LM method, we define the parameter β in Eq. (4.16), which specifies the minimum number of parameters to be randomly selected at each iteration. In the first case, the stochastic step is effective whenever

$$\frac{1}{2(\mu^k)^2\|g(x^k)\|^4} \tag{5.1}$$

is sufficiently close to 1, corresponding to the value both μ^k and $\|g(x^k)\|$ are small. In particular, stochastic steps become more reliable near a minimizer, where the gradient norm is small and the trust-region radius is large. In the second case, when $\beta = 0$, although any subset of parameters can theoretically be selected, in practice, to ensure sufficient reduction in the objective value, we recommend sampling approximately two-thirds of the total parameters; that is, we set $\beta = \frac{2}{3}d$.

5.1. Possion equation

To evaluate and compare the performance of the proposed method, we consider the problem of solving a partial differential equations (PDEs). The Poisson equation is a fundamental model in mathematics and physics, widely used to describe steady-state phenomena such as heat distribution and electrostatics. Here, we focus on the two-dimensional Poisson equation with zero Dirichlet boundary conditions, given by:

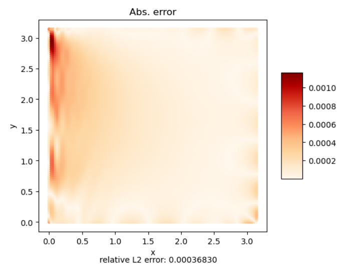
$$\Delta u = -68\sin(8x)\sin(2y), \quad (x, y) \in [0, \pi]^2, \tag{5.2}$$

subject to the boundary condition $u(x, y) = 0$ for all $(x, y) \in \partial[0, \pi]^2$. The exact solution of Eq. (5.2) is

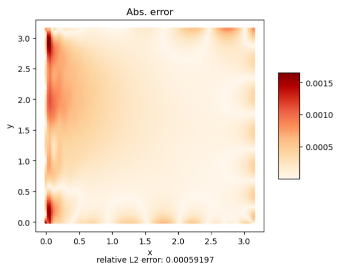
$$u(x, y) = \sin(8x)\sin(2y) \tag{5.3}$$

In this experiment, we sample 2,000 points uniformly across the domain with equal spacing. As for the model, we construct a neural network with two hidden layers, each containing 100 neurons, resulting in a total of 10,500 trainable parameters (M=2000, d=10,500). The tanh function has been used as the activation function. To obtain a more accurate result, we first apply the first-order optimizer Adam as a pretrain. We set the PDE loss:

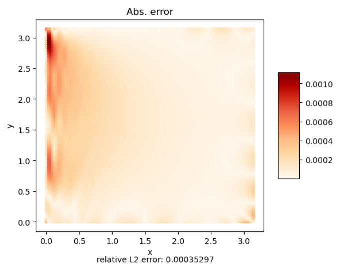
$$\mathcal{L} = \frac{1}{|\Omega_{\text{int}}|} \sum_{(x,y)\in\Omega_{\text{int}}} \left(\Delta\varphi + 68\sin(8x)\sin(2y)\right)^2 + \frac{1}{|\Omega_{\text{bdry}}|} \sum_{(x,y)\in\Omega_{\text{bdry}}} \varphi^2 \tag{5.4}$$



