mathematics

MDPI

Review

Perceptron: Learning, Generalization, Model Selection, Fault Tolerance, and Role in the Deep Learning Era

Ke-Lin Du [1,*], Chi-Sing Leung [2], Wai Ho Mow [3] and M. N. S. Swamy [1]

[1] Department of Electrical and Computer Engineering, Concordia University, Montreal, QC H3G 1M8, Canada
[2] Department of Electrical Engineering, City University of Hong Kong, Hong Kong, China
[3] Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong, China
* Correspondence: kldu@ece.concordia.ca

Abstract: The single-layer perceptron, introduced by Rosenblatt in 1958, is one of the earliest and simplest neural network models. However, it is incapable of classifying linearly inseparable patterns. A new era of neural network research started in 1986, when the backpropagation (BP) algorithm was rediscovered for training the multilayer perceptron (MLP) model. An MLP with a large number of hidden nodes can function as a universal approximator. To date, the MLP model is the most fundamental and important neural network model. It is also the most investigated neural network model. Even in this AI or deep learning era, the MLP is still among the few most investigated and used neural network models. Numerous new results have been obtained in the past three decades. This survey paper gives a comprehensive and state-of-the-art introduction to the perceptron model, with emphasis on learning, generalization, model selection and fault tolerance. The role of the perceptron model in the deep learning era is also described. This paper provides a concluding survey of perceptron learning, and it covers all the major achievements in the past seven decades. It also serves a tutorial for perceptron learning.

Keywords: multilayer perceptron; perceptron; backpropagation; stochastic gradient descent; second-order learning; model selection; robust learning; deep learning

MSC: 68T07

Citation: Du, K.-L.; Leung, C.-S.; Mow, W.H.; Swamy, M.N.S. Perceptron: Learning, Generalization, Model Selection, Fault Tolerance, and Role in the Deep Learning Era. *Mathematics* **2022**, *10*, 4730. https://doi.org/10.3390/math10244730

Academic Editor: Mario Muñoz Organero

Received: 10 October 2022
Accepted: 9 December 2022
Published: 13 December 2022

1. Introduction

Perceptron is the most important neural network model and has been extensively studied in the past six decades. Its history dates to the early 1940s, when McCulloch and Pitts discovered that a neuron could be modeled as a threshold device that could perform logic functions [1]. Later in the 1950s, Rosenblatt [2,3] proposed the perceptron model and its learning algorithm.

In the 1960s, Widrow and Hoff [4] proposed a similar model called Adaline. This model could be trained with the least mean squares (LMS) approach. Interest in neural networks diminished in the 1970s when Minsky and Papert [5] proved that the simple perceptron model [2] was unable to perform complex logic function and could not solve linearly inseparable problems. The revival of interest in the perceptron model took place in 1986, when Rumelhart, Hinton, and Williams [6] trained the multilayer perceptron (MLP) model using the backpropagation (BP) algorithm. Later BP was found to have been described in 1974 by Werbos [7].

For perceptron models, supervised learning is employed. Supervised learning compares the network output and the desired output. It is a closed-loop system, with the error as the feedback signal. In supervised learning, an objective function is first defined. To make the objective cost go toward zero, a gradient-descent procedure, such as LMS [4] and BP algorithms [6], is usually applied.

The MLP model is desirable in regression and classification. When trained with the same input and output patterns, an MLP functions as an autoassociative memory, and the bottleneck layer can serve as a feature extractor [8,9]. LeCun's six-layer MLP model for handwritten digit recognition is the pioneering work in deep neural network and deep learning.

Since 1986, numerous new results on perceptron learning have been proposed. The research accomplishments on MLPs laid the foundation of the neural networks discipline. The MLP has been the most widely used neural network model in the literature. There is a feature extraction process in the MLP. For example, in [10], the MLP model was used for DDoS attack detection and in [11] it was used for the fault prediction in the energy industry. However, to the best of our knowledge, we have not found a systematic survey of the perceptron model in the literature. We understand that the perceptron model is the classical neural network model, and people may think that the topic is not sufficiently new. However, this model by itself is the most widely investigated, and there are a massive number of publications on the model. Even today, people are still actively conducting research on some aspects of this model such as the basic theory of its learning algorithms and their applications in deep learning. While most people do not have an overall knowledge of the model, it is our motivation to conduct a comprehensive and also state-of-the-art survey of the perceptron model and its learning. After reading this paper, readers will be able to have a big picture of the foundation of the most fundamental model in the neural networks discipline, learn how to use this model, and where to start if they wish to conduct research on the perceptron model.

In this AI or deep learning era, numerous new applications of neural networks have been reported, such as sensor networks [12], production optimization [13], the control of a delay teleoperation system [14], and user behavior analysis [15].

This paper is motivated to give a comprehensive, yet state-of-the-art review and also a tutorial on the perceptron model and its learning. This paper is organized as follows. In Section 2, some basic background on machine learning associated with perceptron models are presented. In Section 3, the single-layer perceptron (SLP) and its learning are dealt with. Section 4 introduces the architecture, properties of the MLP, and the BP algorithm. Section 5 presents some representative results on improving the generalization ability of MLP models. Section 6 deals with the optimization of the MLP architecture. In Section 7, various strategies for speeding up the first-order BP algorithm are introduced. Section 8 discusses several second-order learning methods for MLP. Some other learning algorithms, including expectation–maximization (EM) [16] and natural gradient [17], are briefly described in Section 9. Section 10 describes fault-tolerant learning for MLP. Section 11 features the relation between MLP and deep learning. Section 12 presents an illustrative example, then gives some advice on how to select an MLP learning algorithm for a specific application setting, and finally concludes the entire paper.

2. Background

2.1. Neurons

A McCulloch–Pitts neuron [1] collects all signals from other nodes and generates an output, as shown in Figure 1. Its input–output mapping is given by

$$o = \phi(net)\,, \tag{1}$$

where

$$net = \vec{w}^T \vec{x} + \theta \tag{2}$$

where $\vec{x} = (x_1, \ldots, x_{J_1})^T$ is the J_1-dimensional input, $\vec{w} = (w_1, \ldots, w_{J_1})^T$ is the weight vector, and θ is a threshold or bias. $\phi(\cdot)$ is a continuous or discontinuous activation function that maps a real number into $(-1, 1)$ or $(0, 1)$. Popular activation functions are given by

$$\phi(a) = \begin{cases} 1, & a \geq 0 \\ -1 \text{ or } 0, & a < 0 \end{cases}, \quad \text{Hard limiter (threshold)}, \tag{3}$$

$$\phi(a) = \frac{1}{1 + e^{-\beta a}}, \quad \text{Logistic}, \tag{4}$$

$$\phi(a) = \tanh(\beta a), \quad \text{Hyperbolic tangent}, \tag{5}$$

$$\phi(a) = a, \quad \text{Linear}, \tag{6}$$

where β is a gain parameter for the steepness control.

The logistic function and the hyperbolic tangent function are sigmoidal functions, which are monotonically increasing and satisfy $\lim_{a \to +\infty} \phi(a) = 1$, $\lim_{x \to -\infty} \phi(a) = 0$, or -1. The McCulloch–Pitts neuron is used in the MLP, the Hopfield network, and many other models that may utilize other activation functions.

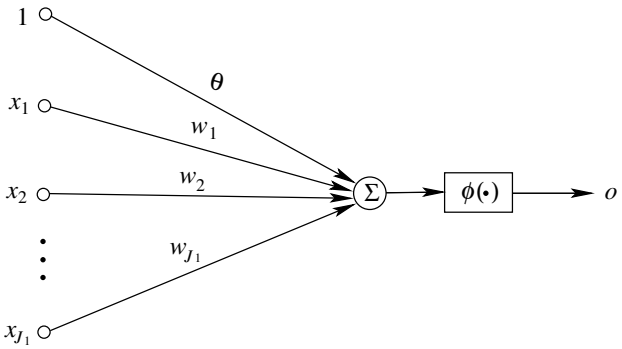

Figure 1. The McCulloch–Pitts neuron model.

2.2. Classification: Linear Separability and Nonlinear Separability

In classification, we have a training data set $\mathcal{D}_{train} = \{(\vec{x}_1, y_1), \cdots, (\vec{x}_N, y_N)\}$, where $\vec{x}_p$'s are the training input, and y_p's are the training outputs that can be ± 1 (or 1 or 0), denoting whether example p belongs to a specific class.

A two-class classification problem is said to be linearly separable, if there exists a linear separating surface that separates all the input samples of class $\mathcal{C}_1$ from class $\mathcal{C}_2$. A McCulloch–Pitts neuron with a hard limiter activation, or say a linear threshold gate (LTG), can realize dichotomy, characterized by a hyperplane

$$\vec{w}^T \vec{x} + \theta = 0\,. \tag{7}$$

An input pattern belongs to $\mathcal{C}_1$ if $\vec{w}^T \vec{x} + \theta \geq 0$, and belongs to $\mathcal{C}_2$ if $\vec{w}^T \vec{x} + \theta < 0$.

A linearly inseparable dichotomy may be nonlinearly separable. A dichotomy of set $\mathcal{X}$, denoted $\{\mathcal{C}_1, \mathcal{C}_2\}$, is *φ-separable* if there exists a mapping $\varphi : R^{J_1} \to R^{J_2}$ [18]

$$\vec{w}^T \varphi(\vec{x}) = 0 \tag{8}$$

such that $\vec{w}^T \varphi(\vec{x}) \geq 0$ when $\vec{x} \in \mathcal{C}_1$ and $\vec{w}^T \varphi(\vec{x}) < 0$ when $\vec{x} \in \mathcal{C}_2$, where $\vec{w} \in R^{J_2}$. The mapping φ is referred to as the concept of functional link network [19], which can map a linearly inseparable dichotomy so that it becomes separable. In particular, when replacing the linear term in an LTG by a high-order polynomial, a polynomial threshold neuron can be used to solve the nonlinearly separable problem. A detailed analysis on the second-order threshold neuron was provided in [20]. However, in the functional link approach, the way to choose the mapping was not addressed [19]. The support vector machine (SVM) algorithm [21] was developed around the same time and could automatically construct the mapping φ and the weight vector $\{\vec{w}, \theta\}$.

2.3. Boolean Function Approximation

In Boolean function approximation, we would like to use a neural network to model a logic or Boolean function. The input and output values of the neural network are Boolean variables. There exist 2^{J_1} combinations for J_1 independent Boolean variables, yielding $2^{2^{J_1}}$ different Boolean functions. Some Boolean functions can be realized using LTGs.

For a set of m points in $\mathbb{R}^n$, if every subset of m or fewer points is linearly independent, it is known to be in *general position*. The function-counting theorem [18] estimates the separating capability of an LTG. It counts the number of linearly separable dichotomies of m points that are in general position in $\mathbb{R}^n$.

The probability of linear dichotomy is denoted $P(m,n)$, for an n-input LTG to separate m points in general position. According to the function-counting theorem [22], $P(m,n) = 1$ when $\frac{m}{n+1} \leq 1$, $P(m,n) \to 1$ when $1 < \frac{m}{n+1} < 2$ and $n \to \infty$, and $P(m,n) = \frac{1}{2}$ when $\frac{m}{n+1} = 2$. When characterizing the statistical capability of a single LTG, $\frac{m}{n+1} = 2$ is typically adopted.

2.4. Function Approximation

In function approximation, there is a training dataset, $\mathcal{D}_{train} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$, where $\vec{x}_p$ and y_p are the input and output of the pth sample, respectively. The input and output can be modeled by

$$y_p = f(\vec{x}_p) + \varepsilon_p, \tag{9}$$

where $f(\cdot)$ is a mapping, and ε_p is a Gaussian noise with zero mean and variance σ_ε^2. Our goal is to construct a model $\hat{f}(\cdot)$ to minimize the mean squared error (MSE):

$$\mathcal{J}_{\text{MSE}} = \frac{1}{N} \sum_{p=1}^{N} (y_p - \hat{f}(\vec{x}_p))^2 . \tag{10}$$

3. Perceptron

3.1. Simple Perceptron

A simple perceptron is a one-neuron perceptron which is actually the McCulloch–Pitts neuron model [1], as shown in Figure 1. It uses the hard limiter as the activation function. Let $\mathcal{D}_{train} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ be the training set, where the $\vec{x}_p$'s are the training inputs, and the y_p's are the training outputs that can be ± 1 (or 1 or 0). In this notation, $y_p = 1$ means that $\vec{x}_p \in \mathcal{C}_1$, and $y_p = -1$ means that $\vec{x}_p \in \mathcal{C}_2$. Our goal is to estimate a weight vector $\vec{w}$ and a threshold θ such that a hyperplane

$$\vec{w}^T \vec{x} + \theta = 0 \tag{11}$$

separates the two classes:

$$\begin{cases} \vec{w}^T \vec{x} + \theta & \geq 0, \quad \text{when } \vec{x} \in \mathcal{C}_1 \\ \vec{w}^T \vec{x} + \theta & < 0, \quad \text{when } \vec{x} \in \mathcal{C}_2 \end{cases} . \tag{12}$$

When more neurons are used, a single-layer perceptron (SLP) is obtained, see Figure 2. An SLP can classify a vector $\vec{x}$ into more classes. The system state of a J_1–J_2 SLP is updated by

$$\vec{net} = \mathbf{W}^T \vec{x} + \vec{\theta}, \tag{13}$$

$$\vec{o} = \vec{\phi}(\vec{net}), \tag{14}$$

where $\vec{net} = \left(net_1, \ldots, net_{J_2}\right)^T$ is the net input vector, $\vec{o} = \left(o_1, \ldots, o_{J_2}\right)^T$ is the output vector, $\vec{\phi}(\vec{net}) = \left(\phi_1(net_1), \ldots, \phi_{J_2}(net_{J_2})\right)^T$, and ϕ_i is the activation function of the ith output neuron.

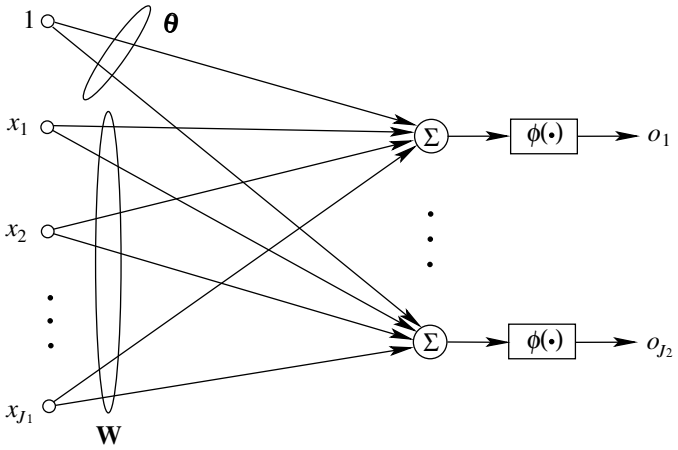

Figure 2. Architecture of the single-layer perceptron.

3.2. Perceptron Learning Algorithm

In the perceptron learning algorithm [3,5], the training patterns are sequentially and repeatedly presented. Given a training pattern $\vec{x}_t$ at time t, we have the updating rule

$$net_{t,j} = \sum_{i=1}^{J_1} x_{t,i} w_{ij}(t) + \theta_j = \vec{w}_j^T \vec{x}_t + \theta_j, \tag{15}$$

$$o_{t,j} = \begin{cases} 1, & \text{when } net_{t,j} > 0 \\ 0, & \text{otherwise} \end{cases}, \tag{16}$$

$$e_{t,j} = y_{t,j} - o_{t,j}, \tag{17}$$

$$\begin{cases} w_{ij}(t+1) &= w_{ij}(t) + \eta x_{t,i} e_{t,j} \\ \theta_j(t+1) &= \theta_j(t) + \eta e_{t,j} \end{cases}, \tag{18}$$

where $net_{t,j}$ stands for the net input of neuron j for example t, w_{ij} the weight from neurons i to j, $\vec{w}_j = \left(w_{1j}, w_{2j}, \ldots, w_{J_1 j}\right)^T$, θ_j the threshold of neuron j, $x_{t,i}$ input i of example t, $o_{t,j}$ and $y_{t,j}$, respectively, the output and the desired output of neuron j for example t, $e_{t,j}$ the error at neuron j for example t, and η the learning rate. w_{ij}s are initialized at random, and η is usually set to 0.5. Learning terminates when the errors $e_{t,j}$'s are equal to zeros in a learning cycle. The perceptron convergence theorem [3,5] asserts that when the training dataset for input patterns are linearly separable, the perceptron learning algorithm can always find decision hyperplanes to correctly separate classes in finite time.

The perceptron learning algorithm is valid only for the classification of linearly separable patterns. When trained with linearly inseparable patterns, it cannot stop the iteration [5]. Failure to converge for linearly inseparable problems is due to the fact that the perceptron learning algorithm is not able to detect the minimum of the error function [23]. Some variants of the perceptron learning, such as the pocket algorithm [24] and the thermal perceptron learning [25], are used for handling linearly inseparable patterns. The pocket algorithm adds a checking amendment to terminate the perceptron learning algorithm, achieving the minimization of the erroneous classification rate [24]. The weight vector having the longest unchanged run is treated as the best solution so far and is put into the *pocket*. The pocket convergence theorem asserts the convergence of the pocket algorithm, when the inputs are integers or rational [24,26]. The thermal perceptron learning [25] introduces a temperature annealing factor into the second term of (18).

3.3. Least Mean Squares Algorithm

The LMS algorithm [4] can reliably separate the patterns of different classes by minimizing the MSE by using the gradient-descent method. The linear activation function is used at the training stage, and thus, unlike (17), the error is given by

$$e_{t,j} = y_{t,j} - net_{t,j}, \tag{19}$$

where $net_{t,j}$ is the same as (15). The weight update is given by (18). At the operation stage, thresholding is performed on the linear output to generate a binary output. The node containing a linear combiner and thresholding is referred to as Adaline.

The LMS algorithm can also be used for estimating the parameters in linear networks [27–29] for linear regression. In linear regression, we have a training set $\mathcal{D}_{train} = \{(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_N, \vec{y}_N))$, where $\vec{x}_j \in \mathbb{R}^{J_1}$ is the input and $\vec{y}_j \in \mathbb{R}^{J_2}$ is the output. Our goal is to find the parameters such that the MSE

$$\mathcal{J}(\mathbf{W}, \vec{\theta}) = \frac{1}{N} \sum_{p=1}^{N} \|\vec{y}_p - \mathbf{W}^T \vec{x}_p - \vec{\theta}\|^2 \tag{20}$$

is minimized.

The LMS rule comprising (15), (18) and (19) is known as the μ-LMS rule. The learning parameter usually takes $0 < \eta < \frac{2}{\max_t \rho_{x_t}}$ to ensure its convergence, where $\rho_{x_t} = \left\| [\vec{x}_t^T \; 1] \right\|^2$.

The Widrow–Hoff delta rule, known as the α-LMS, is obtained by normalizing the input vector [30]:

$$\begin{cases} w_{ij}(t+1) &= w_{ij}(t) + \eta \frac{x_{t,i} e_{t,j}}{\rho_{x_t}} \\ \theta_j(t+1) &= \theta_j(t) + \eta \frac{e_{t,j}}{\rho_{x_t}} \end{cases}, \tag{21}$$

where $0 < \eta < 2$ ensures convergence, and η is practically selected as $0.1 < \eta < 1.0$ [30].

In the above, the convergence results [4,30,31] are based on the stochastic setting, where the training samples are independently drawn over time. Under the condition that the training samples are cyclically and sequentially presented, some convergence results were developed in [32,33]. In [32], the result showed that the LMS rules converged to a limit cycle. The limit cycle reduced to an LMS solution when $\eta \to 0$. That meant there existed a tradeoff on η between the final error and convergence speed. In [33], the LMS algorithm, with decreasing η, generated an ϵ-optimal weight matrix, which was at most ϵ away from the LMS solution, after $O(\frac{1}{\epsilon})$ training cycles. For a fixed η, it needed $\Omega(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ training cycles to obtain an ϵ-optimal weight matrix. The Widrow–Hoff delta rule laid the foundation of modern adaptive signal processing. There is also a complex LMS [34].

3.4. Other Learning Algorithms

Examples of other learning rules for the SLP are Mays' rule [22,30,35], Ho–Kashyap rule [36,37], adaptive Ho–Kashyap rules [22,38], voted perceptron algorithm [39,40], perceptron algorithm for the perceptron using margins [39,41] or margitron [42], one-shot Hebbian learning [43], and nonlinear Hebbian learning [44].

Other examples for SLP learning algorithms are linear programming (LP) [37,45], convex analysis and nonsmooth optimization [23], constrained steepest-descent algorithm based on quadratic programming (QP) [46], fuzzy perceptron [47,48], fuzzy pocket [48], the conjugate-gradient (CG) method [49,50], control-inspired parameter selection for iterative steepest descent and CG algorithms [51], and shifting perceptron algorithm [52].

3.5. Approximation Ability of Perceptron Variants

The nonlinearity of the activation function introduces local minima in the MSE functions. The number of local minima grows exponentially as the input dimension increases [53].

The classification capability of a sign-constrained perceptron was analyzed in [54]. The authors derived a necessary and sufficient criterion for a sign-constrained perceptron to learn all 2^m dichotomies over a given set of m patterns. For a perceptron with n inputs, the VC-dimension is $n+1$, whereas for a sign-constrained perceptron, it is n [54].

When training a perceptron with a set of nonlinearly separable input patterns, the obtained weights may have a limit cycle behavior [55]. The least number of updates for the weights to reach the limit cycle is dependent on the initial weights. The authors of [55] also derived a necessary and sufficient condition for the weights exhibiting a limit cycle behavior, and estimated the number of updates for the weights to reach the limit cycle. An invariant set of the weights that were trained by the perceptron learning algorithm was characterized in [56].

When its majority vote is treated as a binary output, the SLP can compute any Boolean function [57]. When applying a squashing function to the percentage of votes with value 1, the SLP is a universal approximator for any continuous functions in $[0,1]$ [57].

The parallel perceptron implements a soft-winner-take-all gate on the binary outputs of gates on the hidden layer. It is a committee machine. The parallel perceptron is a universal approximator [57]. The parallel delta (p-delta) rule [57] implements a maximization of the margin of individual perceptrons. When trained with the p-delta rule, parallel perceptrons have performance comparable to that of an MLP and an SVM [57]. Direct parallel perceptrons [58] calculate the weights of parallel perceptrons by using an analytical closed-form expression, and they have a linear complexity in terms of both the number of patterns and the input dimension.

4. Multilayer Perceptrons

4.1. Structure

To overcome the linearly inseparable limitation, one can extend the SLP to a multilayered structure, such as an MLP. Madaline models are available by stacking multiple layers of Adalines [30]. Madaline is unable to solve the linearly inseparable problem, as Adaline is a linear network whose consecutive layers can be aggregated into a single layer by multiplying their respective weight matrices.

Madaline I [59,60] and Madaline II [61] are multilayered feedforward networks based on hard limiter neurons or other fixed-logic devices, such as AND, OR, majority-voting elements. Since the hard limiter function is not differentiable, the Madaline I and Madaline II learning rules are based on the LMS rule. The modification of weights is based on the minimal disturbance principle. This disturbance-based updating algorithm is quite slow. They can solve the linearly inseparable problem.

The neuron with the smallest net output in the first hidden layer is selected first and its output is inverted. If the inversion can improve the performance, the input weights of the selected neuron are updated according to the LMS algorithm, then applied to other neurons in the first hidden layer. After all first hidden layer neurons have been considered, the updating is moved to upper layer neurons. The learning speed of Madaline II's rule is slow because we need to feed the training dataset to the network for each inversion of neurons.

A feedforward neural network (FNN) [6] usually has one or more hidden layers between the input and output layers. Such layered FNN is called an MLP. The architecture of the MLP is shown in Figure 3. There are M layers, and layer m has J_m nodes. All the input nodes are collectively treated as the first layer. We denote the weights from layer $(m-1)$ to layer m by $\mathbf{W}^{(m-1)}$, and the bias, output and activation function of neuron i in layer m by $\theta_i^{(m)}$, $o_i^{(m)}$ and $\phi_i^{(m)}(\cdot)$, respectively. Subscript p and superscript (m) correspond to sample p and layer m, respectively. In other words, the input of sample p is

$$\vec{x}_p \equiv \vec{o}_p^{(1)}\,, \tag{22}$$

the network output of sample p is

$$\vec{o}_p \equiv \vec{o}_p^{(M)}, \tag{23}$$

and the desired output of sample p is denoted as $\vec{y}_p$. For $m = 2, \ldots, M$,

$$\vec{net}_p^{(m)} = \left[\mathbf{W}^{(m-1)}\right]^T \vec{o}_p^{(m-1)} + \vec{\theta}^{(m)}, \tag{24}$$

$$\vec{o}_t^{(m)} = \vec{\phi}^{(m)}\left(\vec{net}_p^{(m)}\right), \tag{25}$$

where $\vec{net}_p^{(m)} = \left(net_{p,1}^{(m)}, \ldots, net_{p,J_m}^{(m)}\right)^T$, $\mathbf{W}^{(m-1)} \in \mathbb{R}^{J_{m-1} \times J_m}$, $\vec{o}_p^{(m-1)} = \left(o_{p,1}^{(m-1)}, \ldots, o_{p,J_{m-1}}^{(m-1)}\right)^T$, $\vec{\theta}^{(m)} = \left(\theta_1^{(m)}, \ldots, \theta_{J_m}^{(m)}\right)^T$, and $\vec{\phi}^{(m)}(\cdot)$ applies $\phi_i^{(m)}(\cdot)$ to the ith entry of the vector within.

For classification, all hidden and output neurons are with a sigmoidal function or hard limiter. For function approximation, all hidden neurons are with a sigmoidal function, but the output neurons in layer M use a linear activation function.

When all hidden neurons and output neurons are with a hard limiter activation, an MLP can be considered as a Madaline II [61].

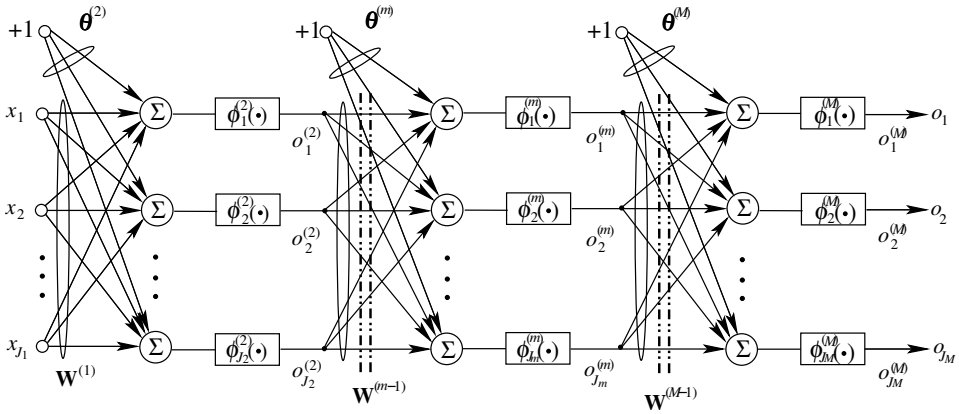

Figure 3. Architecture of the MLP. The MLP has M layers. The mth layer has J_m nodes.

4.2. Approximation Ability

The MLP model has a universal approximation ability when the number of hidden nodes is sufficiently large. The universal approximation capability of a four-layer MLP can be easily obtained from Kolmogorov's theorem [62] and has also been proved in [63–65]. More importantly, the three-layer MLP model with continuous sigmoidal activation functions has been proved to be capable of approximating any continuous multivariate function to an arbitrary accuracy [66–70].

For approximating a target function by the MLP, the necessary number of neurons relies on the geometry of the target function, as well as the minimal number of line segments/hyperplanes [69] or the number of extrema [71] that can construct the basic geometrical shape of the target function.

4.3. Backpropagation Learning Algorithm

The BP algorithm has long been the most popular and fundamental learning algorithm for MLPs [6,7,72,73]. The BP algorithm is a gradient-search technique that minimizes a cost function. The cost function is defined by the MSE:

$$\mathcal{E} = \frac{1}{N}\sum_{p=1}^{N}\mathcal{E}_p = \frac{1}{2N}\sum_{p=1}^{N}\left\|\vec{y}_p - \vec{o}_p\right\|^2, \tag{26}$$

where N is the size of the training set, $\vec{o}_p$ and $\vec{y}_p$ are, respectively, the actual network output and the desired output for training pattern p,

$$\mathcal{E}_p = \frac{1}{2}\|\vec{y}_p - \vec{o}_p\|^2 = \frac{1}{2}\vec{e}_p^T\vec{e}_p \tag{27}$$

is the error of training pattern p, and

$$\vec{e}_p = \vec{y}_p - \vec{o}_p\,, \tag{28}$$

the element i of $\vec{e}_p$ being $e_{p,i} = y_{p,i} - o_{p,i}$.

When minimizing $\mathcal{E}_p$, we have the change of the weight or bias:

$$\Delta_p w_{ij}^{(m-1)} = -\eta \frac{\partial \mathcal{E}_p}{\partial w_{ij}^{(m-1)}} \text{ and } \Delta_p \theta_i^{(m)} = -\eta \frac{\partial \mathcal{E}_p}{\partial \theta_i^{(m)}}, \tag{29}$$

where the learning rate η is a small positive number. For the output layer, the changes of the parameters are given by

$$\begin{cases} \Delta_p w_{ij}^{(M-1)} &= \eta\, e_{p,i} \frac{\partial o_{p,i}}{\partial w_{ij}^{(M-1)}} \\ &= \eta\, e_{p,i} \left(\frac{d\phi_i^{(M)}(a)}{da} \right)_{a=net_{p,i}^{(M)}} o_{p,j}^{(M-1)} \\ \Delta_p \theta_i^{(M)} &= \eta\, e_{p,i} \frac{\partial o_{p,i}}{\partial \theta_i^{(M-1)}} \\ &= \eta\, e_{p,i} \left(\frac{d\phi_i^{(M)}(a)}{da} \right)_{a=net_{p,i}^{(M)}} 1 \end{cases}. \tag{30}$$

If we can consider the term $e_{p,i} \left(\frac{d\phi_i^{(M)}(a)}{da} \right)_{a=net_{p,i}^{(M)}} \equiv \delta_{p,i}^{(M)}$ as a generalized error term, the updating rule in BP can be considered as a generalized delta rule [4]. To calculate the changes of parameters in the hidden and input layers, we can apply the chain rule and then get

$$\Delta_p w_{uv}^{(m-1)} = \eta\, \delta_{p,u}^{(m)}\, o_{p,v}^{(m-1)} \text{ and } \Delta_p \theta_u^{(m)} = \eta\, \delta_{p,u}^{(m)}\, 1\,. \tag{31}$$

The generalized error terms $\delta_{p,u}^{(m)}$'s can be solved by applying the chain rule. The changes of the parameters in the output layer are first calculated. Then, the generalized error terms are backpropagated by means of the chain rule and are used for calculating the generalized error terms in the hidden layers. The BP algorithm requires a continuous differentiable activation function.

4.4. Batch Mode and Online Mode

There are two versions of the BP algorithm, batch and online modes. In the batch mode, the parameters are updated only after a complete presentation of training patterns. In the online mode, the parameters are updated at once after a training pattern is presented. In the online mode, the usual practice is that all the examples are presented cyclically by epoch and in a random order during each epoch, until the convergence criteria are reached. In either batch or online mode BP, the concept follows that of the gradient-descent technique. In the literature, batch mode BP is commonly known as gradient descent, while online mode BP is known as stochastic gradient descent. In BP, too small an η increases the possibility of getting stuck at a local minimum, while too large an η may result in oscillatory traps.

Since the batch mode BP follows the standard gradient-descent technique, the convergence (to a local minimum) is guaranteed when a sufficiently small η is used. There are several works on the convergence for the online mode BP [74–79]. Based on the stochastic

gradient theorem [74–76], if the training examples are randomly drawn from an infinitely large dataset, the convergence of the online mode BP is guaranteed when a decreasing learning rate η_t is used, $\sum_{t=1}^{\infty}\eta_t = \infty$, and $\sum_{t=1}^{\infty}\eta_t^2 < \infty$, where t is the time index for training. For a dataset with finite size, some convergence results are given in [77–80], where the assumptions are that the learning rate is fixed to a constant within a learning cycle, and that it deceases to zero as learning cycles increase.

Online BP can be implemented when a complete training set is unavailable. This is usually implemented for very large training sets. Online BP has a quasi-simulated annealing property arising from the randomness introduced by the small constant learning rate, and this property helps the algorithm escape local minima [81]. However, online BP is not suitable for parallelization. It is usually orders of magnitude faster than batch BP, but with at least the same accuracy, especially for large training sets [82]. Batch BP is usually too slow for large training sets due to the required small learning rate. Online BP can train more quickly and safely by using a larger learning rate. In [82], a recommendation on the learning rates between online BP and batch BP is given by $\eta_{\text{online}} = \sqrt{N}\eta_{\text{batch}}$. When η is small enough, the number of epochs required for learning has a linear relationship with η.

With respect to the absolute value of the expected difference and the expected squared difference, and with any analytic sigmoidal function, the convergence for online BP and batch BP are derived and compared for different η in [83]. For batch BP, $\eta \geq 2$ is required for convergence, while for online BP, η increases up to N. In [84], for online BP training of three-layer feedforward networks subject to the minimum error, a dynamic learning rate ensures the error sequence converges to the global minimum error. In [85], the maximal learning rates are derived as a function of batch size for both gradient descent and adaptive algorithms, such as Adam, by using random matrix theory.

4.5. Momentum Term

By adding a momentum term to the BP algorithm, the BP with momentum (BPM) algorithm is obtained as [6]

$$\begin{cases} \Delta_p w_{ij}^{(m-1)}(t+1) &= -\eta \frac{\partial \mathcal{E}_p}{\partial w_{ij}^{(m-1)}} + \alpha \Delta_p w_{ij}^{(m-1)}(t) \\ \Delta_p \theta_i^{(m)}(t+1) &= -\eta \frac{\partial \mathcal{E}_p}{\partial \theta_i^{(m)}} + \alpha \Delta_p \theta_{ij}^{(m-1)}(t) \end{cases}, \tag{32}$$

where α is the momentum factor; usually $0 < \alpha \leq 1$ and typically $\alpha = 0.9$. The momentum term can effectively improve the convergence when the estimated weights are in almost-flat steady downhill regions of the error surface and has a stabilizing effect when the estimated weights are in regions with high fluctuations. The momentum term introduces second-order information, and thus BPM resembles the CG method [49,50]. BPM is analyzed in [86] and the conditions for convergence are derived.

The convergence rate and MSE performance of BPM was investigated in [87]. BPM can be viewed as standard stochastic gradient method having a rescaled but larger step size. A decaying momentum factor can retain adaptation but avoid performance degradation. A stability analysis of two BPM algorithms for quadratic functions derives the optimal learning rates and the optimal momentum factors simultaneously, leading to the fastest convergence [88].

4.6. Variance Reduction

For stochastic gradient descent, the variance of the randomly drawn gradients will never go to zero. A sublinear convergence rate is obtained only under decreasing step-size sequences. The variance-reduced technique reduces the variance to zero.

The stochastic average gradient (SAG) method [89] uses the sum of the latest individual gradients as an estimator of the descent direction. SAG has a linear convergence rate using a step size $\eta = O(1/L_{max})$, L_{max} being a Lipschitz constant. SAG requires $O(nd)$ storage and uses a biased gradient estimator, for n training examples in R^d. SAG only computes a single

stochastic gradient at each iteration. Stochastic variance reduced gradient (SVRG) [90] uses an unbiased gradient estimator. SAGA [91] improves SAG by using an unbiased update of SVRG. SAGA has the same linear convergence and storage as SAG, but with a much simpler proof. SVRG [90] has performance similar to SAG and SAGA, but only requires $O(d)$ memory. SVRG needs to tune the number of inner iterations, two gradients are computed per iteration, and the full gradient needs to be computed every time the reference point is changed. However, their step size depends on L_{max}, which may not be difficult to decide for some problems. The convergence rate of the variance-reduced methods depends on the number of training examples n, while that of classic SGD is not dependent on n. This means that variance-reduced methods can perform worse than SGD in the early iterations when n is very large.

The stochastic dual coordinate ascent (SDCA) method [92] extends coordinate descent methods to the finite-sum problem. Stochastic coordinate descent implements a coordinate-wise gradient descent with respect to a randomly chosen variable and updates this variable while keeping the other variables unchanged.

Recent developments to accelerate the convergence of stochastic optimization by exploiting second-order information are discussed in [93]. This is achieved by stochastic variants of quasi-Newton methods that approximate the curvature of the objective function using stochastic gradient information.

5. Generalization

5.1. Generalization Error

A neural network trained by minimizing the MSE on a training set is not guaranteed to perform well on an unseen test set. Therefore, we are more interested in the generalization ability, which is the performance of a trained network on unseen samples. When a network is over-trained with too many parameters and training epochs, or too few training examples, good results may be produced for the training examples, but the network generalizes poorly. This is known as the overfitting phenomenon. In statistics, when a model has an excessive number of parameters and fits the noise in the data, overfitting occurs. A network's generalization capability is decided by three factors, namely, the training set size, problem complexity, and network architecture [94,95].

The generalization error consists of a bias and a variance term [96]. The bias term arises from the finite number of parameters of the model, known as the approximation ability of the network. The variance term arises from the finite number of training data. For an MLP with J_1 inputs and one output, the total generalization error is bounded as a function of the order of the number of hypothesis parameters, P, and the number of examples, N [97,98]. A larger P leads to a smaller approximation error due to a larger model, but also leads to a larger estimation error due to overfitting. When $P \propto N^{\frac{1}{3}}$, the best tradeoff between the approximation and estimation errors is achieved, and the generalization error for the MLP is maintained at $O\left(\frac{1}{P}\right)$ [97]. This result is in agreement with that of an MLP with a sigmoidal activation [99].

Cross-validation is a traditional way to measure the generalization ability of a trained network. In the simple cross-validation scheme, the dataset is partitioned into a training set and a test set. The training set is used for training the network, while the test set is used for evaluating the generalization ability of the trained network.

5.2. Generalization by Stopping Criterion

Overtraining can be avoided by stopping the training process before the global minimum is reached. Neural networks trained with iterative gradient descent gradually learn a mapping by increasing components of frequency. When stopping the training at a due point, the network will not learn the high-frequency noise. The training error always decreases with time, but the generalization error decreases to a minimum and then rises as the network is overtrained.

Cross-validation is performed to decide when to stop. Three early stopping criteria are empirically compared in [100]. A slower stopping criterion on average yields a small improvement in generalization, but requires a much longer training time [100]. A statistical analysis of overtraining on the three-layer MLP is given in [101].

5.3. Generalization by Regularization

Regularization [95,102–105] is a popular approach to improving generalization. Let $\vec{w}$ be the collection of all weights and bias terms. The training error $\mathcal{E}$ is a function of $\vec{w}$. A positive penalization term $\mathcal{E}_c(\vec{w})$ is added to the training objective to penalizes poor generalization. Then, the overall objective function is given by

$$\mathcal{E}_T(\vec{w}) = \mathcal{E}(\vec{w}) + \lambda_c \mathcal{E}_c(\vec{w}), \tag{33}$$

where λ_c is the regularization parameter. The regularization method is valid for the iterative gradient-based techniques as well as the one-step optimization such as a singular value decomposition (SVD). Regularization decreases the representation capability of the network, i.e., it increases the bias term but improves the variance term (bias–variance dilemma [96]).

Two most common regularizers employed to improve the generalization ability are, respectively, the weight-decay method [106,107] and the input perturbation method [102]. In the simple weight-decay, the constraint term is given by

$$\mathcal{E}_c(\vec{w}) = \sum_{i=1}^{P} w_i^2. \tag{34}$$

The inclusion of a weight-decay constraint term results in a small magnitude for the trained weights, yielding a smooth network output function and an improved generalization ability [106]. The input perturbation method [102] requires a robustness in the approximation ability to input perturbations. Let $o_{j,p}$ be the jth element of the network output with respect to the pth training sample, and $x_{i,p}$ be the ith element of the pth training input vector. In the input perturbation method [102], the constraint term is given by

$$\mathcal{E}_c(\vec{w}) = \sum_{p=1}^{N} \sum_{i=1}^{J_1} \sum_{j=1}^{J_M} \left(\frac{\partial o_{j,p}}{\partial x_{i,p}} \right)^2. \tag{35}$$

Network pruning techniques help to improve generalization as well [108,109]. At the end of the training of the weight-decay procedure, those connections with small weights can be removed from the network. One can improve generalization by training a network, when a small amount of noise (jitter) is added to the weights and inputs but no noise is added to the output [104,108–110]. In sum, the effect of adding noise in weights is similar to that of the weight-decay regularizer. On the other hand, the effect of adding noise in the input is similar to that of the input perturbation method [104,109].

Early stopping behaves like a simple weight-decay technique [104] in terms of the evolution of the effective number of weights. Weight sharing is used to reduce the number of network parameters, thus improving generalization [6,111].