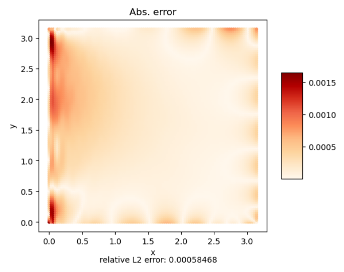
a) LM



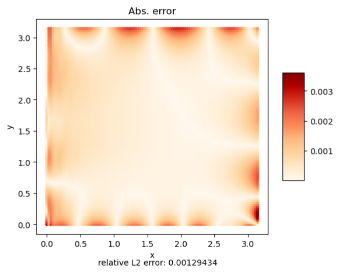
b) LMB



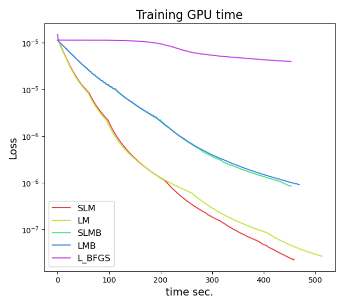
c) SLM



d) SLMB



e) L-BFGS



f) Loss vs GPU Time

Figure 2 Figures2.a)–e) present a comparison of the solutions to problem (5.2) obtained using LM, LMB, SLM, SLMB, and L-BFGS. Figure2.f) illustrates the loss versus GPU time (seconds) for these five algorithms, evaluated with 1,000 LM steps, 1,000 LMB steps, 1,050 SLM steps, 1,050 SLMB steps, and 5,000 L-BFGS steps.

where Ω_{int} and Ω_{bdry} denote the sets of points sampled from the interior and the boundary of the domain. The relative L_2 error is defined as

$$\text{Error} = \frac{\|\varphi - u\|}{\|u\|}, \tag{5.5}$$

The absolute error is given by $|\varphi - u|$.

It is evident from Fig. 2 that SLM outperforms the other four algorithms. Furthermore, since the updates involve a smaller Jacobian matrix, it is possible to reduce the computational cost of solving the large linear system in Eq. (3.3). As a result, both SLM and SLMB achieve shorter overall GPU training times, despite requiring more iterations.

Table 1 Average time per step (seconds)

Method	LM	LMB	SLM	SLMB	L-BFGS
Avg. time/ step	0.5124	0.4678	0.4359	0.4302	0.0904

5.2. Helmholtz equation

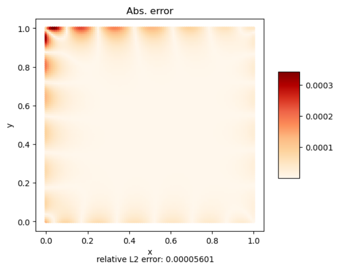
The Helmholtz equation describes time-harmonic solutions to the wave equation under the assumption of a sinusoidal time dependence. Unlike the Poisson equation, which models static potential fields, the Helmholtz equation characterizes steady-state oscillatory phenomena and inherently involves a characteristic frequency parameter. In this experiment, we consider a two–dimensional We consider the Helmholtz equation in the following form:

$$-\Delta u - k^2 u = f, \quad (x, y) \in [0,1]^2, \tag{5.6}$$

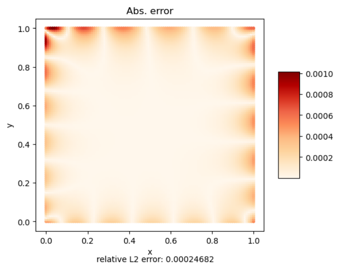
where the source term is given by $f = k^2 \sin(kx) \sin(ky)$.

Here, k denotes the wavenumber, which is related to the frequency of the oscillations in the solution. The pde subject to the boundary condition $u(x, y) = 0$ for all $(x, y) \in \partial[0,1]^2$and the exact solution of Eq. (5.6) is

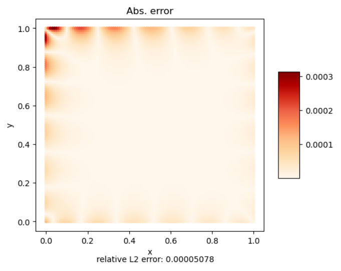
$$u(x, y) = \sin(kx) \sin(ky) \tag{5.7}$$