The step size can be viewed as a regularization parameter [112]. The stochastic gradient method is analyzed in [113]. Regularization is controlled by the step size, the number of passes, and the minibatch size.

5.4. Selection of Regularization Parameter

The variance term can be controlled when the regularization parameter is appropriately chosen. However, too large a regularization parameter will yield an excessive bias term. Therefore, it is crucial to select a regularization parameter.

An appropriate regularization parameter can be selected by cross-validation [100,114,115]. The dataset is randomly divided into a training set and a validation (test) set, with the major portion of the dataset included in the training set and the remaining in the validation set. If only one sample is left for validation, the method is known as leave-one-out cross-validation. A number of networks are trained with different regularization parameters, and then the best trained network is selected based on cross-validation. However, cross-validation may be limited by the dataset size and it is a time-consuming process.

The regularization parameter can also be selected by generalization error estimation methods [94,95,116–122]. The methods are based on information criteria, such as the Akaike information criterion (AIC) [123], Schwartz's Bayesian information criterion (BIC) [124], and Rissanen's minimum description length (MDL) principle [125,126]. Most of these criteria can be expressed as a sum of two components, measuring the training error and penalizing the complexity, respectively [125,127]. For example, a good generalization can be realized when encoding the weights with short bit-lengths based on the MDL principle [110]. In these approaches, we have a generalization error estimation formula which is a function of the training error and trained weights. A number of networks are trained with different regularization parameters, and then the best trained network is chosen based on the estimated generalization errors.

The third approach is to estimate the regularization parameter during training based on a Bayesian framework [103,107,128,129]. In those works, the regularization term comes from the assumption that a prior probability is assigned to the weights. Then, the regularization parameter is related to the variance of the weights. In [103,128,129], the results focused on radial basis function (RBF) networks. The experiments in [128,129] successfully demonstrated the advantage of using this approach to select RBF centers and to automatically select the regularization parameter. In [107], the results can be applied to general MLPs.

6. Optimization on Network Size

Given a problem, an appropriate network size is very important. Too small a network will lead to underfitting, i.e., too large a bias term, corresponding to an inaccurate approximation. In contrast, too large a network will lead to overfitting, i.e., too large a variance term, corresponding to a poor generalization ability. This section introduces some approaches for finding an optimal network size.

6.1. Destructive Approach: Network Pruning

Network pruning starts with a network with a large size, then removes the redundant nodes or weights during or after training. Pruning methods can be either sensitivity-based or regularization-based methods [130,131]. In sensitivity-based methods, the sensitivity of the objective function $\mathcal{E}$ with respect to the removal of a weight or node is first estimated, and the least important weight or node is then removed. The regularization-based methods adds a regularization term to the training objective to punish a network of complex architecture. Some network pruning algorithms are surveyed in [130].

6.1.1. Sensitivity-Based Network Pruning

In the sensitivity-based pruning, the sensitivity measure is utilized to characterize the contribution of individual weights or nodes when solving a problem, and the less important weights or nodes are then removed. Let $\mathcal{E}_T$ be the training objective and w be a weight of the output of a hidden node. The normalized sensitivity is defined by

$$S_w^{\mathcal{E}_T} = \lim_{\Delta w \to 0} \frac{\frac{\Delta \mathcal{E}_T}{\mathcal{E}_T}}{\frac{\Delta w}{w}} = \frac{w}{\mathcal{E}_T} \frac{\partial \mathcal{E}_T}{\partial w}. \tag{36}$$

In the skeletonization technique [132], the sensitivity measure is defined as $S_w^{\mathcal{E}_T} = -w \frac{\partial \mathcal{E}_T}{\partial w}$, which utilizes the first-order approximation. This sensitivity measure

was applied in Karnin's pruning method, where no retraining procedure was applied [133]. This method was improved to avoid removing an input node or a particular hidden layer and to incorporate a fast retraining algorithm in [134], or to introduce the local relative sensitivity index in [135]. In [136], a loss-based sensitivity regularization is implemented by shrinking and then pruning parameters with low loss sensitivity. A node-sensitivity regularized node pruning scheme is proposed in [137].

Sensitivity-based methods that use linear models for hidden nodes are developed in [131,138], where redundant hidden nodes are well-modeled as linear models of their net inputs or linear combinations of the outputs of the other nodes. In [139], network pruning is implemented based on orthogonal transforms, e.g., SVD and QR decomposition. In [140], SVD is used to evaluate the significance of the number of hidden layer neurons of a trained network, based on which a pruning/growing technique is implemented. Pruning exploiting PCA of the node activations of successive layers is implemented in [141]. In [142], mutual information is used to prune both the input and hidden nodes. Other sensitivity-based methods utilizing retraining are described in [143,144].

6.1.2. Second-Order Expansion Pruning

Based on the second-order Taylor expansion of the objective function, the optimal brain damage (OBD) [145] and optimal brain surgeon (OBS) [146] procedures were developed around the early 1990's. Let $\vec{w}$ be the collection of all weights and biases. At the time of convergence, the gradient approaches zero and thus a change in $\vec{w}$ leads to an increase in $\mathcal{E}$:

$$\Delta\mathcal{E} \simeq \frac{1}{2}\Delta\vec{w}^T\mathbf{H}\Delta\vec{w}, \tag{37}$$

where $\mathbf{H} = \left[\frac{\partial^2\mathcal{E}}{\partial w_i \partial w_j}\right]$ is the Hessian matrix. A weight w_i can be removed by setting $\Delta w_i = w_i$ and all $\Delta w_j = 0$, $j \neq i$, and this introduces a change in $\mathcal{E}$.

OBS is derived from the saliency (37), while OBD is an OBS having diagonal $\mathbf{H}$. A weight with the smallest saliency is deleted. Optimal cell damage (OCD) [147] is obtained by first implementing OBD and further removing irrelevant input and hidden nodes. Unit-OBS [148] modifies OBS by removing a whole node at each step. Based on a block-diagonal approximation of the Hessian, principal components pruning [141] lies between OBD and OBS. Early brain damage (EBD) [149] extends OBD and OBS by implementing early stopping, and it allows the revival of the already pruned weights. Variance nullity pruning [150] is derived from an output sensitivity analysis. In case of a diagonal $\mathbf{H}$, OBD and the output sensitivity analysis are equivalent, if a gradient search and the MSE function are used.

The above approaches rely on the efficient computation of the Hessian matrix. In [151], an efficient procedure for calculating the Hessian matrix was described. Instead of an exact calculation of the Hessian matrix, nowadays, the approximation [152]

$$\mathbf{H} \approx \left[\sum_{p=1}^{N}\sum_{k=1}^{J_M}\frac{\partial o_{p,k}}{\partial w_i}\frac{\partial o_{p,k}}{\partial w_j}\right] \tag{38}$$

is usually used.

RLS-based pruning exploits the error covariance matrix obtained during RLS training [152]. RLS-based pruning and OBD have a similar number of pruned weights and similar generalization ability, but with a considerable computational saving and suitable for online situation. In a similar way, extended Kalman filtering (EKF) based pruning techniques for MLPs and recurrent neural networks are given in [153,154].

6.1.3. Regularization-Based Methods

The complexity of a trained network can be measured by its effective number of parameters [95]. The penalty approach aims at reducing the effective number of parameters [95]

by adding a regularization term into the objective function. Basically, there are two kinds of regularization approaches.

The first approach drives trained weights to have small magnitudes. The trained weights have a relatively flat distribution around zero. In this case, most trained weights have a contribution to the approximation ability of the trained network. In this approach, the regularization term is given by $\vec{w}^T \mathbf{R} \vec{w}$, where $\mathbf{R}$ is a positive-definite matrix. The standard weight-decay [155,156] is a representative method of this approach. Its regularization term is the sum of the squares of all the weights. This leads to the assumption that the a priori distribution [107] of weights is zero-mean Gaussian. During training, the decay effect of a weight is proportional to its magnitude. Hence, many trained weights are small but not very close to zero. The effective number of parameters of the trained network is small and the complexity of the trained network is small. The conventional RLS algorithm is actually a weight-decay algorithm [152], since they both use the same objective function, i.e., the sum of the squares of all the weights. The error covariance matrix from the RLS training has similar properties as $\mathbf{H}$. The initial value of the error covariance matrix can be used to control the generalization ability.

The second approach drives unnecessary weights to zero and removes them during training. In [157,158], the regularization term is the sum of the absolute values of the weights. This leads to the assumption that the a priori distribution of weights is zero-mean Laplacian. During training, the decay effect of a weight is a constant. Hence, many trained weights are very close to zero and a skeleton network is obtained.

Weight-smoothing regularization incorporates the Jacobian profile smoothness as a constraint [159]. Several other regularization techniques that use neural Jacobians are the double BP [160], the input perturbation [109], generalized regular network [161], and the approximate smoother [162].

When using the L_1-/L_2-norm of weight vectors at a group level, the method has a better generalization and pruning efficiency than weight decay, weight elimination, and approximate smoother [163]. In [164], a nonconvex integrated transformed L_1 regularizer applied to the weight matrix space is introduced to remove redundant connections and unnecessary neurons simultaneously. It is shown in [165] that compressing the weights of a layer has the same effect as compressing the input of the layer.

6.2. Constructive Approach: Network Growing

In the constructive approach, a small network is first selected and then hidden nodes are added until a specified performance is reached. The constructive approach has the ability to escape a local minimum by adding a new hidden node [166], since the process of adding neuron changes the shape of the error function. The weights of the newly added nodes can be randomly initialized.

Cascade-correlation learning [167] is a well-known constructive learning approach. The constructed network has direct connections between the input and output nodes. In the beginning, there is no hidden node. If the network cannot solve a problem after some training cycles, the one with the maximum covariance among a set of randomly initialized candidate nodes is added to the network. A newly added node is connected to the input nodes and all existing nodes. With previously trained nodes frozen, the network is trained by using the Quickprop algorithm [168]. The error signals are not backpropagated for training.

Many constructive algorithms are inspired from the cascade-correlation learning [169,170]. In [170], constructive BP allows the addition of multiple new nodes at a time, and continuous automatic structure adaptation, including both addition and deletion of nodes, can be performed. Cascade-correlation learning is not suitable for VLSI implementation due to the irregularity in the network architecture. By controlling the connectivity, cascaded-correlation learning generates a strictly layered architecture with a restricted fan-in and a small depth [171].

A quasi-Newton method for constructing the three-layer MLP was reported in [172]. The dependence identification (DI) algorithm [173] is a batch learning process. The MLP training problem is transformed into a set of quadratic optimization problems. In [174], a constructive training algorithm for the three-layer MLP is proposed for classification problems, where the Ho–Kashyap algorithm [37] is used to train the network and a pruning procedure is also incorporated. In [175], the proposed online constructive learning algorithm for the MLP integrates the weight scaling technique [176,177] for escaping local minima, and the quadratic programming (QP) and linear programming (LP)-based procedure for initializing the weights of a newly added neuron. In [178], the proposed constructive approach for MLP learning finds excellent solutions by exploiting the flat regions.

There are also some constructive methods for training MLPs with LTG neurons for classification, such as the tower algorithm [24], tiling algorithm [179], and upstart algorithm [180]. All three algorithms exploit the pocket algorithm [24].

7. Acceleration on BP

Due to the gradient-descent nature, the BP algorithm is slow in convergence. This section describes numerous measures for accelerating the convergence of the BP algorithm.

7.1. Eliminating Premature Saturation

Slow convergence is primarily due to the premature saturation of the output of the sigmoidal functions. When *net* has a large absolute value, the derivative of $\phi(net)$ is small and then the weight update is negligible. This leads to an excessively long learning time. This phenomenon is the well-known flat-spot problem. Premature saturation can be resolved by modifying the slope of the sigmoidal function, or by modifying the objective function.

An analysis of static premature saturation of output nodes is performed in [181]. This premature saturation arises from random initial weights at the beginning of training. In [182], a dynamic mechanism analysis for premature saturation identifies the momentum term as the leading cause of premature saturation. One can prevent premature saturation by temporarily modifying the momentum factor α.

In [183], the BP update rule is revised by adding a term relating to the degree of saturation to the objective function. The additional term is a parabolic function of the outputs of the output nodes, given by $\sum_{i=1}^{J_M} \lambda(o_i - 0.5)^n$, where λ is a scale factor, n is an exponent, and o_i is the actual output of the ith node in the output layer. Similarly, in [184], the partial derivatives of the logistic activation function are generalized so that the error signals are significantly enlarged when the outputs of the neurons (hidden or output) approach saturation. Other authors also modify the objective functions to avoid premature saturation [76,185]. However, the main drawback of those approaches is that there is no selection guide for the tunable parameters. For example, in [183], the selection rules for λ and n were not adjusted.

7.2. Adapting Learning Parameters

The performances of BP and BPM rely on the values of the learning rate η and the momentum parameter α. There are some heuristics for accelerating the learning by adaptively adjusting η and α. According to [6], the process of starting with a large η and then gradually decreasing it resembles simulated annealing [186]. For gradient-based methods, the learning parameters are usually updated once every epoch.

7.2.1. Globally Adapted Learning Parameters

All the weights are updated by using the global learning parameters, namely, the learning rate η and momentum parameter α. In [187], η has its optimal value as the reciprocal of the largest eigenvalue of the Hessian $\mathbf{H}$ of the objective function. The largest eigenvalue is estimated by an online algorithm, without the calculation of the Hessian.

Instead of using complicated algorithms to update the learning parameters, a simple and popular search-then-converge schedule is developed in [188]. At the beginning, the

schedule selects a large η, then decreases it gradually during the learning process. In the bold-driver technique [189,190], the learning rate η and momentum parameter α are varied by four rules according to whether or not an update when training an epoch improves the training objective. The approach is able to escape from local minimum.

The BP rule, or the gradient-descent step, is formulated as

$$\vec{w}(t+1) = \vec{w}(t) - \eta_t \vec{g}(t), \tag{39}$$

where the gradient $\vec{g}(t) = \left.\frac{\partial \mathcal{E}}{\partial \vec{w}}\right|_{\vec{w}=\vec{w}(t)}$. In [191], fast convergence is achieved by adapting η for the batch BP according to the instantaneous values of $\mathcal{E}(t)$ and its gradient. The method, however, leads to a jumpy behavior of the weights, since η is very large in the neighborhood of a local or global minimum. In [192], both η_t and α are updated adaptively by a correlation-based heuristic. This algorithm achieves an exponential change of η, and outperforms the locally adapted learning-rate-based method proposed in [193].

In [194], η is derived from the local approximation of the Lipschitz constant. The algorithm is robust against the jumpy behavior [191] and ensures that the objective function decreases for every weight update. It outperforms BP, delta-bar-delta [195], and the bold-driver technique [189].

The learning parameters are also adapted by the fuzzy inference system (FIS) [196]. The method incorporates Jacobs' heuristics [195] on the unknown learning parameters by using fuzzy IF–THEN rules. The heuristics are driven by the behavior of $\mathcal{E}(t)$. η and α are, respectively, adjusted by an FIS. Fuzzy BP converges much faster than BP, with much smaller MSE [196].

7.2.2. Locally Adapted Learning Parameters

In the locally adapted learning approach, each weight or bias term w_i can have its own learning rate $\eta_{t,i}$ so that

$$\Delta w_i(t) = -\eta_{t,i}\, g_i(t)\,. \tag{40}$$

Locally adaptive learning algorithms may use weight-specific learning rates, such as the methods proposed in [193,197], SuperSAB [198], delta-bar-delta [195], Quickprop [168], incremental equalized error BP (EEBP) [199], and the globally convergent strategy [200].

In [197], the authors set the learning rate corresponding to each input weight to a neuron to be inversely proportional to the neuron's fan-in. This balances the learning speeds of nodes with different fan-ins. The method increases the convergence speed, and its justification is verified from the eigenvalue distribution of $\mathbf{H}$ [201].

In [193] and in SuperSAB [198], $\eta_{t,i}$ is heuristically adapted by

$$\eta_{t+1,i} = \begin{cases} \eta_0^+ \eta_{t,i}, & \text{if } g_i(t) \cdot g_i(t-1) > 0 \\ \eta_0^- \eta_{t,i}, & \text{if } g_i(t) \cdot g_i(t-1) < 0 \end{cases}, \tag{41}$$

where $\eta_0^+ > 1$, $0 < \eta_0^- < 1$, and in SuperSAB, $\eta_0^+ \simeq \frac{1}{\eta_0^-}$. Since $\eta_{t,i}$ changes exponentially, multiple successive acceleration steps will yield too large or too small $\eta_{t,i}$. For this reason, SuperSAB includes a momentum term. Delta-bar-delta [195] has a similar update for $\eta_{t,i}$, but implements linear acceleration and exponential deceleration. When the momentum term is included, delta-bar-delta diverges sometimes, and an adaptive momentum can be introduced to improve delta-bar-delta [202]. However, the selection of the parameters is more difficult for delta-bar-delta.

In [203,204], $\eta_{t,i}$ and $\alpha_{t,i}$ are derived using the second-order-based, first-order-based, and CG-based approaches, utilizing the first- and second-order derivatives of $\mathcal{E}$ with respect to $\eta_{t,i}$ and $\alpha_{t,i}$. The computational and storage complexities are at most three times that of standard BP, but with a tenfold speed. In [200], based on Wolfe's conditions for linear search and the Lipschitz condition, the authors provide a general theoretical result for developing batch BP with local learning rates. Conditions for global convergence are given.

Quickprop [167,168,205] heuristically adapts $\alpha_{t,i}$. The method uses error gradient at two consecutive time steps, which is essentially a discrete approximation to second-order derivatives. Thus, Quickprop is a quasi-Newton method. Fahlman improved the flat-spot problem by adding 0.1 to the derivative of the sigmoidal function [168].

7.3. Initializing Weights

The initial weights should be selected as close as possible to a global minimum before training. Improper weight initialization may lead to slow convergence, and the network may get stuck at a local minimum. All the weights are typically initialized with small random positive values, or with small zero-mean random numbers [6]. Randomness in weights helps to break the symmetry and thus reduce redundancy in the network. Experiments have demonstrated the extreme sensitivity of BP to the initial weight configuration, which has a complex fractal-like structure [206].

In [207], the authors give the maximum amplitude for the initial weights from a statistical analysis. It is theoretically verified in [181] that when the maximal value of the weights increases, the probability of a neuron becoming prematurely saturated increases.

7.3.1. Heuristics for Weight Initialization

An empirical optimal initialization of the weights is to set the weights terminating at a node to be uniformly distributed in $\left[-3/\sqrt{n}, 3/\sqrt{n}\right]$, where n is the number of weights terminating at the node [208,209]. A similar idea is implemented in [210]. A number of heuristics for weight initialization are also discussed and compared by comprehensive simulations in [208]. In [208], the authors recommend a weight range of $[-0.77, 0.77]$ according to their empirical result.

In [211], for the three-layer MLP, the maximum magnitude of the weights between the hidden and output layers is inversely proportional to the hidden-layer fan-in, based on a multidimensional geometry analysis. The mini-max initialization technique [71] approximates continuous functions by using the number of extrema to characterize the complexity of a function.

7.3.2. Weight Initialization Using Parametric Estimation

In [212], a three-layer MLP with prototypes is initialized by clustering the normalized augmented pattern vectors of unit length. In [213], hidden-layer weights are initialized by using clustering and nearest neighbors, and the output-layer weights are initialized by an SVD. In [214], the weights are forward-layerwise-initialized using eigenvectors of the cross-moment matrix based on Stein's identity, and the weights for the output layer are initialized by generalized linear modeling. In [215,216], the initial hidden-layer weights are calculated in such a way that each hidden node approximates a range of the desired function and the sigmoidal function is replaced by the piecewise-linear approximation. In [217,218], the outputs of the hidden and output neurons are assigned in the nonsaturation region, and the initial weights are estimated using the least squares and linear algebraic methods. In [219], the weight initialization is implemented by using the orthogonal least squares (OLS) method. In [220], two weight initialization techniques that combine random weight initialization with pseudoinverse are given.

An ICA-based weight initialization for the three-layer MLP was derived in [221]. The hidden-layer weights were initialized by extracting the salient feature components from the input data. A weight initialization based on discriminant learning is proposed in [222]. In [223], the maximum covariance initialization method implements a weight initialization procedure similar to the cascade-correlation algorithm [167].

A weight initialization technique based on Taylor series expansion is mathematically well-founded [224]. Based on a Taylor series development of a nonlinear mapping between the input and output of the examples and the sigmoidal function, two weight initialization techniques for the three-layer MLP have been derived from the first- and second-order identifications of the mapping [224].

7.4. Adapting Activation Function

When a node has a large net input, *net*, it is close to a saturation state. In this case, the first-order derivative of the sigmoidal function is very small, leading to very slow weight update. This can be easily solved by adding a bias, say 0.1, to $\dot{\phi}(net)$ [168]. It is also suggested that the designed error function should go to infinity when $\dot{\phi}(net) \to 0$ [225] such that a finite nonzero error update is achieved. The flat-spot problem can be effectively solved by defining such an activation function, $\phi_\mu(net) = \mu net + (1-\mu)\phi(net)$, with $\mu \in [0,1]$ [226]. When starting with $\mu = 1$, every node has a linear activation function, thus the objective function $\mathcal{E}(\vec{w})$ is a polynomial of the weight vector $\vec{w}$ and has few local minima. BP is used to find a local minimum in $\mathcal{E}$, then μ is gradually decreased and BP is applied at the same time until $\mu = 0$. This is an annealing process, which helps to reach a global or suboptimal minimum.

For the logistic and hyperbolic tangent functions in (4) and (5), the gain β is the slope of the activation function. In BP, typically, $\beta = 1$. Modified BP with adaptive β has improved performance in terms of learning speed and generalization [227–229]. According to a theorem given in [230], increasing β has the same effect as increasing η.

A sigmoidal activation function with a wide linear part, called the log-exp function, was derived in [231]. The MLP with such an activation function can learn quickly since the wide nonsaturation region helps to prevent premature saturation. The extended linear part is particularly suitable for implementation. In order to improve BP, a fuzzy system for adapting the gain β is given in [230].

7.5. Other Acceleration Techniques

Gradient reuse is a simple strategy for improving the convergence speed [232]. Gradients computed are reused several times until the weight updates do not decrease the objective function. Batch mode is used to generate a more accurate estimate of the true gradient. However, this method is valid only for simple problems [233]. The weight extrapolation technique can accelerate BP by extrapolating each individual weight [234]. This procedure is easy to implement and is activated only a few times during BP iterations.

Inspired by the common PID control algorithm in feedback control, a three-term BP algorithm [235,236] introduces a proportional term to the BP update rule. It can be implemented in both batch and online modes. Compared to BP, it is easy to select learning parameters in three-term BP, and it has a much higher convergence speed and can escape local minima easily.

Successive approximation BP [237] can effectively avoid local minima. Training is performed in N_{ph} successive BP learning phases, each terminated at a predefined accuracy δ_i, $i = 1, \ldots, N_{\text{ph}}$. The overall training error is given by $\mathcal{E} < 2^{N_{\text{ph}}} \prod_{i=1}^{N_{\text{ph}}} \delta_i$. The method significantly outperforms BP in terms of convergence speed and generalization.

For gradient descent with delayed updates, the effects of the delay become negligible after a few iterations and the algorithm converges at the same optimal rate as standard gradient descent [238]. This is particularly important for distributed parallel implementations.

Resilient propagation (RProp) [239] is one of the best performing first-order methods, although it is derived from a heuristic. It is a batch-learning algorithm. Each weight is updated according to the sequence of signs of the partial derivatives in each weight w_i. RProp is not sensitive to the initial parameters. It converges much faster and has much lower computational complexity than BP, Quickprop [168], and SuperSAB [198]. RProp is comparable to the CG method in terms of performance [240]. It is suitable for hardware implementation and is not susceptible to numerical problems. There are a number of variants of RProp [241].

8. Second-Order Acceleration

The first-order BP learning converges slowly if the error surface is flat in a dimension. Second-order optimization methods theoretically provide significantly faster convergence.

The Hessian $\mathbf{H}$ is utilized in second-order methods. The calculation of $\mathbf{H}$ can be implemented based on information from the BP algorithm [151].

Second-order methods can be categorized into matrix-type and vector-type. Matrix-type methods need to store the Hessian and its inverse. However, for MLPs, $\mathbf{H}$ is ill-conditioned [242]. Newton's methods and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [243] are matrix-type methods, and they converge faster than BP typically by two orders of magnitude. The computational complexity is not less than $O(P^2)$. Vector-type methods need to store a few vectors. Some representative algorithms are limited-memory BFGS [244], one-step secant (OSS) [78,245], CG algorithms [246,247], and scaled CG [248]. They converge faster than BP by typically one order of magnitude. In these algorithms, the Hessian is calculated iteratively, or its structure is implicitly exploited. Line search or trust-region search is implemented.

Second-order methods are implemented in batch mode to maintain the numerical sensitivity of the calculated second-order gradients. Learning parameters are automatically adapted. However, second-order methods become trapped in a local minimum more frequently than the BP algorithm [249].

8.1. Newton's Methods

To implement Newton's methods, one has to explicitly compute and store the Hessian [78,250]. As variants of the classical Newton's method, Newton's methods are matrix-type algorithms. Some examples are the classical Newton, Gauss–Newton, and Levenberg–Marquardt (LM) methods. They are quadratically convergent.

Classical Newton's method is obtained by approximating $\mathcal{E}(\vec{w})$ at time $t+1$ into its Taylor's expansion at time t, and ignoring the third- and higher-order terms. With the approximation, the updating on the weight vector is given by

$$\vec{w}(t+1) = \vec{w}(t) + \vec{d}(t)\,, \tag{42}$$

where

$$\vec{d}(t) = -\mathbf{H}^{-1}(t)\vec{g}(t) \tag{43}$$

and the gradient vector is given by $\vec{g}(t) = \frac{\partial \mathcal{E}}{\partial \vec{w}}\Big|_{\vec{w}=\vec{w}(t)}$. For the MLP, the Hessian is nearly singular [251], and thus directly using (43) to find $\vec{d}(t)$ is not practical. Instead, we can solve the following set of linear equations for $\vec{d}(t)$ by using an SVD or QR decomposition [252]

$$\vec{g}(t) = -\mathbf{H}(t)\vec{d}(t)\,. \tag{44}$$

In the classic Newton method, the computation of $\mathbf{H}(t)$ needs global information and $O(P^3)$ floating-point operations are required for computing the search direction.

8.1.1. Gauss–Newton Method

The Gauss–Newton method is obtained by reformulating $\mathcal{E}(\vec{w})$. The updating equations have the same form as (42) and (43), but $\vec{g}(t)$ and $\mathbf{H}(t)$ are, respectively, defined by

$$\vec{g}(t) = \mathbf{J}^T(t)\ \vec{\epsilon}(t), \tag{45}$$

$$\mathbf{H}(t) = \mathbf{H}_{\mathrm{GN}}(t) = \mathbf{J}^T(t)\ \mathbf{J}(t), \tag{46}$$

where $\vec{\epsilon}(t) = \vec{\epsilon}(\vec{w}(t)) = (\epsilon_1, \epsilon_2, \cdots, \epsilon_N)^T$, $\epsilon_p = \left\|\vec{e}_p\right\|$ with $\vec{e}_p = \vec{o}_p - \vec{y}_p$, the Jacobian matrix $\mathbf{J}(t) = \mathbf{J}(\vec{w}(t)) = \frac{\partial \vec{\epsilon}(\vec{w})}{\partial \vec{w}} = [J_{ij}]$, and $J_{ij} = \frac{\partial \epsilon_i}{\partial w_j}$. The Hessian $\mathbf{H}(t)$ is approximated using first-order derivatives only. It is possible that $\mathbf{J}$ is ill-conditioned and $\mathbf{H}$ is noninvertible. In such a case, we can use the iterative Gauss–Newton method [243] based on the generalized secant method.

8.1.2. Levenberg–Marquardt Method

The LM method [253] adds a small identity matrix to $\mathbf{H}$ to eliminate the possible singularity. The quadratic approximation error to $\mathcal{E}(\vec{w})$ is minimized at step t under the constraint that the step length $\|\vec{d}(t)\|$ is within a trust region, by using the Karush–Kuhn–Tucker (KKT) theorem [243]. That means, we use a modified Hessian, given by

$$\mathbf{H}_{\mathrm{LM}}(t) = \mathbf{H}(t) + \sigma(t)\mathbf{I}\,, \tag{47}$$

where $\sigma(t) > 0$ is a small positive number, which determines the size of the trust region. The LM method, as a modification of the Gauss–Newton method, is obtained by replacing $\mathbf{H}_{\mathrm{GN}}$ by $\mathbf{H}_{\mathrm{LM}}$ [78,254]. Thus, $\mathbf{H}_{\mathrm{LM}}$ is always invertible.

For large σ, the LM method becomes the BP with $\eta = \frac{1}{\sigma}$; on the other hand, for small σ, it degenerates to the Gauss–Newton method. Thus, one has to balance between the guaranteed convergence of the gradient descent and the rapid convergence of the classical Newton method. In [254], an adaptive rule on σ was developed, given by

$$\sigma(t) = \begin{cases} \sigma(t-1)\gamma, & \text{when} \quad \mathcal{E}(t) \geq \mathcal{E}(t-1) \\ \frac{\sigma(t-1)}{\gamma}, & \text{when} \quad \mathcal{E}(t) < \mathcal{E}(t-1) \end{cases}, \tag{48}$$

where a constant factor was applied, $\gamma > 1$. A typical selection is 0.01 for $\sigma(0)$ and 10 for γ [254]. The Jacobian $\mathbf{J}$ can be calculated by a modification to BP [254]. $\sigma(t)$ can also be adapted according to the hook step [253], Powell's dogleg method [243], and other heuristics [243].

The LM method is an efficient algorithm for medium-sized neural networks [254]. It requires a large space to store the Jacobian, the approximated Hessian, and the inverse of a $P \times P$ matrix at each iteration. Some LM variants, such as the trust-region-based error aggregated training (TREAT) algorithm [255] and a modified LM method given in [256], aim at reducing the storage and computational complexity. The recursive LM given in [257] is used for online training of neural networks. In [258], the LM method is tailored for an arbitrary network architecture.

In [259], the quasi-Hessian matrix and gradient vector are computed directly, and the Jacobian is not used. The computation of the quasi-Hessian matrix and gradient vector reduces the memory requirements by a factor of NJ_M, where J_M is the number of output nodes. In [260], a forward-only LM method utilizes the forward-only computation, while in traditional methods, the Jacobian is calculated by a forward and backward computation. In [261], backpropagation is used for the matrix–matrix multiplication, reducing the computation time of the LM method by a factor of $O(J_M)$. In a modified ML algorithm [262], a singularity point in the learning rate is eliminated and there is only one learning rate. The error stability and weights boundedness are assured by the Lyapunov technique.

8.1.3. Other Methods

When implementing Newton's methods for MLP learning, $\mathbf{H}$ is difficult to calculate iteratively, and thus the cost for calculating $\mathbf{H}^{-1}$ is also high. For Newton's methods, the diagonal terms of $\mathbf{H}$ also have the ill-representability problem, and a good initial estimate of the weights is also required. In block Hessian-based Newton's method [251], a block Hessian $\mathbf{H}_{\mathrm{b}}$ is defined to approximate $\mathbf{H}$.

In the LM case, the calculation of the inverse of $\mathbf{H}_{\mathrm{b}} + \lambda\mathbf{I}$ can be broken into the calculation of the inverse of each diagonal block $\mathbf{H}_{\mathrm{b}}^{(m)} + \lambda\mathbf{I}$; thus, the problem is decomposed into $M-1$ subproblems, each corresponding to a layer. The inverse in each of the subproblems can be recursively calculated by the matrix inversion lemma.

LM with adaptive momentum (LMAM) and optimized LMAM [263] have the merits of both LM and CG methods, and thus they can help LM escape local minima. LMAM is globally convergent. Both LMAM and optimized LMAM outperform LM, BFGS, and Polak–Ribiere CG with restarts [246].

The attractor-based trust-region method [264] alternates its two phases for MLP learning. The first phase implements a trust-region-based local search for rapid training and global convergence [243], whereas the second phase implements an attractor-based global search to escape from local minima by using a quotient gradient system [265]. Alternating the two phases results in a fast convergence to global optimum. This algorithm outperforms BPM, BP with tunneling [266], and LM algorithms in terms of number of epochs, training time, and MSE [264].

Stochastic second-order methods are attractive due to their low computational cost in each iteration. However, the performance is highly dependent on the approximation of the Hessian. In [267], an accelerated regularized subsampled Newton method is proposed based on Nesterov's acceleration.

8.2. Quasi-Newton Methods

In quasi-Newton methods, the Hessian or its inverse is approximated iteratively. The Hessian of an MLP is symmetric and usually positive definite. Quasi-Newton methods having positive-definite Hessian are referred to as variable-metric methods. As a class of variable-metric methods, secant methods approximate the Hessian using differences. Quasi-Newton methods have the same storage requirement as Newton's methods.

Two globally convergent strategies, namely, line search and trust-region search, are employed in quasi-Newton methods. These strategies retain the rapid convergence property of the classical Newton's method and are generally applicable [243]. In the line-search strategy, given the direction $\vec{d}(t)$ from (43) or (44) based on the approximated Hessian, the updating is given by

$$\lambda(t) = \arg\min_{\lambda \geq 0} \mathcal{E}\left(\vec{w}(t) + \lambda \vec{d}(t)\right), \tag{49}$$

$$\vec{w}(t+1) = \vec{w}(t) + \lambda(t)\vec{d}(t)\ . \tag{50}$$

The line-search strategy guarantees that the objective function decays at each iteration. The optimal $\lambda(t)$ can be derived as a representation using the Hessian. Inexact line-search and line-search-free methods are also implemented in quasi-Newton methods for MLP learning [268].

Secant methods can be of rank one or rank two. The Broyden family includes many rank-two and rank-one methods [243]. The Davidon–Fletcher–Powell (DFP) and BFGS methods are dual rank-two secant methods in the Broyden family. BFGS has become the most popular variable-metric method [269]. Many properties of DFP and BFGS are common to the Broyden family.

8.2.1. BFGS Method

In BFGS [243,244,269], the Hessian or its inverse is updated iteratively. The inverse of the Hessian is updated by

$$\begin{aligned}\mathbf{H}^{-1}(t+1) = \mathbf{H}^{-1}(t) &+ \left(1 + \frac{\vec{z}^T(t)\,\mathbf{H}^{-1}(t)\,\vec{z}(t)}{\vec{s}^T(t)\,\vec{z}(t)}\right)\frac{\vec{s}(t)\vec{s}^T(t)}{\vec{s}^T(t)\vec{z}(t)} \\ &- \left(\frac{\vec{s}(t)\vec{z}^T(t)\mathbf{H}^{-1}(t) + \mathbf{H}^{-1}(t)\vec{z}(t)\vec{s}^T(t)}{\vec{s}^T(t)\vec{z}(t)}\right),\end{aligned} \tag{51}$$

where

$$\vec{z}(t) = \vec{g}(t+1) - \vec{g}(t)\,, \tag{52}$$

$$\vec{s}(t) = \vec{w}(t+1) - \vec{w}(t)\,. \tag{53}$$

The initial values $\vec{w}(0)$, $\vec{g}(0)$, and $\mathbf{H}^{-1}(0)$ need to specified. Typically, $\mathbf{H}^{-1}(0)$ is set as the identity matrix. BFGS has a computational complexity of $O(P^2)$, and it needs to store matrix $\mathbf{H}^{-1}$ as well.

The BFGS as well as all the secant methods satisfies the quasi-Newton condition or secant relation [243,269], given by

$$\mathbf{H}^{-1}(t+1)\vec{z}(t) = \vec{s}(t). \tag{54}$$

When using an inexact line search in BFGS, the number of error function evaluations will decrease substantially.

8.2.2. One-Step Secant Method

OSS method [78,245] is a memoryless BFGS.

It is derived by inserting $\mathbf{H}^{-1}(t) = \mathbf{I}$, I being the identity matrix, into the BFGS update equation given by (51), and then multiplying both sides by $-\vec{g}(t+1)$ to get the search direction. As such, the Hessian is not stored, and the calculation of a matrix inverse is not needed when calculating a new search direction. OSS has a superb computational complexity of $O(P)$. The price paid is a considerable loss of second-order information, leading to a slow convergence compared to the BFGS.

When implementing an exact line search, OSS generates conjugate directions. Parallel implementations of BFGS and OSS are treated in [270]. In [271], a parallel implementation of a secant method using inexact search is developed for MLP learning.

8.2.3. Other Secant Methods

In limited-memory BFGS methods, second-order information from the most recent iterations are exploited to implement part of the Hessian approximation [244]. Limited-memory BFGS algorithms typically have a memory complexity of $O(P)$ and do not need accurate line searches [272]. Other examples of quasi-Newton variants are variable-memory BFGS [273], memory-optimal BFGS methods [274], and the trust-region implementation of the BFGS method [275].

In [276], the proposed limited-memory quasi-Newton methods exploit an iterative scheme of a generalized BFGS type method, and approximate the Hessian by using a rank-two formula arising from a fast unitary transform. They have a computational complexity of $O(P\log(P))$ and a memory complexity of $O(P)$. BFGS and CG have a close relationship, which helps to formulate algorithms with variable or limited memory [269]. One can treat memoryless or limited-memory quasi-Newton methods as a tradeoff between the CG and quasi-Newton methods.

8.3. Conjugate-Gradient Methods

The CG method [78,248,277,278] is a popular alternative to BP. CG implements a series of line searches along noninterfering directions that are built to utilize the Hessian structure but not store it. The method constructs a sequence of P successive search directions that satisfy the so-called **H**-conjugate property. The memory requirement is $O(P)$, which is nearly four times that of BP [234]. The computational complexity per weight update cycle is much higher than that of BP. This is because line search with an appropriate step size requires multiple evaluations of $\mathcal{E}$ or its derivative, which require the presentation of the entire training set.

In the CG method for MLPs, the update is given by

$$\vec{w}(t+1) = \vec{w}(t) + \lambda(t)\vec{d}(t), \tag{55}$$

$$\vec{d}(t+1) = -\vec{g}(t+1) + \beta(t)\vec{d}(t), \tag{56}$$

where $\lambda(t)$ is the exact step size to the minimum of $\mathcal{E}$ along $\vec{d}(t)$ and it is obtained by a linear search according to (49), and $\beta(t)$ is a step size for determining the direction $\vec{d}(t+1)$.

The speed of CG is critically dependent upon the efficiency of the line search. Fast CG algorithms with inexact line searches [272,279] have been used for MLP learning [280].

Scaled CG [248] avoids the line search by using a scalar to keep the Hessian **H** positive definite, as implemented in the LM method.

There are many choices of $\beta(t)$, such as the Hestenes–Stiefel [277], Fletcher–Reeves [281], Polak–Ribiere [282], Dai–Yuan [283], and conjugate descent [243] methods. The popular Polak–Ribiere method is given by [282]

$$\beta(t) = \frac{\vec{g}^T(t+1)\vec{z}(t)}{\vec{g}^T(t)\vec{g}(t)}, \tag{57}$$

where $\vec{z}(t)$ is given by (52). Usually, $\vec{w}(0)$ is selected as a random vector, and $\vec{d}(0) = -\vec{g}(0)$. The process is terminated when $\|\vec{g}(t)\|$ is small enough. The computational complexity of the CG method is $O(P)$.

The CG method can be treated as an extended version of BPM by an automatic selection of the learning parameters $\eta(t)$ and $\alpha(t)$ in each epoch [203,247,278,284]. Compared to BP, the CG method gets more easily stuck at a bad local minimum, since it deterministically moves toward the bottom of any valley it reaches, due to the line-search procedure [234,285]. The CG method is usually executed multiple times with different random $\vec{w}(0)$, and finally the $\vec{w}$ yielding the minimum error is taken as the solution [234].

There are some examples of the CG method in MLP learning. In [286], the MLP is decomposed into a set of Adalines. Each Adaline has a local MSE function. The desired output of each Adaline is calculated by error backpropagation. Each Adaline is learned by using a modified line-search-free CG. The localized method is typically one order of magnitude faster than the CG-based global method. In [287], based on the spectral scaling parameter, a self-scaled CG for MLP learning is derived from the principles of several CG methods.

8.4. Extended Kalman Filtering Methods

The Kalman filtering method is an optimum online estimation method for linear systems. The EKF method is obtained from the linearization of a nonlinear system. EKF is a general-purpose incremental training method, and can be used for training any MLP. Recursive least squares (RLS) can be regarded as a reduced form of EKF, and is more popular in adaptation.