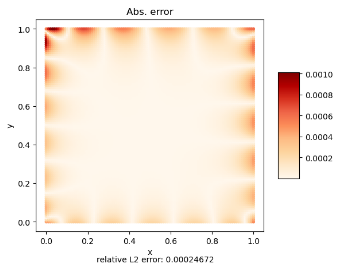
a) LM



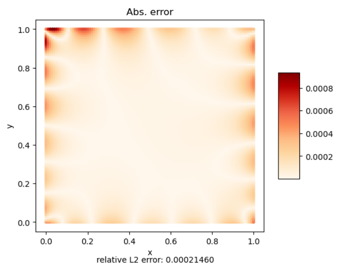
b) LMB



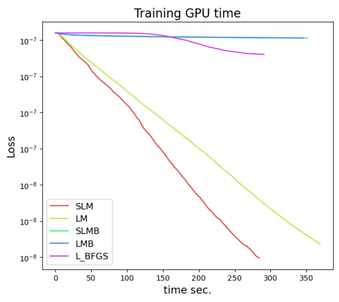
c) SLM



d) SLMB



e) L-BFGS



f) Loss vs GPU Time

Figure 3 Figures3.a)–e) present a comparison of the solutions to problem (5.6) obtained using LM, LMB, SLM, SLMB, and L-BFGS. Figure3.f) illustrates the loss versus GPU time (seconds) for these five algorithms, evaluated with 1,000 LM steps, 1,100 LMB steps, 1,150 SLM steps, 1,150 SLMB steps, and 3,500 L-BFGS steps.

In the experiments, we uniformly sample 2,000 points across the domain with equal spacing and use 7,560 trainable parameters (M=2000, d=7560). We set $k = 2\pi$, and the PDE loss is defined as:

$$\mathcal{L} = \frac{1}{|\Omega_{\text{int}}|} \sum_{(x,y)\in\Omega_{\text{int}}} \left(\Delta\varphi + k^2\varphi + f\right)^2 + \frac{1}{|\Omega_{\text{bdry}}|} \sum_{(x,y)\in\Omega_{\text{bdry}}} \varphi^2 \tag{5.8}$$

The L_2 error and absolute error are defined as same in problem 5.1.

From Fig. 3, it is evident that SLM continues to achieve superior performance compared to the other algorithms. However both SLMB and LMB fail to achieve sufficient loss reduction during the training process. Moreover, as shown in Table 2, it is interesting to observe that SLM reduces the average time per iteration by nearly one-third compared to LM.

Table 2 Average time per step (seconds)

Method	LM	LMB	SLM	SLMB	L-BFGS
Avg. time/ step	0.3686	0.3133	0.2470	0.3054	0.0830

5.3. regression tasks

Regression is a method for observing the relationship between the features and output variables and gives people ability to predict the output based on the true features information. In this experiment, we construct a neural network with two hidden layers, each containing 100 neurons. We utilize two widely used regression datasets CCS and WQ from [33, 34], which consist of 8 features with 1,030 instances ($M = 1,030$, $d = 11,100$) and 11 features with 4,898 instances ($M = 4,898$, $d = 11,400$), respectively. The loss function is defined as:

$$\mathcal{L} = \sum_{i=1}^{M} |\varphi(\theta, X_i) - Y_i|^2 , \tag{5.9}$$

where X_i and Y_i denote the feature vector and the corresponding target value of the i-th instance, respectively.