8.4.1. Extended Kalman Filtering

In the EKF approach, learning is treated as the parametric identification problem of a nonlinear system. The weight vector $\vec{w}$ is regarded as a state vector, and the MLP is regarded as an unknown mapping $\vec{f}(\cdot)$ from the state vector and the input $\vec{x}(t)$ onto the output. The behavior of the MLP is described by the following state equations, given by

$$\vec{w}(t+1) = \vec{w}(t) + \vec{v}(t), \tag{58}$$

$$\vec{y}(t) = \vec{f}(\vec{w}(t), \vec{x}(t)) + \vec{\epsilon}(t), \tag{59}$$

where $\vec{x}(t)$ is the input, $\vec{y}(t)$ the observed or desired output, $\vec{v}$ the observation noise, and $\vec{\epsilon}$ the measurement noise. $\vec{v} = \mathcal{N}(0, \mathbf{Q}(t))$ and $\vec{\epsilon} = \mathcal{N}(0, \mathbf{R}(t))$ are assumed to be Gaussian with zero mean and covariance matrices $\mathbf{Q}(t)$ and $\mathbf{R}(t)$, respectively.

EKF estimates the weight vector $\vec{w}$ such that the sum of the squared prediction errors of all past observations is minimized. It is a minimum variance estimator derived from the Taylor series expansion of $\vec{f}(\cdot)$ in the vicinity of the previous estimate. EKF [153,154,288–290] is given by

$$\vec{w}(t+1) = \vec{w}(t) + \mathbf{K}(t+1)\left[\vec{y}(t+1) - \vec{f}(\vec{w}(t), \vec{x}(t+1))\right], \tag{60}$$

$$\mathbf{K}(t+1) = \mathbf{P}(t)\mathbf{F}^T(t+1)\left[\mathbf{F}(t+1)\mathbf{P}(t)\mathbf{F}^T(t+1) + \mathbf{R}(t+1)\right]^{-1}, \tag{61}$$

$$\mathbf{P}(t+1) = \mathbf{P}(t) - \mathbf{K}(t+1)\mathbf{F}(t+1)\mathbf{P}(t) + \mathbf{Q}(t), \tag{62}$$

where $\mathbf{K} \in \mathbb{R}^{P \times N_y}$ is the Kalman gain, N_y the dimension of $\vec{y}$, $\mathbf{P} \in \mathbb{R}^{P \times P}$ the conditional error covariance matrix, $\vec{f}(\vec{w}(t), \vec{x}(t+1))$ the estimated output, and $\mathbf{F}(t+1) = \left.\frac{\partial \vec{f}}{\partial \vec{w}}\right|_{\vec{w}=\vec{w}(t)} \in \mathbb{R}^{N_y \times P}$. Usually, $\mathbf{P}(0) = \frac{1}{\varepsilon}\mathbf{I}$, $\vec{w}(0) = \mathcal{N}(0, \mathbf{P}(0))$, and a small number $\varepsilon > 0$.

The above method is the global EKF method for MLP learning [289,291]. BP is a degenerate form of EKF [288]. Some fast algorithms for MLP learning are fading-memory EKF (FMEKF) and U-D factorization-based FMEKF (UD-FMEKF) [292], and the EKF variant reported in [293]. By partitioning the learning task into many small-scaled, separate, localized identification subproblems, localized EKF methods effectively reduce the complexity of the EKF method. There are some localized EKF methods, such as the multiple extended Kalman algorithm (MEKA) [294] and the decoupled EKF algorithm (DEKF) [295]. For localized algorithms, those off-diagonal terms in $\mathbf{P}$ in the corresponding global EKF method are set to zero to remove coupling.

The Kalman filtering method is based on the assumption of a Gaussian noise. The extended H_∞ filtering (EHF) method [296] extends the EKF method to enhance the robustness to non-Gaussian noise.

8.4.2. Recursive Least Squares

When $\mathbf{R}(t)$ reduces to the identity matrix $\mathbf{I}$ and $\mathbf{Q}(t)$ to the zero matrix $\mathbf{O}$, the EKF method reduces to the RLS method. The RLS method has been applied to MLP learning in [152,291,297,298]. It is typically an order of magnitude faster than the first-order online BP. For the same accuracy, RLS requires tenfold less epochs than BP [298].

In [152], RLS is derived from the minimization of the sum of the squared prediction errors of all prior observations plus an additional constraint on weights. For the same initialization condition as that of EKF, RLS is inherently a weight-decay technique, and the weight-decay effect is controlled by $\mathbf{P}(0)$, typically $\mathbf{P}(0) = \frac{1}{\varepsilon}\mathbf{I}$. Usually, for smaller ε, the training accuracy is better, whereas for larger ε, a better generalization is achieved [152]. Two modified RLS algorithms, namely true weight-decay RLS and input perturbation RLS, were developed to improve the generation ability in [299]. Generalized RLS [300] has a general decay term in the energy function. It has the same computational complexity as RLS. Neural networks trained by generalized RLS have a substantially improved generalization ability.

Like localized EKF, a complex problem can be divided into multiple localized subproblems, each of which is solved by RLS. There exist some localized RLS algorithms, such as the local linearized LS (LLLS) method [301] and the block RLS (BRLS) algorithm [302].

9. Other Learning Algorithms

The expectation–maximization (EM) method [16] is the most popular method for providing the maximum-likelihood (ML) solution for the parameters in case of incomplete data. The EM method has been used for MLP learning [303,304]. The EM approach is an iterative statistical technique. In the E-step, the conditional expectation of hidden nodes is calculated based on the observation and current weight vector. In the M-step, a better weight vector is updated based on the conditional expectation of hidden nodes. The BP algorithm is a special case of the generalized EM algorithm for an iterative ML estimation. Injecting carefully chosen noise can speed the average convergence of the EM method as it climbs a hill of probability or log-likelihood [305]. Noise injection is also applied to bidirectional BP, which trains a neural network with backpropagation in both the backward and forward directions using the same synaptic weights [306].

Amari also used the information geometry to explain the learning in MLPs [304]. In their formulation, the learning was regarded as an alternative projection in the manifold of an exponential family.

In a non-Euclidean parametric space with a Riemannian metric structure, the ordinary gradient is not along the steepest direction of the cost function. In such a case, the natural gradient gives the steepest direction [17]. Natural gradient descent can be viewed as a

type of second-order optimization method, with the Fisher information matrix acting as a substitute for the Hessian [307]. In many cases, the Fisher information matrix is equivalent to the generalized Gauss–Newton matrix, which approximates the Hessian [307]. Natural gradient descent has a linear convergence in a Riemannian weight space. Online natural gradient learning yields the Fisher efficient estimator; thus, it converges to the optimal batch mode asymptotically. The flat-spot problem, which occurs in BP, will not occur, when using natural gradient [17]. A natural conjugate gradient method for MLP learning is discussed in [308].

MLPs can be trained by iterative layerwise learning methods [309–313]. A weight update is implemented layerwise, and at each layer the weight is updated by solving a set of linear equations. These algorithms typically converge one to two orders of magnitude faster than BP. The layerwise approach can be jointly used with BP [309,313] or Kalman filtering method [310]. In [314], for a three-layer MLP, the hidden-layer weights are updated by BP, while the output-layer weights are updated by solving a set of linear equations. In [315], the hidden-layer weights are updated by a batch-mode Kalman filtering method, while the output-layer weights are updated by solving a set of linear equations. Compared to LM, this method uses an order of magnitude less time at the same accuracy [316].

The parameter-wise algorithm [317] extends the layerwise approach. At each iteration, the weights can be optimized one by one while the other variables are fixed. Compared with the BPM and layerwise algorithms, the parameter-wise algorithm achieves a faster convergence by a factor of more than an order of magnitude.

Lyapunov's stability theory has been applied for weight update [318–320]. A generalization of BP is developed for training MLPs in [318]. Two adaptive versions of BP proposed in [319] converge much faster than BP and EKF. A gradient-descent method without error backpropagation is proposed for MLP learning in [321], and the method has a great potential for concurrency. A linear programming (LP) method is also used for training MLPs [322]. Fuzzy BP (FBP) algorithm [323] outperforms BP considerably in terms of convergence speed and can escape local minima easily, and QuickFBP [324] is a fast version of FBP. Inspired by the chaotic learning process in the brain, chaotic BP integrates the intrinsic chaos of real neurons into BP, leading to a global search ability [325].

The MLP with hard-limiting activation, known as the binary MLP, is typically used for classification. It has the merits of having extremely simple operations, simple internal representations, an easy hardware implementation, and an easy rule extraction. The binary MLP can be trained based on fuzzy logic and error backpropagation [326,327]. In the fuzzy logic implementation, one needs to design a set of fuzzy rules for updating all the weights. For classification, a critical problem in training the binary MLP is to find big linearly separable subsets [328]. Multicore learning (MCL) has been proposed for constructing binary MLPs, based on some lemmas about linear separability [328]. It simplifies the equations of the weights and biases, producing a smaller hidden layer.

10. Fault-Tolerant Learning

In the early 1980s, people believed that MLPs had a built-in ability against node or weight failures. That is, node or weight failures did not affect the performance of a trained MLP much. However, many results [329–334] have shown that if special measures are not considered at the training stage, the network faults could result in a rapid performance degradation. Thus, it is important to have a fault-tolerant MLP. Network faults can be in different forms, e.g., weight noise [329] and open-node fault [335,336].

10.1. Open-Node Fault

In the open-node fault model, some of the hidden nodes are disconnected from the output layer [331]. Put another way, the outputs of those damaged hidden nodes get stuck at zero. Some methods to deal with this fault model have been developed.

Injecting a random node fault [333,337] is a typical heuristic. In this approach, during training, a random node fault is artificially included in the output of hidden nodes.

Zhou et al. [336] proposed the T3 algorithm to train a network. In T3, the break point is first identified from the fault curve, and then the network is trained by injecting a random node fault. The objective of T3 is to maximize the number of faulty nodes that an MLP can afford. However, when there are a large number of hidden nodes, we have to search over a huge space of potential faulty networks, leading to excessive training time.

It is well known that the output of a network node is very sensitive to large weights. Another approach is to limit the weight magnitude. One can add the classical weight-decay regularizer into the objective function to limit the weight magnitude [333], or one can hard-bound the weight magnitude to a small value during training [338,339]. Limiting weight magnitude, however, makes the theoretical analysis of the method very difficult. For example, in [333], the selection guide for the regularization parameter is unavailable even though the fault statistics are available. To optimize the performance of MLPs in a faulty setting, a number of trained networks need to be trained by using different weight-decay parameters. Then, for each of the trained network, a huge number of faulty networks are generated to study the performance in the faulty setting by feeding the testing or training set to those faulty networks. Clearly, this approach is computationally intensive.

Instead of modified the training algorithm, the replication technique aims at modifying the trained network by replicating hidden nodes from a trained network [335,340]. This approach, however, needs to use additional source.

Another idea to protect MLPs is to modify the objective functions. Neti et al. [341] defined the training of a fault-tolerant MLP as a mini-max problem, which minimizes the maximum of the training errors over all potential faulty networks with a single node fault. An objective function was defined in [341]. A mini-max problem is, however, very difficult to solve. Some authors [342,343] also proposed to modify the objective function. In their approach, the objective function consisted of two terms: an MSE of a fault-free MLP and the sum of MSEs of the faulty networks. In [344], a similar objective function was defined and the corresponding learning algorithm was proposed. The method could improve generalization and fault tolerance [344]. The above techniques, which modify the objective function, are just effective for handling a single node fault. For multinode open-fault situation, the computational cost for handling the second term of the objective function is very high. For example, for an MLP with J_2 hidden nodes, for a single node fault, the number of potential faulty networks is equal to J_2. For a multinode fault, the number of potential faulty networks becomes $\sum_{i=1}^{J_2} \binom{M}{i}$. Moreover, in [342–344], the theoretical guideline to set the weighting factor for MSE terms of the faulty networks was not addressed. In [345], a fault-tolerant regularizer was proposed. The performance of this tolerant regularizer was better than that of Zhou et al.'s approach. However, the result was suitable for RBF networks only [345].

10.2. Multiplicative Weight Noise

Multiplicative weight noise [346–350] refers to a zero-mean symmetrical noise with variance proportional to the weight magnitude. This noise comes from the finite-precision representation of the trained weights in an implementation [329].

There are several studies on the behavior of multiplicative weight noise [196,351–353]. The effect of multiplicative weight noise on Adaline and Madaline has been comprehensively analyzed in [351,352]. Choi et al. [196] applied a statistical approach to derive various *output sensitivity measures* for MLPs. For a binary or bipolar input, the sensitivity to weight perturbation can be accurately computed by using the algorithm proposed in [353].

The output sensitivity is closely related to the performance of a faulty network with multiplicative weight noise. Based on the MSE sensitivity, a fault-tolerance model was proposed for RBF networks in [354], where multiplicative noise was injected in all the parameters. By extending Choi's results [196], error sensitivity measures were developed for MLPs in [349]. In [355], sensitivity measures were proposed for split-complex-valued MLPs with noise in inputs and weights.

Various fault-tolerant training methods have been developed for multiplicative weight noise on MLPs. The effect of a multiplicative weight noise is amplified by the magnitude of the associated weight. To reduce its effect, the magnitude of the weights can be kept small. In [339], the magnitude of the most harmful weight, defined by a relevance measure, is shrunk at each step of training. In [338], the magnitude of a weight is upper-limited in a modified BP learning algorithm. In [356], a feasible weight range is determined from the average and variance of the weights' magnitude.

Based on statistics, the effect of multiplicative weight noise can be canceled out when the magnitudes of the input weights of a node are similar. Simon [357] suggested a distributed fault-tolerant training algorithm, in which the training error was minimized subject to an equality constraint on weight magnitudes. Based on the result on MSE sensitivity [354], a fault-tolerant RBF network can be obtained by using a weight-decay regularizer [358]. An explicit regularization method can be used to train a network, such as an MLP [347,349] and an RBF network [350], so that it is able to tolerate multiplicative weight noise. In the explicit regularization method, the objective function is given by

$$\mathcal{J} = \mathcal{J}_{\text{MSTE}} + \lambda \mathcal{J}_{\text{MSES}}, \tag{63}$$

where $\mathcal{J}_{\text{MSTE}}$ stands for the mean squared training error, $\mathcal{J}_{\text{MSES}}$ for the MSE sensitivity, and λ is the regularization parameter. However, the selection rule on λ in the regularization term has not been addressed theoretically.

In [359], the authors presented an objective function, which was similar to (63), for training RBF networks to tolerate multiplicative weight noise. The relationship between λ and the variance of multiplicative weight noise was proposed. A fault-tolerant training algorithm was derived, and the generalization ability of the trained network was also given.

11. Perceptron in the Deep Learning Era

Hinton proposed the deep belief network in 2006, starting the deep learning era [360]. However, the first deep learning model is known as the convolutional neural network model proposed by LeCun in 1989 [361]. Deep learning and reinforcement learning are indispensable factors to achieve human-level or better AI systems. They both have strong connections with the brain functions.

In this deep learning era, the foundations of all the deep learning methods are laid on the ideas from MLP research. Stochastic gradient descent is the cornerstone of machine learning. Deep neural networks have usually millions of connections and are difficult to scale. For deep learning, the traditional second-order learning methods are not viable due to its superlinear complexity. The error backpropagation is commonly used for the training of deep neural networks.

11.1. Solving the Difficulties

Deep convolutional neural networks [361] are the most popular and important deep learning models due to their excellent performance on many well-known image benchmarks such as ImageNet. Deep MLPs were successfully applied to speech recognition [362]. Deep neural networks are obtained by expanding the traditional neural network models into multiple layers.

Deep neural networks are generally trained in two phases [363]: in the first phase, pretraining one layer at a time by unsupervised learning; in the second phase, fine-tuning the entire network by the ultimate criterion. Layerwise pretraining finds the optimal solution for a layer at a time, by fixing all the weights of the subsequent layers. This is a greedy procedure. A deep network with such unsupervised layerwise pretraining almost always outperforms the cases when there is no pretraining phase [364]. This is explained by the fact that the pretraining phase acts as a regularizer [365] for the supervised optimization problem.

A deep convolutional network [366] utilizes many hierarchical layers of tiled convolutional filters to model receptive fields. The LeNet-5 model [361] is a six-layer MLP

model trained with an incremental BP method. It consists of convolutional, pooling, and fully connected layers. Weight sharing and averaging/subsampling are applied, and OBD is employed for the architecture optimization. Other convolutional models are AlexNet, ZFNet, VGGNet, GoogleNet, and ResNet, which have deeper architecture and millions of connections [241].

A deep network can well approximate a target function with high nonlinearity and realize the desired feature representations. However, the high network complexity makes it very difficult to train and easy to overfit. BP or stochastic gradient descent has three difficulties in training a deep network, i.e., overfitting, vanishing gradient, and extreme computational load. Deep convolutional networks have solved all these difficulties. Hinton credited the success to dropout training, ReLU function, the use of GPUs, and techniques for generating more training examples from the existing ones in a talk at NIPS 2012. Dropout is an effective solution to overfitting. It trains some randomly selected nodes at a certain percentage, while the outputs of other nodes are set to be zero to deactivate the nodes. Dynamic regularization of the learning model can also resolve overfitting [367].

A representative solution to the vanishing gradient problem of BP or stochastic gradient descent is the use of the ReLU activation function [368]

$$\sigma(x) = \max(0, x). \tag{64}$$

The ReLU function is a piecewise linear function, which outputs zero for negative input and retains the positive input. The max operation allows the ReLU function to compute much faster than a sigmoidal function. The ReLU function better transmits the error than a sigmoidal function. It also induces a sparsity in the hidden nodes. The ReLU function has a derivative of one for all active nodes, and has a zero gradient whenever a node is inactive. Some alternatives to the ReLU function are LeakyReLU and the parametric deformable exponential linear unit (PDELU) [369]. PDELU pushes the mean value of activation responses closer to zero, which ensures the steepest descent when training a deep neural network.

11.2. Why Deep Learning Always Achieves Good Results

Standard learning theory suggests that such highly expressive networks should heavily overfit [66] and therefore not generalize at all. Occam's razor states that simple hypotheses generalize well. Overfitting is a challenge for deep neural networks when the training data are insufficient.

Surprisingly, deep neural networks typically perform the best, with many more parameters than data points. A solid theoretical foundation is still missing. Some attempts have been made to explain this.

In fact, the popular use of the ReLU activation function $\phi(x) = \max(x, 0)$ is used for alleviating the gradient vanishing problem by eliminating the saturation zone, and thus the error propagation over the network is possible. The use of a ReLU activation also substantially reduces the number of functioning nodes, since as long as the weighted sum is negative, its output will be zero. This substantially decreases the total number of parameters and is an inherent regularization process.

To explain the nonoverfitting problem in the overparameterized case, in [370], instead of using the minimal expected value of the square loss, the generalization error is measured by the maximum loss. The exact number of parameters that ensure both zero training error as well as a good generalization error is estimated. A solution of a regularization problem is guaranteed to yield a good training error as well as a good generalization error and estimate the error on test data. In [371], it is proved that gradient descent can find the global minima of the cross-entropy loss for an overparameterized deep fully connected ReLU network for binary classification, trained with a proper random weight initialization, under certain assumptions on the training data.

In [372], random matrix theory is applied to analyze the weight matrices of deep neural networks. The training process implicitly implements self-regularization, even in

the absence of explicit regularization, such as dropout or weight norm constraints. For backpropagation-trained deep fully connected networks, noise accumulation is generally bound, and adding additional layers does not worsen the signal-to-noise ratio beyond a limit [373]. Noise accumulation can be suppressed entirely when the slope of the activation function is smaller than unity.

The existence of suboptimal local minima and saddle points in the highly nonconvex loss landscape is important for the performance of trained deep neural networks. This is proved in [374] for deep ReLU networks with a squared loss or cross-entropy loss under reasonable assumptions. In [375], experiments validate that sufficiently deep and wide neural networks are not negatively impacted by suboptimal local minima. Suboptimal local minima, even though degenerated to saddle points, exist for fully connected sigmoid networks. The local minima can be escaped from via a nonincreasing path on the loss curve. This provides a partial explanation for the successful application of deep neural networks.

Deep neural networks generalize remarkably well, indicating a strong inductive bias toward functions with a low generalization error. In [376], $P_{SGD}(f|\mathcal{S})$, the probability that an overparameterized deep network trained with stochastic gradient descent converges on a function f consistent with a training set $\mathcal{S}$, is empirically calculated. $P_{SGD}(f|\mathcal{S})$ correlates remarkably well with the Bayesian posterior $P_B(f|\mathcal{S})$, and $P_B(f|\mathcal{S})$ is strongly biased towards low-error and low-complexity functions. Thus, a strong inductive bias in the parameter-function map, which determines $P_B(f|\mathcal{S})$, is the primary explanation for why deep networks generalize so well. $P_B(f|\mathcal{S})$ is the first-order determinant of $P_{SGD}(f|\mathcal{S})$.

11.3. Deep versus Shallow

Although most of the recent works advocate a deep architecture, some researchers support a shallow architecture. We here review a few representative papers on their approximation ability as well as performance comparison.

Compared to the three-layer MLP, the four-layer MLP can usually approximate a target with fewer connections, but may introduce more local minima [64,69,377]. Based on a geometrical interpretation of the MLP, a small four-layer MLP can generate better results for a target function with a flat surface located in its domain [69]. The approximation of smooth multivariable functions with an MLP is treated in [378]. For a specified approximation order, explicit results for the necessary number of hidden nodes and its distributions to the hidden layers are given. It turns out that more than two hidden layers are not necessary when minimizing the number of necessary hidden nodes.

A ReLU network is also proved to be universal approximator [379,380]. Explicit estimates on the size of a ReLU neural network are derived for approximating Lipschitz functions to any given accuracy in the L^∞-norm [380]. In [381], a deep ReLU neural network is constructed that approximates a function with a specified accuracy, and tight dimension-dependent bounds on the computational complexity are given in terms of the size and depth of this network. For deep neural networks with certain smooth functions including sigmoidal functions and ReLU, the approximation error is proved to decay exponentially in the number of nonzero weights [382]. When approximating sufficiently smooth functions, finite-width deep networks require a strictly smaller connectivity than finite-depth wide networks [382]. Deep convolutional neural networks with general convolution, activation, and pooling operators is analyzed in [383]. The translation-invariance of the features in the resulting feature extractor is proved to become progressively more prominent as the network depth increases.

The capacity of layered, fully connected architectures of linear threshold neurons is investigated in [384]. In general, under the same other conditions, shallow networks compute more functions than deep networks, and the functions computed by deep networks are more regular. According to an abstract theorem in function approximation [385], without the requirement of robust parameter selection, deep networks using a nonsmooth activation function such as ReLU, do not provide any significant advantage over shallow networks in terms of the degree of approximation alone.

Bounds on the approximation error with respect to the size of the tanh neural network are derived in [386]. Tanh neural networks with only two hidden layers suffice to approximate functions at comparable or better rates than much deeper ReLU neural networks [386].

Training deep neural networks with error backpropagation is implausible from a biological perspective. In [387], classification is implemented in a three-layer network with a single readout layer by using biologically plausible, local learning rules. The hidden layer weights are either fixed or trained with unsupervised, local learning rules, and the readout layer is trained with a supervised, local learning rule. Networks with localized receptive fields perform significantly better than fully connected networks and can reach the backpropagation performance on MNIST. A three-layer spiking model with leaky integrate-and-fire neurons and spike-timing-dependent plasticity for training the readout layer achieves >98.2% test accuracy on MNIST, which is close to the performance of three-layer rate networks trained with backpropagation. The performance of the shallow spiking network models is comparable to most biologically plausible models of deep learning.

12. Discussion and Conclusions

Before we conclude this paper, we give an example to demonstrate the performance of some popular methods described in this paper.

12.1. An Example—Iris Classification

The iris database from the UCI Repository of machine learning databases was used as a benchmark for evaluating MLP learning algorithms. The iris database contains 150 patterns, which belongs to three classes. Each pattern has four numeric properties. We divided the dataset into a training set (80%) and a testing set (20%). The performance goal was set as 0.001, and the maximum number of epochs was 1000. Eight popular learning algorithms (RProp, BFGS, one-step secant, LM, scaled CG, CG with Powell–Beale restarts, Fletcher–Powell CG, and Polak–Ribiere CG algorithms) were implemented and compared.

A three-layer (4-4-3) MLP network was selected. When training the network, if an example belonged to class i, the ith output node outputted 1, while all the other output nodes outputted -1. The logistic sigmoidal function was used in the hidden layer, and the linear function was used in the output layer. When evaluating the network, the output node with the largest output was set to 1, while the outputs at all the other nodes were set -1. Table 1 lists the training results for 50 independent runs. Figure 4 shows the learning curves for a random run. In this example, BFGS and LM generated the best MSE performance: they used less time for convergence, but as second-order algorithms, they used more memory. All of the algorithms generated good MSE and classification performances. The simulations were performed on a PC with Intel Core i7 CPU 10700 @2.90 GHz and 16 GB RAM based on the MATLAB Neural Networks Toolbox.

Table 1. A 4-4-3 MLP trained with eight algorithms: performance obtained by averaging 50 random runs. RP—Rprop, SCG—scaled CG, CGB—CG with Powell–Beale restarts, CGF—Fletcher–Powell CG, and CGP—Polak–Ribiere CG.

Algorithm	Mean Epochs	Training MSE	Classifier Accuracy (%)	Std	Mean Training Time (s)	Std (s)
RP	990.94	0.025	96.00	0.034	0.7372	0.2262
LM	238.54	0.007	100.00	0.000	0.3186	0.3494
BFGS	154.72	0.015	93.33	0.000	0.2619	0.1778
OSS	999.94	0.027	96.53	0.034	1.1087	0.0555
SCG	903.16	0.016	95.40	0.037	0.7462	0.1720
CGB	439.48	0.028	95.27	0.064	0.5456	0.3567
CGF	562.64	0.021	95.87	0.038	0.6820	0.3278
CGP	573.38	0.021	96.27	0.033	0.6962	0.3565

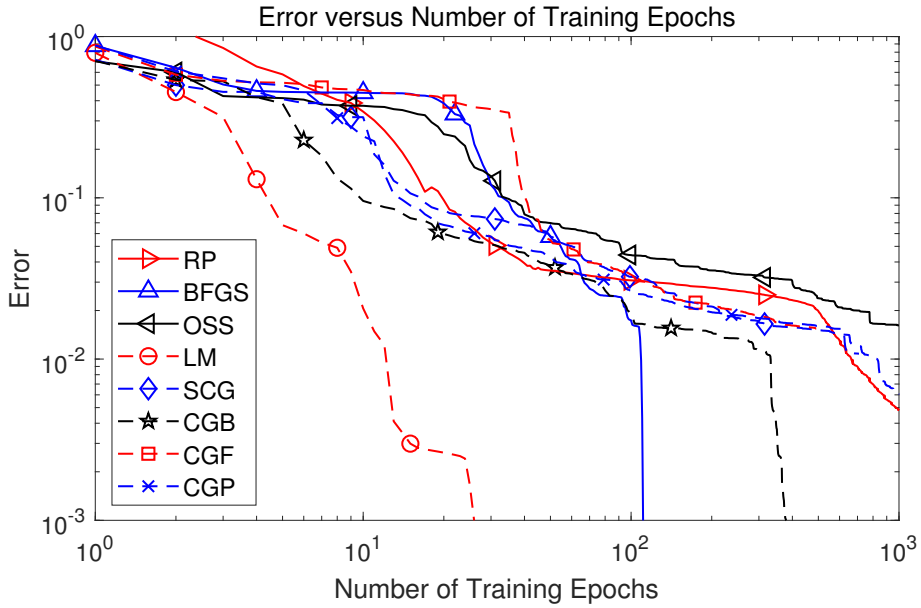

Figure 4. A 4-4-3 MLP trained with eight methods: the evolution curves of the training error for a random run.

12.2. Discussion on the Popular Learning Algorithms

The above iris classification example compared the performance of eight popular MLP learning algorithms. The first-order algorithms, namely, BP and BP with momentum, whether in batch or in online mode, were not compared here, since they are usually slow in convergence, get easily trapped at bad local minima, and it is difficult to select the learning rate and/or the momentum factor. RProp, as the best performing first-order algorithm, was used here for comparison.

Among the eight methods, RProp is a first-order method, LM is a Newton method, BFGS and OSS are quasi-Newton methods, and the other four are CG methods. It is seen that all the algorithms generated good performance. The second-order methods LM and BGFS gave the best performance in terms of convergence speed and total time, but at a cost of higher computational and memory complexities at each epoch. OSS is a memoryless BFGS, and it trades accuracy for lower complexity. The CG algorithms had a performance below that of LM and BFGS but were better than OSS. RProp had a performance comparable to that of several CG algorithms, and this was in agreement with the conclusion given in [240]. The computational complexity of LM is $O(N_w^3)$ and its memory complexity is $O(NJ_M + N_w^2)$. The computational and memory complexities for BFGS are, respectively, $O(N_w^2)$ and $\frac{1}{2} + O(N_w)$. The computational complexity for OSS is $O(N_w)$ and it does not need to store the Hessian. The CG methods are closely related to the quasi-Newton method but conduct a series of line searches along noninterfering directions. They have a memory complexity of $O(N_w)$, which is about four times that of BP. It is well known that BP has linear time and memory complexities. Compared to BP, the computational complexity of CG is significantly increased for each weight update cycle due to the line searches for an appropriate step size. Like BP, CG is also easily trapped at a bad local minimum. CG converges as fast as RLS, but has a lower complexity.

From the complexity and convergence analysis, we suggest that LM be used for medium sized networks, when the number of weights is less than one thousand. BFGS can be used in networks of up to 5000 weights. The CG algorithms can be applied to even larger networks. OSS and RProp are effective heuristic-based methods, and they are not recommended for large networks, since their performance is not theoretically justified. However, for deep learning, with hundreds of thousands or even millions of parameters to adapt, the first-order gradient descent method is the only choice, due to its low computational and memory complexities as well as its proved convergence.

12.3. Summary

The perceptron model was invented by Rosenblatt in 1958, but it is dated to 1943 when McCulloch and Pitts proposed the neuron model. The perceptron model is the most fundamental and also most used neural network model. There have been numerous research publications during the past seven decades. The number of publications on the fundamental theory of the perceptron model and its learning is several thousands, while that on the applications of the perceptron model to various fields is hundreds of thousands. The perceptron model as well as the entire neural network discipline has become a branch of applied mathematics.

Although the perceptron model is so important for the neural network discipline, there are very few comprehensive survey papers in the literature. This paper was motivated to fill the gap. In this paper, we provided a comprehensive, yet state-of-the-art review on the perceptron model, with importance attached to the MLP model and its learning. More specifically, we treated some of the most important topics that are associated with the perceptron model, namely, SLP and its learning, various learning techniques for MLP, including the first-order BP and accelerating techniques for BP, second-order methods, the optimization of network architectures, and fault-tolerant learning. The role of MLP in the deep learning era was also dwelled on, and some important topics on deep learning were also described.

This paper was also meant to serve as a tutorial on the perceptron model. Therefore, we provided details of the equations for the important algorithms that the readers can easily master. We provided an illustrative iris classification example that compared eight most popular MLP learning algorithms, on the basis of which we compared the algorithms and gave our advice on how to select an MLP learning algorithm for a specific application.

Although the perceptron model has become a fundamental method in this deep learning era, it is still a live model. Most popular deep learning models are based on the perceptron model or have a perceptron model as their component layers. We introduced how the difficulties of deep learning were addressed by the research accomplishments of the perceptron model. In recent years, for big data science, many research efforts on first-order learning such as stochastic gradient descent and stochastic coordinate descent, and even on stochastic second-order algorithms, have been conducted on the perceptron model, see Section 4.6. These algorithms are particularly useful for training deep neural networks.

Many deep learning models have been proposed so far. These models are mostly pre-trained by using numerous datasets especially ImageNet, and many other image datasets. Most of the models are also based on convolutional neural network models, which use convolution as the major feature extraction operation. Convolution is an effective operation for physical signals and images. The trained models are then used in numerous specific applications by transfer learning. However, transfer learning is based on analogical reasoning. It is only reasonable for applications with different but related tasks. Thus, transferring the learned deep neural network models to non-image-based classification is theoretically unfounded. Theoretical investigations for transfer learning across different domains are needed for the future of deep learning.

While convolutional-neural-network-based deep learning models may be very effective for classification of images, image-based classification is only an important portion of pattern recognition. In most situations where classification and nonlinear regression problems are under consideration [10,11], the inputs are not physical signals and cannot be organized in a rectangular form, and the use of convolutional-neural-network-based deep learning models for handling those problems is not theoretically justified. For example, in DDoS attack detection problems, the commonly used input features are source IP address, prefix of source IP, flow duration, flow number per time interval, packet size, packet number per flow, unique IP address per time interval, etc. These features are not spatial data and are not in an array form. For a major portion of classification problems, the MLP model or MLP-based deep learning models are preferred to deep convolutional neural networks.

Due to the limitation on the length, many topics that are closely associated with perceptron could not be treated in this paper. For example, various application fields of the perceptron model were not discussed in this paper.

Some other topics related to perceptron are mentioned below. A Sigma-Pi network [6] is a generalization of an MLP by using product nodes and summation nodes to build higher-order terms. It is also a universal approximator [67], and provides inherently higher mapping capabilities than first-order models such as MLPs. It can be trained by the BP learning rule. However, for sigma-Pi network, the numbers of product terms and weights have a combinatorial increase.

The perceptron model has been extended to the complex domain. The complex perceptron learning algorithm [388], whose weights take complex values, has a better separating power than the perceptron learning. The convergence of split complex BP [389] for training complex-valued neural networks has been proved. Complex-valued gradient descent algorithm with an adaptive complex-valued step size has been proposed for the training of complex-valued neural networks [390].

In addition to the analysis-based methods described so far, there are numerous MLP learning methods based on metaheuristics [391]. Those methods were not described here. Hardware and parallel algorithm implementations were also not described in this paper.

Neural network models are traditionally treated as black-box models for modeling unknown systems. The knowledge for human experts cannot be easily integrated into the learning process and human cannot easily understand the function of a trained network. There are some attempts to extract rules from trained three-layer MLPs [231]. One can refer to [241] for a recent progress in perceptron and machine learning.

Author Contributions: Conceptualization, K.-L.D., C.-S.L., W.H.M. and M.N.S.S.; methodology, K.-L.D., C.-S.L., W.H.M. and M.N.S.S.; software, K.-L.D.; writing-original draft preparation, K.-L.D. and C.-S.L.; writing-review and editing, W.H.M. and M.N.S.S.; supervision, M.N.S.S.; project administration, W.H.M. and M.N.S.S.; funding acquisition, W.H.M. and M.N.S.S. All the authors contributed equally. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the General Research Fund of the Hong Kong Research Grants Council under project no. 16214422.

Data Availability Statement: All data included in the main text.

Acknowledgments: We are grateful to the anonymous referees for their valuable comments, which have helped to considerably improve the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Mathm. Biophys.* **1943**, *5*, 115–133. [CrossRef]
2. Rosenblatt, R. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386–408. [CrossRef] [PubMed]
3. Rosenblatt, R. *Principles of Neurodynamics*; Spartan Books: New York, NY, USA, 1962.
4. Widrow, B.; Hoff, M.E. Adaptive switching circuits. In *IRE Eastern Electronic Show and Convention (WESCON) Record, Part 4*; IRE: New York, NY, USA, 1960; pp. 96–104.
5. Minsky, M.L.; Papert, S. *Perceptrons*; MIT Press: Cambridge, MA, USA, 1969.
6. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*; MIT Press: Cambridge, MA, USA, 1986; Volume 1, pp. 318–368.
7. Werbos, P.J. Beyond Regressions: New tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. Thesis, Harvard University, Cambridge, MA, USA, 1974.
8. Bourlard, H.; Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. *Biol. Cybern.* **1988**, *59*, 291–294. [CrossRef]
9. Kramer, M.A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE J.* **1991**, *37*, 233–243. [CrossRef]
10. Wang, M.; Lu, Y.; Qin, J. A dynamic MLP-based DDoS attack detection method using feature selection and feedback. *Comput. Secur.* **2020**, *88*, 101645. [CrossRef]

11. Orru, P.F.; Zoccheddu, A.; Sassu, L.; Mattia, C.; Cozza, R.; Arena, S. Machine learning approach using MLP and SVM algorithms for the fault prediction of a centrifugal pump in the oil and gas industry. *Sustainability* **2020**, *12*, 4776. [CrossRef]
12. Liu, G. Data collection in MI-assisted wireless powered underground sensor networks: Directions, recent advances, and challenges. *IEEE Commun. Mag.* **2021**, *59*, 132–138. [CrossRef]
13. Zhang, K.; Wang, Z.; Chen, G.; Zhang, L.; Yang, Y.; Yao, C.; Wang, J.; Yao, J. Training effective deep reinforcement learning agents for real-time life-cycle production optimization. *J. Pet. Sci. Eng.* **2022**, *208*, 109766. [CrossRef]
14. Lu, S.; Ban, Y.; Zhang, X.; Yang, B.; Liu, S.; Yin, L.; Zheng, W. Adaptive control of time delay teleoperation system with uncertain dynamics. *Front. Neurorobot.* **2022**, *16*, 928863. [CrossRef]
15. Qin, X.; Liu, Z.; Liu, Y.; Liu, S.; Yang, B.; Yin, L.; Liu, M.; Zheng, W. User OCEAN personality model construction method using a BP neural network. *Electronics* **2022**, *11*, 3022. [CrossRef]
16. Dempster, A.P.; Laird, N.M.; Rubin, D.B. Maximum likelihood from incomplete data via the em algorithm. *J. R. Stat. Soc. B* **1977**, *39*, 1–38.
17. Amari, S.I. Natural gradient works efficiently in learning. *Neural Comput.* **1998**, *10*, 251–276. [CrossRef]
18. Cover, T.M. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. Electron. Comput.* **1965**, *14*, 326–334. [CrossRef]
19. Pao, Y.H.; Takefuji, Y. Functional-link net computing: Theory, system architecture, and functionalities. *IEEE Comput.* **1992**, *25*, 76–79. [CrossRef]
20. Volper, D.; Hampson, S.E. Quadratic function nodes: Use, structure and training. *Neural Netw.* **1990**, *3*, 93–107. [CrossRef]
21. Boser, B.E.; Guyon, I.M.; Vapnik, V.N. A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, PA, USA, 27–29 July 1992; pp. 144–152.
22. Hassoun, M.H. *Fundamentals of Artificial Neural Networks*; MIT Press: Cambridge, MA, USA, 1995.
23. Eitzinger, C.; Plach, H. A new approach to perceptron training. *IEEE Trans. Neural Netw.* **2003**, *14*, 216–221. [CrossRef]
24. Gallant, S.I. Perceptron-based learning algorithms. *IEEE Trans. Neural Netw.* **1990**, *1*, 179–191. [CrossRef]
25. Frean, M. A thermal perceptron learning rule. *Neural Comput.* **1992**, *4*, 946–957. [CrossRef]
26. Muselli, M. On convergence properties of pocket algorithm. *IEEE Trans. Neural Netw.* **1997**, *8*, 623–629. [CrossRef]
27. Kohonen, T. Correlation matrix memories. *IEEE Trans. Comput.* **1972**, *21*, 353–359. [CrossRef]
28. Kohonen, T. *Self-Organization and Associative Memory*, 3rd ed.; Springer: New York, NY, USA, 1989.
29. Anderson, J.A. Simple neural network generating an interactive memory. *Math. Biosci.* **1972**, *14*, 197–220. [CrossRef]
30. Widrow, B.; Lehr, M.A. 30 years of adaptive neural networks: Perceptron, Madaline, and backpropagation. *Proc. IEEE* **1990**, *78*, 1415–1445. [CrossRef]
31. Widrow, B.; Stearns, S.D. *Adaptive Signal Processing*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1985.
32. Wang, Z.Q.; Manry, M.T.; Schiano, J.L. LMS learning algorithms: Misconceptions and new results on convergence. *IEEE Trans. Neural Netw.* **2000**, *11*, 47–57. [CrossRef] [PubMed]
33. Luo, Z.Q. On the convergence of the LMS algorithm with adaptive learning rate for linear feedforward networks. *Neural Comput.* **1991**, *3*, 226–245. [CrossRef]
34. Bouboulis, P.; Theodoridis, S. Extension of Wirtinger's calculus to reproducing kernel Hilbert spaces and the complex kernel LMS. *IEEE Trans. Signal Process.* **2011**, *59*, 964–978. [CrossRef]
35. Mays, C.H. Adaptive Threshold Logic. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 1963.
36. Ho, Y.C.; Kashyap, R.L. An algorithm for linear inequalities and its applications. *IEEE Trans. Electron. Comput.* **1965**, *14*, 683–688. [CrossRef]
37. Duda, R.O.; Hart, P.E. *Pattern Classification and Scene Analysis*; Wiley: New York, NY, USA, 1973.
38. Hassoun, M.H.; Song, J. Adaptive Ho-Kashyap rules for perceptron training. *IEEE Trans. Neural Netw.* **1992**, *3*, 51–61. [CrossRef]
39. Khardon, R.; Wachman, G. Noise tolerant variants of the perceptron algorithm. *J. Mach. Learn. Res.* **2007**, *8*, 227–248.
40. Freund, Y.; Schapire, R. Large margin classification using the perceptron algorithm. *Mach. Learn.* **1999**, *37*, 277–296. [CrossRef]
41. Krauth, W.; Mezard, M. Learning algorithms with optimal stability in neural networks. *J. Phys. A* **1987**, *20*, 745–752. [CrossRef]
42. Panagiotakopoulos, C.; Tsampouka, P. The Margitron: A generalized perceptron with margin. *IEEE Trans. Neural Netw.* **2011**, *22*, 395–407. [CrossRef] [PubMed]
43. Vallet, F. The Hebb rule for learning linearly separable Boolean functions: Learning and generalisation. *Europhys. Lett.* **1989**, *8*, 747–751. [CrossRef]
44. Bolle, D.; Shim, G.M. Nonlinear Hebbian training of the perceptron. *Network* **1995**, *6*, 619–633. [CrossRef]
45. Mansfield, A.J. *Training Perceptrons by Linear Programming*; NPL Report DITC 181/91; National Physical Laboratory: Teddington, UK, 1991.
46. Perantonis, S.J.; Virvilis, V. Efficient perceptron learning using constrained steepest descent. *Neural Netw.* **2000**, *13*, 351–364. [CrossRef]
47. Keller, J.M.; Hunt, D.J. Incorporating fuzzy membership functions into the perceptron algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.* **1985**, *7*, 693–699. [CrossRef]
48. Chen, J.L.; Chang, J.Y. Fuzzy perceptron neural networks for classifiers with numerical data and linguistic rules as inputs. *IEEE Trans. Fuzzy Syst.* **2000**, *8*, 730–745.
49. Nagaraja, G.; Krishna, G. An algorithm for the solution of linear inequalities. *IEEE Trans. Comput.* **1974**, *23*, 421–427. [CrossRef]