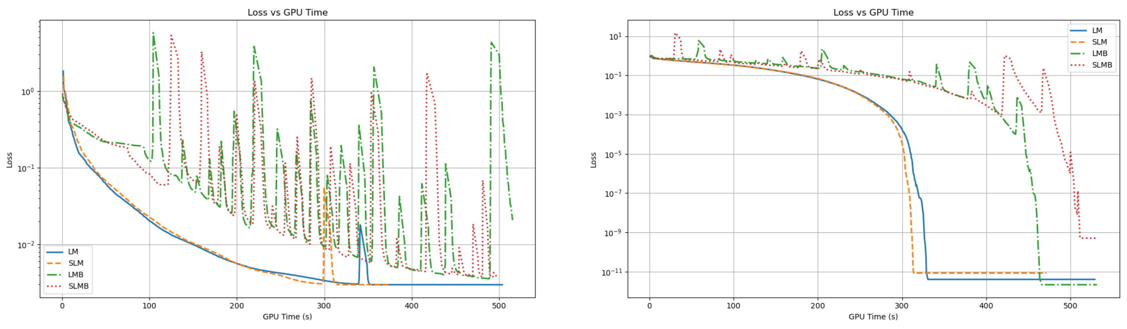


Figure 4 The left figure presents the training curves on the CCS dataset, corresponding to 1,000 LM steps, 1,050 SLM steps, 1,000 LMB steps, and 1,000 SLMB steps. The right figure shows the results on the WQ dataset, with 1,200 LM steps, 1,500 SLM steps, 1,200 LMB steps, and 1,300 SLMB steps.

On both datasets, SLM achieves accuracy comparable to that of the LM algorithm while significantly reducing the average time per step. Table 3 reports the exact average time per step for each algorithm.

Table 3 Average time per step (seconds) on two datasets

Task	LM	LMB	SLM	SLMB
CCS	0.5191	0.5088	0.3991	0.5100
WQ	0.4409	0.4430	0.3143	0.4091

6. Conclusion

The Stochastic Levenberg–Marquardt (SLM) method is designed for large-scale neural network problems. It aims to reduce the size of the Jacobian matrix, thereby lowering the computational cost of each LM update step. The method is particularly effective for problems that require high-accuracy solutions. The convergence analysis establishes a global convergence property under specific assumptions and also provides insight into the definition of the parameter β. In practice, it is common to set $\beta = \frac{2}{3}d$ when certain conditions are satisfied. Furthermore, a Jacobian matrix update strategy is introduced and extended to the SLM framework. The experimental evaluation includes applications of neural networks to solving the Poisson equation, the Helmholtz equation, and two regression tasks. In both PDE problems, SLM achieves the highest accuracy while requiring the least computational time. In the regression tasks, SLM attains accuracy comparable to that of the traditional LM method.

For future work, the update strategy for the regularization parameter μ^k could be further refined to account for the randomness introduced by parameter subset selection.

References

[1] Oludare Abiodun et al. “State-of-the-art in artificial neural network applications: A survey”. In: *Heliyon* 4 (Nov. 2018), e00938.

[2] Andrew W. Senior et al. “Improved protein structure prediction using potentials from deep learning”. In: *Nature* 577.7792 (2020), pp. 706–710.

[3] Giuseppe Carleo and Matthias Troyer. “Solving the quantum many-body problem with artificial neural networks”. In: *Science* 355.6325 (2017), pp. 602–606.

[4] Thomas Fischer and Christopher Krauss. “Deep learning with long short-term memory networks for financial market predictions”. In: *European Journal of Operational Research* 270.2 (2018), pp. 654–669. ISSN: 0377-2217.

[5] M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991.

[6] Guofei Pang, Lu Lu, and George Em Karniadakis. “fPINNs: Fractional Physics-Informed Neural Networks”. In: *SIAM Journal on Scientific Computing* 41.4 (2019), A2603–A2626.

[7] Mohsen Hayati and Karami Behnam. “Feedforward Neural Network for Solving Partial Differential Equations”. In: *Journal of Applied Sciences* 7 (Dec. 2007).

[8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536.

[9] Donald W. Marquardt. “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. In: *Journal of the Society for Industrial and Applied Mathematics* 11.2 (1963), pp. 431–441.

[10] Jorge J. Moré. “The Levenberg-Marquardt algorithm: Implementation and theory”. In: *Numerical Analysis*. Ed. by G. A. Watson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 105–116. ISBN: 978-3-540-35972-2.

[11] Henri P. Gavin. “The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems c ©”. In: 2013.