50. Nagaraja, G.; Chandra Bose, R.P.J. Adaptive conjugate gradient algorithm for perceptron training. *Neurocomputing* **2006**, *69*, 368–386. [CrossRef]
51. Diene, O.; Bhaya, A. Perceptron training algorithms designed using discrete-time control Liapunov functions. *Neurocomputing* **2009**, *72*, 3131–3137. [CrossRef]
52. Cavallanti, G.; Cesa-Bianchi, N.; Gentile, C. Tracking the best hyperplane with a simple budget perceptron. *Mach. Learn.* **2007**, *69*, 143–167. [CrossRef]
53. Fontenla-Romero, O.; Guijarro-Berdinas, B.; Perez-Sanchez, B.; Alonso-Betanzos, A. A new convex objective function for the supervised learning of single-layer neural networks. *Pattern Recognit.* **2010**, *43*, 1984–1992. [CrossRef]
54. Legenstein, R.; Maass, W. On the classification capability of sign-constrained perceptrons. *Neural Comput.* **2008**, *20*, 288–309. [CrossRef] [PubMed]
55. Ho, C.Y.-F.; Ling, B.W.-K.; Lam, H.-K.; Nasir, M.H.U. Global convergence and limit cycle behavior of weights of perceptron. *IEEE Trans. Neural Netw.* **2008**, *19*, 938–947. [CrossRef] [PubMed]
56. Ho, C.Y.-F.; Ling, B.W.-K.; Iu, H.H.-C. Invariant set of weight of perceptron trained by perceptron training algorithm. *IEEE Trans. Syst. Man Cybern. Part B* **2010**, *40*, 1521–1530. [CrossRef]
57. Auer, P.; Burgsteiner, H.; Maass, W. A learning rule for very simple universal approximators consisting of a single layer of perceptrons. *Neural Netw.* **2008**, *21*, 786–795. [CrossRef]
58. Fernandez-Delgado, M.; Ribeiro, J.; Cernadas, E.; Ameneiro, S.B. Direct parallel perceptrons (DPPs): Fast analytical calculation of the parallel perceptrons weights with margin control for classification tasks. *IEEE Trans. Neural Netw.* **2011**, *22*, 1837–1848. [CrossRef]
59. Widrow, B. Generalization and information storage in networks of Adaline neurons. In *Self-Organizing Systems 1962*; Jacbi, M.Y.G., Goldstein, G., Eds.; Spartan Books: Washington, DC, USA, 1962; pp. 435–461.
60. Hoff, M.E. Learning Phenomena in Networks of Adaptive Switching Circuits. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 1962.
61. Widrow, B.; Winter, R.G.; Baxter, R. Learning phenomena in layered neural networks. In Proceedings of the 1st IEEE International Conference Neural Networks, San Diego, CA, USA, 21–24 June 1987; Volume 2, pp. 411–429.
62. Kolmogorov, A.N. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Akad. Nauk. USSR* **1957**, *114*, 953–956.
63. Cybenko, G. *Continuous Valued Neural Networks with Two Hidden Layers Are Sufficient*; Technical Report; Dept of Computer Science, Tufts University: Medford, MA, USA, 1988.
64. Tamura, S.; Tateishi, M. Capabilities of a four-layered feedforward neural network: Four layers versus three. *IEEE Trans. Neural Netw.* **1997**, *8*, 251–255. [CrossRef]
65. Huang, G.B. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Trans. Neural Netw.* **2003**, *14*, 274–291. [CrossRef]
66. Cybenko, G. Approximation by superposition of a sigmoid function. *Math. Control Signals Syst.* **1989**, *2*, 303–314. [CrossRef]
67. Hornik, K.M.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
68. Funahashi, K. On the approximate realization of continuous mappings by neural networks. *Neural Netw.* **1989**, *2*, 183–192. [CrossRef]
69. Xiang, C.; Ding, S.Q.; Lee, T.H. Geometrical interpretation and architecture selection of MLP. *IEEE Trans. Neural Netw.* **2005**, *16*, 84–96. [CrossRef] [PubMed]
70. Llanas, B.; Lantaron, S.; Sainz, F.J. Constructive approximation of discontinuous functions by neural networks. *Neural Process. Lett.* **2008**, *27*, 209–226. [CrossRef]
71. Zhang, X.M.; Chen, Y.Q.; N, N.A.; Shi, Y.Q. Mini-max initialization for function approximation. *Neurocomputing* **2004**, *57*, 389–409. [CrossRef]
72. Werbos, P.J. Backpropagation through time: What it does and how to do it. *Proc. IEEE* **1990**, *78*, 1550–1560. [CrossRef]
73. Du, K.-L.; Swamy, M.N.S. *Neural Networks in a Softcomputing Framework*; Springer: London, UK, 2006.
74. Finnoff, W. Diffusion approximations for the constant learning rate backpropagation algorithm and resistance to locol minima. *Neural Comput.* **1994**, *6*, 285–295. [CrossRef]
75. Fine, T.L.; Mukherjee, S. Parameter convergence and learning curves for neural networks. *Neural Comput.* **1999**, *11*, 747–769. [CrossRef]
76. Oh, S.H. Improving the error backpropagation algorithm with a modified error function. *IEEE Trans. Neural Netw.* **1997**, *8*, 799–803.
77. Wu, W.; Feng, G.; Li, Z.; Xu, Y. Deterministic convergence of an online gradient method for BP neural networks. *IEEE Trans. Neural Netw.* **2005**, *16*, 533–540. [CrossRef]
78. Battiti, R. First- and second-order methods for learning: Between steepest sescent and newton's method. *Neural Netw.* **1992**, *4*, 141–166. [CrossRef]
79. Gori, M.; Maggini, M. Optimal convergence of on-line backpropagation. *IEEE Trans. Neural Netw.* **1996**, *7*, 251–254. [CrossRef]
80. Wu, W.; Xu, Y.S. Deterministic convergence of an on-line gradient method for neural networks. *J. Computat. Appl. Math.* **2002**, *144*, 335–347. [CrossRef]

81. Cochocki, A.; Unbehauen, R. *Neural Networks for Optimization and Signal Processing*; John Wiley & Sons, Inc.: New York, NY, USA, 1993.
82. Wilson, D.R.; Martinez, T.R. The general inefficiency of batch training for gradient descent learning. *Neural Netw.* **2003**, *16*, 1429–1451. [CrossRef] [PubMed]
83. Xu, Z.-B.; Zhang, R.; Jing, W.-F. When does online BP training converge? *IEEE Trans. Neural Netw.* **2009**, *20*, 1529–1539.
84. Zhang, R.; Xu, Z.-B.; Huang, G.-B.; Wang, D. Global convergence of online BP training with dynamic learning rate. *IEEE Trans. Neural Netw. Learn. Syst.* **2012**, *23*, 330–341. [CrossRef]
85. Granziol, D.; Zohren, S.; Roberts, S. Learning rates as a function of batch size: A random matrix theory approach to neural network training. *J. Mach. Learn. Res.* **2022**, *23*, 1–65.
86. Wang, J.; Yang, J.; Wu, W. Convergence of cyclic and almost-cyclic learning with momentum for feedforward neural networks. *IEEE Trans. Neural Netw.* **2011**, *22*, 1297–1306. [CrossRef]
87. Yuan, K.; Ying, B.; Sayed, A.H. On the influence of momentum acceleration on online learning. *J. Mach. Learn. Res.* **2016**, *17*, 1–66.
88. Zhang, N. A study on the optimal double parameters for steepest descent with momentum. *Neural Comput.* **2015**, *27*, 982–1004. [CrossRef] [PubMed]
89. Roux, R.N.; Schmidt, M.; Bach, F. A stochastic gradient method with an exponential convergence rate for finite training sets. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 2663–2671.
90. Johnson, R.; Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 315–323.
91. Defazio, A.; Bach, F.; Lacoste-Julien, S. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. *Adv. Neural Inf. Process. Syst.* **2014**, *27*, 1646–1654
92. Shalev-Shwartz, S.; Zhang, T. Stochastic dual coordinate ascent methods for regularized loss. *J. Mach. Learn. Res.* **2013**, *14*, 567–599.
93. Mokhtari, A.; Ribeiro, A. Stochastic Quasi-Newton Methods. *Proc. IEEE* **2020**, *108*, 1906–1922. [CrossRef]
94. Moody, J. Note on generalization, regularization, and architecture selection in nonlinear learning systems. In *First IEEE-SP Workshop on Neural Networks for Signal Processing*; Morgan Kaufmann: San Mateo, CA, USA, 1991; pp. 1–10.
95. Moody, J.; Hanson, S.J.; Lippmann, R.P. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. *Adv. Neural Inf. Process. Syst.* **1992**, *4*, 847–854.
96. Geman, S.; Bienenstock, E.; Doursat, R. Neural networks and the bias/variance dilemma. *Neural Comput.* **1992**, *4*, 1–58. [CrossRef]
97. Niyogi, P.; Girosi, F. Generalization bounds for function approximation from scattered noisy dat. *Adv. Comput. Math.* **1999**, *10*, 51–80. [CrossRef]
98. Niyogi, P.; Girosi, F. *On the Relationship between Generalization Error, Hypothesis Complexity, and Sample Complexity for Radial Basis Functions*; Tech. Rep.; MIT: Cambridge, MA, USA, 1994.
99. Barron, A.R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inf. Theory* **1993**, *39*, 930–945. [CrossRef]
100. Prechelt, L. Automatic early stopping using cross validation: Quantifying the criteria. *Neural Netw.* **1998**, *11*, 761–767. [CrossRef]
101. Amari, S.; Murata, N.; Muller, K.R.; Finke, M.; Yang, H. Statistical theory of overtraining–is cross-validation asymptotically effective. In *Advances in Neural Information Processing Systems 8*; Morgan Kaufmann: San Mateo, CA, USA, 1996; pp. 176–182.
102. Wu, L.; Moody, J. A smoothing regularizer for feedforward and recurrent neural networks. *Neural Comput.* **1996**, *8*, 461–489. [CrossRef]
103. Orr, M.J. Regularization in the selection of radial basis function centers. *Neural Comput.* **1995**, *7*, 606–623. [CrossRef]
104. Bishop, C.M. *Neural Networks for Pattern Recognition*; Oxford University Press: Oxford, UK, 1995.
105. Guo, P. Studies of Model Selection and Regularization for Generalization in Neural Networks with Applications. Ph.D. Thesis, The Chinese University of Hong Kong, Hong Kong, China, 2002.
106. Krogh, A.; Hertz, J.A. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*; Morgan Kaufmann: San Mateo, CA, USA, 1992; pp. 950–957.
107. Mackay, D.J.C. A practical bayesian framework for backpropagation networks. *Neural Comput.* **1992**, *4*, 448–472. [CrossRef]
108. Reed, R.; Marks, R.J.; Oh, S. Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter. *IEEE Trans. Neural Netw.* **1995**, *6*, 529–538. [CrossRef] [PubMed]
109. Bishop, C.M. Training with noise is equivalent to Tikhonov regularization. *Neural Comput.* **1995**, *7*, 108–116. [CrossRef]
110. Hinton, G.E.; Camp, D.V. Keeping neural networks simple by minimizing the description length of the weights. In Proceedings of the 6th Annual ACM Conference on Computational Learning Theory, Santa Cruz, CA, USA, 26–28 July 1993; pp. 5–13.
111. Nowlan, S.J.; Hinton, G.E. Simplifying neural networks by soft weight-sharing. *Neural Comput.* **1992**, *4*, 473–493. [CrossRef]
112. Tarres, P.; Yao, Y. Online learning as stochastic approximation of regularization paths: Optimality and almost-sure convergence. *IEEE Trans. Inf. Theory* **2014**, *60*, 5716–5735. [CrossRef]
113. Lin, J.; Rosasco, L. Optimal rates for multi-pass stochastic gradient methods. *J. Mach. Learn. Res.* **2017**, *18*, 1–47.
114. Janssen, P.; Stoica, P.; Soderstrom, T.; Eykhoff, P. Model structure selection for multivariable systems by cross-validation. *Int. J. Control* **1998**, *47*, 1737–1758. [CrossRef]
115. Wang, C.; Venkatesh, S.; Stephen, J. Optimal stopping and effective machine complexity in learning. In *Advances in Neural Information Processing Systems 6*; Morgan Kaufmann: San Mateo, CA, USA, 1994; pp. 303–310.

116. Sugiyama, M.; Ogawa, H. Optimal design of regularization term and regularization parameter by subspace information criterion. *Neural Netw.* **2002**, *15*, 349–361. [CrossRef]
117. Sugiyama, M.; Müller, K.-R. The subspace information criterion for infinite dimensional hypothesis spaces. *J. Mach. Learn. Res.* **2003**, *3*, 323–359.
118. Onoda, T. Neural network information criterion for the optimal number of hidden units. In Proceedings of the IEEE International Conference on Neural Networks, ICNN'95, Perth, WA, Australia, 27 November–1 December 1995; pp. 275–280.
119. Murata, N.; Yoshizawa, S.; Amari, S. Network information criterion–determining the number of hidden units for an artificial neural network model. *IEEE Trans. Neural Netw.* **1994**, *5*, 865–872. [CrossRef]
120. Vapnik, V.N. *The Nature of Statistical Learning Theory*; Springer: Berlin/Heidelberg, Germany, 1995.
121. Cherkassky, V.; Shao, X.; Mulier, F.M.; Vapnik, V.N. Model complexity control for regression using vc generalization bounds. *IEEE Trans. Neural Netw.* **1999**, *10*, 1075–1089. [CrossRef] [PubMed]
122. Wada, Y.; Kawato, M. Estimation of generalization capability by combination of new information criterion and cross validation. *IEICE Trans.* **1991**, *2*, 955–965.
123. Akaike, H. A new look at the statistical model identification. *IEEE Trans. Autom. Control* **1974**, *19*, 716–723. [CrossRef]
124. Schwarz, G. Estimating the dimension of a model. *Ann. Stat.* **1978**, *6*, 461–464. [CrossRef]
125. Rissanen, J. Modeling by shortest data description. *Automatica* **1978**, *14*, 465–477. [CrossRef]
126. Rissanen, J. Hypothesis selection and testing by the mdl principle. *Computer* **1999**, *42*, 260–269. [CrossRef]
127. Gallinari, P.; Cibas, T. Practical complexity control in multilayer perceptrons. *Signal Process.* **1999**, *74*, 29–46. [CrossRef]
128. Chen, S. Local regularization assisted orthogonal least squares regression. *Neurocomputing* **2006**, *69*, 559–585. [CrossRef]
129. Chen, S.; Hong, X.; Harris, C.J.; Sharkey, P.M. Sparse modelling using orthogonal forward regression with press statistic and regularization. *IEEE Trans. Syst. Man Cybern. Part B* **2004**, *34*, 898–911. [CrossRef]
130. Reed, R. Pruning algorithms—A survey. *IEEE Trans. Neural Netw.* **1993**, *4*, 40–747. [CrossRef] [PubMed]
131. Chandrasekaran, H.; Chen, H.H.; Manry, M.T. Pruning of basis functions in nonlinear approximators. *Neurocomputing* **2000**, *34*, 29–53. [CrossRef]
132. Mozer, M.C.; Smolensky, P. Using relevance to reduce network size automatically. *Connect. Sci.* **1989**, *1*, 3–16. [CrossRef]
133. Karnin, E.D. A simple procedure for pruning back-propagation trained neural networks. *IEEE Trans. Neural Netw.* **1990**, *1*, 239–242. [CrossRef] [PubMed]
134. Goh, Y.S.; Tan, E.C. Pruning neural networks during training by backpropagation. In Proceedings of the IEEE Region 10's Ninth Annual Int Conf (TENCON'94), Singapore, 22–26 August 1994; pp. 805–808.
135. Ponnapalli, P.V.S.; Ho, K.C.; Thomson, M. A formal selection and pruning algorithm for feedforward artificial neural network optimization. *IEEE Trans. Neural Netw.* **1999**, *10*, 964–968. [CrossRef]
136. Tartaglione, E.; Bragagnolo, A.; Fiandrotti, A.; Grangetto, M. LOss-Based SensiTivity rEgulaRization: Towards deep sparse neural networks. *Neural Netw.* **2022**, *146*, 230–237. [CrossRef]
137. Cho, H.; Jang, J.; Lee, C.; Yang, S. Efficient architecture for deep neural networks with heterogeneous sensitivity. *Neural Netw.* **2021**, *134*, 95–106. [CrossRef]
138. Jiang, X.; Chen, M.; Manry, M.T.; Dawson, M.S.; Fung, A.K. Analysis and optimization of neural networks for remote sensing. *Remote Sens. Rev.* **1994**, *9*, 97–144. [CrossRef]
139. Kanjilal, P.P.; Banerjee, D.N. On the application of orthogonal transformation for the design and analysis of feedforward networks. *IEEE Trans. Neural Netw.* **1995**, *6*, 1061–1070. [CrossRef]
140. Teoh, E.J.; Tan, K.C.; Xiang, C. Estimating the number of hidden neurons in a feedforward network using the singular value decomposition. *IEEE Trans. Neural Netw.* **2006**, *17*, 1623–1629. [CrossRef] [PubMed]
141. Levin, A.U.; Leen, T.K.; Moody, J. Fast pruning using principal components. In *Advances in Neural Information Processing Systems 6*; Morgan Kaufmann: San Mateo, CA, USA, 1994; pp. 847–854.
142. Xing, H.-J.; Hu, B.-G. Two-phase construction of multilayer perceptrons using information theory. *IEEE Trans. Neural Netw.* **2009**, *20*, 715–721. [CrossRef] [PubMed]
143. Sietsma, J.; Dow, R.J.F. Creating artificial neural networks that generalize. *Neural Netw.* **1991**, *4*, 67–79. [CrossRef]
144. Castellano, G.; Fanelli, A.M.; Pelillo, M. An iterative pruning algorithm for feedforward neural networks. *IEEE Trans. Neural Netw.* **1997**, *8*, 519–531. [CrossRef]
145. Cun, Y.L.; Denker, J.S.; Solla, S.A. Optimal brain damage. In *Advances in Neural Information Processing Systems*; Morgan Kaufmann: San Mateo, CA, USA, 1990; pp. 598–605.
146. Hassibi, B.; Stork, D.G.; Wolff, G.J. Optimal brain surgeon and general network pruning. In Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, USA, 23–26 March 1992; pp. 293–299.
147. Soulie, T.C.T.F.F.; Gallinari, P.; Raudys, S. Variable selection with neural networks. *Neurocomputing* **1996**, *12*, 223–248.
148. Stahlberger, A.; Riedmiller, M. Fast network pruning and feature extraction using the unit-obs algorithm. In *Advances in Neural Information Processing Systems 9*; Morgan Kaufmann: San Mateo, CA, USA, 1997; pp. 655–661.
149. Tresp, V.; Neuneier, R.; Zimmermann, H.G. Early brain damage. In *Advances in Neural Information Processing Systems 9*; Morgan Kaufmann: San Mateo, CA, USA, 1997; pp. 669–675
150. Engelbrecht, A.P. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Trans. Neural Netw.* **2001**, *12*, 1386–1399. [CrossRef]