[12] Hao Yu and Bogdan Wilamowski. “Levenberg−Marquardt Training”. In: Feb. 2011, pp. 1–16. ISBN: 978-1-4398-0283-0.

[13] N. Yamashita and M. Fukushima. “On the Rate of Convergence of the Levenberg-Marquardt Method”. In: *Topics in Numerical Analysis*. Ed. by Goetz Alefeld and Xiaojun Chen. Vienna: Springer Vienna, 2001, pp. 239–249. ISBN: 978-3-7091-6217-0.

[14] José de Jesús Rubio. “Stability Analysis of the Modified Levenberg – Marquardt Algorithm for the Artificial Neural Network Training”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.8 (2021), pp. 3510–3524.

[15] Jinyan Fan and Jianyu Pan. “Convergence Properties of a Self-adaptive Levenberg-Marquardt Algorithm Under Local Error Bound Condition”. In: *Computational Optimization and Applications* 34.1 (2006), pp. 47–62.

[16] Murat A. Erdogdu and Andrea Montanari. “Convergence rates of sub-sampled Newton methods”. In: *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’15. Montreal, Canada: MIT Press, 2015, pp. 3052–3060.

[17] Albert S. Berahas, Raghu Bollapragada, and Jorge Nocedal and. “An investigation of Newton-Sketch and subsampled Newton methods”. In: *Optimization Methods and Software* 35.4 (2020), pp. 661–680.

[18] Dong C. Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical Programming* 45.1 (1989), pp. 503–528.

[19] Ganchen Xing, Jian Gu, and Xiantao Xiao. “Convergence analysis of a subsampled Levenberg-Marquardt algorithm”. In: *Operations Research Letters* 51.4 (2023), pp. 379–384. ISSN: 0167-6377.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[21] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[22] Asifullah Khan et al. “A Survey of the Recent Architectures of Deep Convolutional Neural Networks”. In: *Artificial Intelligence Review* 53.8 (2020), pp. 5455–5516.

[23] Alberto Garcia-Garcia et al. “A survey on deep learning techniques for image and video semantic segmentation”. In: *Applied Soft Computing* 70 (2018), pp. 41–65. ISSN: 1568-4946.

[24] Ruo-Yu Sun. “Optimization for Deep Learning: An Overview”. In: *Journal of the Operations Research Society of China* 8.2 (2020), pp. 249–294.

[25] Christian J. Darken, Joseph T. Chang, and John E. Moody. “Learning rate schedules for faster stochastic gradient search”. In: *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop* (1992), pp. 3–12.

[26] Shun ichi Amari. “Backpropagation and stochastic gradient descent method”. In: *Neurocomputing* 5.4 (1993), pp. 185–196. ISSN: 0925-2312.

[27] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *ArXiv* abs/1609.04747 (2016).

[28] S. Gratton, A. S. Lawless, and N. K. Nichols. “Approximate Gauss–Newton Methods for Nonlinear Least Squares Problems”. In: *SIAM Journal on Optimization* 18.1 (2007), pp. 106–132.

[29] “6. Global Convergence of the Basic Algorithm”. In: *Trust Region Methods*, pp. 115–168.

[30] Ruobing Chen, Matt Menickelly, and Katya Scheinberg. “Stochastic optimization using a trust-region method and random models”. In: *Mathematical Programming* 169 (2015), pp. 447 –487.

[31] Ya-xiang Yuan. “A Review of Trust Region Algorithms for Optimization”. In: *ICM99: Proceedings of the Fourth International Congress on Industrial and Applied Mathematics* (Sept. 1999).

[32] C. G. Broyden. “A Class of Methods for Solving Nonlinear Simultaneous Equations”. In: *Mathematics of Computation* 19 (1965), pp. 577–593.

[33] I-Cheng Yeh. *Concrete Compressive Strength*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5
1998.

[34] Cerdeira A. Almeida F. Matos T. Cortez Paulo and J. Reis. *Wine Quality*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56S3T. 2009.