151. Bishop, C.M. Exact calculation of the hessian matrix for the multilayer perceptron. *Neural Comput.* **1992**, *4*, 494–501. [CrossRef]
152. Leung, C.S.; Wong, K.W.; Sum, P.F.; Chan, L.W. A pruning method for the recursive least squared algorithm. *Neural Netw.* **2001**, *14*, 147–174. [CrossRef] [PubMed]
153. Sum, J.; Chan, L.W.; Leung, C.S.; Young, G. Extended kalman filter-based pruning method for recurrent neural networks. *Neural Comput.* **1998**, *10*, 1481–1505. [CrossRef]
154. Sum, J.; Leung, C.S.; Young, G.; Kan, W.K. On the kalman filtering method in neural network training and pruning. *IEEE Trans. Neural Netw.* **1999**, *10*, 161–166. [CrossRef]
155. Hinton, G.E. Connectionist learning procedure. *Artif. Intell.* **1989**, *40*, 185–234. [CrossRef]
156. Weigend, A.S.; Rumelhart, D.E.; Huberman, B.A. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems 3*; Morgan Kaufmann: San Mateo, CA, USA, 1991; pp. 875–882.
157. Ishikawa, M. Learning of modular structured networks. *Artif. Intell* **1995**, *7*, 51–62. [CrossRef]
158. A, A.G.; Lam, S.H. Weight decay backpropagation for noisy data. *Neural Netw.* **1998**, *11*, 1127–1137.
159. Aires, F.; Schmitt, M.; Chedin, A.; Scott, N. The weight smoothing regularization of mlp for jacobian stabilization. *IEEE Trans. Neural Netw.* **1999**, *10*, 1502–1510. [CrossRef] [PubMed]
160. Drucker, H.; Cun, Y.L. Improving generalization performance using double backpropagation. *IEEE Trans. Neural Netw.* **1992**, *3*, 991–997. [CrossRef]
161. Poggio, T.; Girosi, F. Networks for approximation and learning. *Proc. IEEE* **1990**, *78*, C1481–C1497. [CrossRef]
162. Moody, J.; Rognvaldsson, T. Smoothness regularizers for projective basis function networks. *Adv. Neural Inf. Process. Syst.* **1997**, *4*, 585–591.
163. Wang, J.; Xu, C.; Yang, X.; Zurada, J.M. A novel pruning algorithm for smoothing feedforward neural networks based on group lasso method. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 2012–2024. [CrossRef]
164. Ma, R.; Miao, J.; Niu, L.; Zhang, P. Transformed ℓ_1 regularization for learning sparse deep neural networks. *Neural Netw.* **2019**, *119*, 286–298. [CrossRef]
165. Fabisch, A.; Kassahun, Y.; Wohrle, H.; Kirchner, F. Learning in compressed space. *Neural Netw.* **2013**, *42*, 83–93. [CrossRef]
166. Hirose, Y.; Yamashita, K.; Hijiya, S. Back-propagation algorithm which varies the number of hidden units. *Neural New.* **1991**, *4*, 61–66. [CrossRef]
167. Fahlman, S.E.; Lebiere, C. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*; Morgan Kaufmann: San Mateo, CA, USA, 1990; pp. 524–532.
168. Fahlman, S.E. Faster-learning variations on back-propagation: An empirical study. In Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann: San Francisco, CA, USA, 1989; pp. 38–51.
169. Kwok, T.Y.; Yeung, D.Y. Objective functions for training new hidden units in constructive neural networks. *IEEE Trans. Neural Netw.* **1997**, *8*, 1131–1148. [CrossRef]
170. Lehtokangas, M. Modelling with constructive backpropagation. *Neural Netw.* **1999**, *12*, 707–716. [CrossRef] [PubMed]
171. Phatak, D.S.; Koren, I. Connectivity and performance tradeoffs in the cascade correlation learning architecture. *IEEE Trans. Neural Netw.* **1994**, *5*, 930–935. [CrossRef] [PubMed]
172. Setiono, R.; Hui, L.C.K. Use of quasi-newton method in a feed-forward neural network construction algorithm. *IEEE Trans. Neural Netw.* **1995**, *6*, 273–277. [CrossRef]
173. Moody, J.O.; Antsaklis, P.J. The dependence identification neural network construction algorithm. *IEEE Trans. Neural Netw.* **1996**, *7*, 3–13. [CrossRef]
174. Rathbun, T.F.; Rogers, S.K.; DeSimio, M.P.; Oxley, M.E. MLP iterative construction algorithm. *Neurocomputing* **1997**, *17*, 195–216. [CrossRef]
175. Liu, D.; Chang, T.S.; Zhang, Y. A constructive algorithm for feedforward neural networks with incremental training. *IEEE Trans. Circuits Syst.–I* **2002**, *49*, 1876–1879.
176. Fukuoka, Y.; Matsuki, H.; Minamitani, H.; Ishida, A. A modified back-propagation method to avoid false local minima. *Neural Netw.* **1998**, *11*, 1059–1072. [CrossRef]
177. Rigler, A.K.; Irvine, J.M.; Vogl, T.P. Rescaling of variables in back propagation learning. *Neural Netw.* **1991**, *4*, 225–229. [CrossRef]
178. Satoh, S.; Nakano, R. Fast and stable learning utilizing singular regions of multilayer perceptron. *Neural Process. Lett.* **2013**, *38*, 99–115. [CrossRef]
179. Mezard, M.; Nadal, J.P.S. Learning in feedforward layered networks: The tiling algorithm. *J. Phys.* **1989**, *A22*, 2191–2203. [CrossRef]
180. Frean, M. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Comput.* **1990**, *2*, 198–209. [CrossRef]
181. Lee, Y.; Oh, S.H.; Kim, M.W. The effect of initial weights on premature saturation in back-propagation training. In Proceedings of the IEEE International Joint Conf Neural Networks, Seattle, WA, USA, 8–12 July 1991; pp. 765–770.
182. Vitela, J.E.; Reifman, J. Premature saturation in backpropagation networks: Mechanism and necessary condition. *Neural Netw.* **1997**, *10*, 721–735. [CrossRef] [PubMed]
183. Lee, H.M.; Chen, C.M.; Huang, T.C. Learning efficiency improvement of back-propagation algorithm by error saturation prevention method. *Neurocomputing* **2001**, *41*, 125–143. [CrossRef]

184. Ng, S.C.; Leung, S.H.; Luk, A. Fast and global convergent weight evolution algorithm based on modified back-propagation. In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; pp. 3004–3008.
185. Wang, X.G.; Tang, Z.; Tamura, H.; Ishii, M. A modified error function for the backpropagation algorithm. *Neurocomput* **2004**, *57*, 477–484. [CrossRef]
186. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
187. Cun, Y.L.; Simard, P.Y.; Pearlmutter, B. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In *Advances in Neural Information Processing Systems 5*; Morgan Kaufmann: San Mateo, CA, USA, 1993; pp. 156–163.
188. Darken, C.; Moody, J. Towards faster stochastic gradient search. In *Advances in Neural Information Processing Systems 4*; Morgan Kaufmann: San Mateo, CA, USA, 1992; pp. 1009–1016.
189. Vogl, T.P.; Mangis, J.K.; Rigler, A.K.; Zink, W.T.; Alkon, D.L. Accelerating the convergence of the backpropagation method. *Biol. Cybern.* **1988**, *59*, 257–263. [CrossRef]
190. Battiti, R. Accelerated backpropagation learning: Two optimization methods. *Complex Syst.* **1989**, *3*, 331–342.
191. Parlos, A.G.; Femandez, B.; Atiya, A.F.; Muthusami, J.; Tsai, W.K. An accelerated learning algorithm for multilayer perceptron networks. *IEEE Trans. Neural Netw.* **1994**, *5*, 493–497. [CrossRef]
192. Yam, Y.F.; Chow, T.W.S. Extended backpropagation algorithm. *Electron. Lett.* **1993**, *29*, 1701–1702. [CrossRef]
193. Silva, F.M.; Almeida, L.B. Speeding up backpropagation. In *Advanced Neural Computers*; Eckmiller, R., Ed.; North-Holland: Amsterdam, The Netherlands, 1990; pp. 151–158.
194. Magoulas, G.D.; Vrahatis, M.N.; Androulakis, G.S. Effective backpropagation training with variable stepsize. *Neural Netw.* **1997**, *10*, 69–82. [CrossRef] [PubMed]
195. Jacobs, R.A. Increased rates of convergence through learning rate adaptation. *Neural Netw.* **1988**, *1*, 295–307. [CrossRef]
196. Choi, J.Y.; Choi, C.H. Sensitivity of multilayer perceptrons with differentiable activation functions. *IEEE Trans. Neural Netw.* **1992**, *3*, 101–107. [CrossRef] [PubMed]
197. Tesauro, G.; Janssens, B. Scaling relationships in back-propagation learning. *Complex Syst.* **1988**, *2*, 39–44.
198. Tollenaere, T. Supersab: Fast adaptive backpropation with good scaling properties. *Neural Netw.* **1990**, *3*, 561–573. [CrossRef]
199. Martens, J.P.; Weymaere, N. An equalized error backpropagation algorithm for the on-line training of multilayer perceptrons. *IEEE Trans. Neural Netw.* **2002**, *13*, 532–541. [CrossRef]
200. Magoulas, G.D.; Vrahatis, M.N.; Plagianakos, V.P. Globally convergent algorithms with local learning rates. *IEEE Trans. Neural Netw.* **2002**, *13*, 774–779. [CrossRef]
201. Cun, Y.L.; Kanter, I.; Solla, S.A. Second order properties of error surfaces: Learning time and generalization. In *Advances in Neural Information Processing Systems 3*; Morgan Kaufmann: San Mateo, CA, USA, 1991; pp. 918–924.
202. Minai, A.A.; Williams, R.D. Backpropagation heuristics: A study of the extended delta-bar-delta algorithm. In Proceedings of the IEEE International Conference on Neural Networks, San Diego, CA, USA, 17–21 June 1990; pp. 595–600.
203. Yu, X.H.; Chen, G.A.; Cheng, S.X. Dynamic learning rate optimization of the backpropagation algorithm. *IEEE Trans. Neural Netw.* **1995**, *6*, 669–677. [PubMed]
204. Yu, X.H.; Chen, G.A. Efficient backpropagation learning using optimal learning rate and momentum. *Neural Netw.* **1997**, *10*, 517–527. [CrossRef]
205. Veitch, A.C.; Holmes, G. A modified quickprop algorithm. *Neural Comput.* **1991**, *3*, 310–311. [CrossRef] [PubMed]
206. Kolen, J.F.; Pollack, J.B. Backpropagation is sensitive to initial conditions. *Complex Syst.* **1990**, *4*, 269–280.
207. Drago, G.; Ridella, S. Statistically controlled activation weight initialization. *IEEE Trans. Neural Netw.* **1992**, *3*, 627–631. [CrossRef] [PubMed]
208. Thimm, G.; Fiesler, E. High-order and multilayer perceptron initialization. *IEEE Trans. Neural Netw.* **1997**, *8*, 349–359. [CrossRef] [PubMed]
209. Wessels, L.F.A.; Barnard, E. Avoiding false local minima by proper initialization of connections. *IEEE Trans. Neural Netw.* **1992**, *3*, 899–905. [CrossRef]
210. McLoone, S.; Brown, M.D.; Irwin, G.; Lightbody, G. A hybrid linear/nonlinear training algorithm for feedforward neural networks. *IEEE Trans. Neural Netw.* **1998**, *9*, 669–684. [CrossRef] [PubMed]
211. Yam, Y.F.; Chow, T.W.S. Feedforward networks training speed enhancement by optimal initialization of the synaptic coefficients. *IEEE Trans. Neural Netw.* **2001**, *12*, 430–434. [CrossRef]
212. Denoeux, T.; Lengelle, R. Initializing backpropagation networks with prototypes. *Neural Netw.* **1993**, *6*, 351–363. [CrossRef]
213. Smyth, S.G. Designing multilayer perceptrons from nearest neighbor systems. *IEEE Trans. Neural Netw.* **1992**, *3*, 323–333. [CrossRef]
214. Yang, Z.; Zhang, H.; Sudjianto, A.; Zhang, A. An effective SteinGLM initialization scheme for training multi-layer feedforward sigmoidal neural networks. *Neural Netw.* **2021**, *139*, 149–157. [CrossRef]
215. Nguyen, D.; Widrow, B. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In Proceedings of the Internatinal Joint Conference Neural Networks, San Diego, CA, USA, 17–21 June 1990; pp. 21–26.
216. Osowski, S. New approach to selection of initial values of weights in neural function approximation. *Electron. Lett.* **1993**, *29*, 313–315. [CrossRef]

217. Yam, Y.F.; Chow, T.W.S.; Leung, C.T. A new method in determining the initial weights of feedforward neural networks. *Neurocomputing* **1997**, *16*, 23–32. [CrossRef]
218. Yam, Y.F.; Chow, T.W.S. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing* **2000**, *30*, 219–232. [CrossRef]
219. Lehtokangas, M.; Saarinen, P.; Huuhtanen, P.; Kaski, P. Initializing weights of a multilayer perceptron network by using the orthogonal least squares algorithm. *Neural Comput.* **1995**, *7*, 982–999. [CrossRef]
220. Chen, C.L.; Nutter, R.S. Improving the training speed of three-layer feedforward neural nets by optimal estimation of the initial weights. In Proceedings of the International Joint Conference Neural Networks, Seattle, WA, USA, 8–12 July 1991; pp. 2063–2068.
221. Yam, Y.F.; Leung, C.T.; Tam, P.K.S.; Siu, W.C. An independent component analysis based weight initialization method for multilayer perceptrons. *Neurocomputing* **2002**, *48*, 807–818. [CrossRef]
222. Chumachenko, K.; Iosifidis, A.; Gabbouj, M. Feedforward neural networks initialization based on discriminant learning. *Neural Netw.* **2022**, *146*, 220–229. [CrossRef] [PubMed]
223. Lehtokangas, M.; Korpisaari, P.; Kaski, K. Maximum covariance method for weight initialization of multilayer perceptron networks. In Proceedings of the European symp Artificial Neural Netw (ESANN'96), Bruges, Belgium, 24–26 April 1996; pp. 243–248.
224. Costa, P.; Larzabal, P. Initialization of supervised training for parametric estimation. *Neural Process. Lett.* **1999**, *9*, 53–61. [CrossRef]
225. Hinton, G.E. *Connectionist Learning Procedures*; Tech. Rep.; Carnegie-Mellon University: Pittsburgh, PA, USA, 1987.
226. Yang, L.; Lu, W. Backpropagation with homotopy. *Neural Comput.* **1993**, *5*, 363–366. [CrossRef]
227. Kruschke, J.K.; Movellan, J.R. Benefits of gain: Speeded learning and minimal layers in back-propagation networks. IEEE Trans. Syst. Man Cybern. **1991**, *21*, 273–280. [CrossRef]
228. Sperduti, A.; Starita, A. Speed up learning and networks optimization with extended back propagation. *Neural Netw.* **1993**, *6*, 365–383. [CrossRef]
229. Chandra, P.; Singh, Y. An activation function adapting training algorithm for sigmoidal feedforward networks. *Neurocomputing* **2004**, *61*, 429–437. [CrossRef]
230. Eom, K.; Jung, K.; Sirisena, H. Performance improvement of backpropagation algorithm by automatic activation function gain tuning using fuzzy logic. *Neurocomputing* **2003**, *50*, 439–460. [CrossRef]
231. Duch, W. Uncertainty of data, fuzzy membership functions, and multilayer perceptrons. *IEEE Trans. Neural Netw.* **2005**, *6*, 1. [CrossRef] [PubMed]
232. Hush, D.R.; Salas, J.M. Improving the learning rate of back-propagation with the gradient reuse algorithm. In Proceedings of the IEEE International Conference Neural orks (ICNN'88), San Diego, CA, USA, 24–27 July 1988; pp. 441–447.
233. Pfister, M.; Rojas, R. Speeding-up backpropagation–a comparison of orthogonal techniques. In Proceedings of the International Joint Conference on Neural Networks, Nagoya, Japan, 25–29 October 1993; pp. 517–523.
234. Kamarthi, S.V.; Pittner, S. Accelerating neural network training using weight extrapolations. *Neural Netw.* **1999**, *12*, 1285–1299. [CrossRef] [PubMed]
235. Zweiri, Y.H.; Whidborne, J.F.; Seneviratne, L.D. *Optimization and Stability of a Three-Term Backpropagation Algorithm*; Technical Report EM-2000-01; Department of Mechanical Engineering, King's College London: London, UK, 2000.
236. Zweiri, Y.H.; Whidborne, J.F.; Seneviratne, L.D. A three-term backpropagation algorithm. *Neurocomputing* **2003**, *50*, 305–318. [CrossRef]
237. Liang, Y.C.; Feng, D.P.; Lee, H.P.; Lim, S.P.; Lee, K.H. Successive approximation training algorithm for feedforward neural networks. *Neurocomputing* **2002**, *42*, 311–322. [CrossRef]
238. Stich, S.U.; Karimireddy, S.P. The error-feedback framework: Better rates for SGD with delayed gradients and compressed updates. *J. Mach. Learn. Res.* **2020**, *21*, 1–36.
239. Riedmiller, M.; Braun, H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, USA, 25–29 October 1993; pp. 586–591.
240. Hannan, J.M.; Bishop, J.M. A comparison of fast training algorithms over two real problems. In Proceedings of the IEE Conference on Artificial Neural Networks, Cambridge, UK, 21–24 July 1997; pp. 1–6.
241. Du, K.-L.; Swamy, M.N.S. *Neural Networks and Statistical Learning*, 2nd ed.; Springer: London, UK, 2019.
242. Saarinen, S.; Bramley, R.; Cybenko, G. Ill conditioning in neural network training problems. *SIAM J. Sci. Comput.* **1993**, *14*, 693–714. [CrossRef]
243. Fletcher, R. *Practical Methods of Optimization*; Wiley: New York, NY, USA, 1991.
244. Battiti, R.; Masulli, F. Bfgs optimization for faster automated supervised learning. In Proceedings of the International Neural Network Conference, Paris, France, 9–13 July 1990; pp. 757–760.
245. Battiti, R.; Masulli, G.; Tecchiolli, G. Learning with first, second, and no derivatives: A case study in high energy physics. *Neurocomputing* **1994**, *6*, 181–206. [CrossRef]
246. Johansson, E.M.; Dowla, F.U.; Goodman, D.M. Backpropagation learning for multilayer feedforward neural networks using the conjugate gradient method. *Int. J. Neural Syst.* **1991**, *2*, 291–301. [CrossRef]
247. van der Smagt, P. Minimisation methods for training feed-forward neural networks. *Neural Netw.* **1994**, *7*, 1–11. [CrossRef]
248. Moller, M.F. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Netw.* **1993**, *6*, 525–533. [CrossRef]
249. Haykin, S.M. *Neural networks: A Comprehensive Foundation*; Prentice Hall: Upper Saddle River, NJ, USA, 1999.

250. Barnard, E. Optimization for training neural nets. *IEEE Trans. Neural Netw.* **1992**, *3*, 232–240. [CrossRef]
251. Wang, Y.J.; Lin, C.T. A second-order learning algorithm for multilayer networks based on block hessian matrix. *Neural Netw.* **1998**, *11*, 1607–1622. [CrossRef] [PubMed]
252. Golub, G.H.; van Loan, C.F. *Matrix Computation*, 2nd ed.; John Hopkins University Press: Baltimore, MD, USA, 1989.
253. More, J.J. The levenberg-marquardt algorithm: Implementation and theory. In *Numerical Analysis, Lecture Notes in Mathematics 630*; Watson, G.A., Ed.; Springer: Berlin, Germany, 1978; pp. 105–116.
254. Hagan, M.T.; Menhaj, M.B. Training feedforward networks with the marquardt algorithm. *IEEE Trans. Neural Netw.* **1994**, *5*, 989–993. [CrossRef] [PubMed]
255. Chen, Y.X.; Wilamowski, B.M. TREAT: A trust-region-based error-aggregated training algorithm for neural networks. In Proceedings of the International Joint Conference Neural Networks, Honolulu, HI, USA, 12–17 May 2002; pp. 1463–1468.
256. Wilamowski, B.M.; Iplikci, S.; Kaynak, O.; Efe, M.O. An algorithm for fast convergence in training neural networks. In Proceedings of the International Joint Conference Neural Networks, Wahington, DC, USA, 15–19 July 2001; pp. 1778–1782.
257. Ngia, L.S.H.; Sjoberg, J. Efficient training of neural nets for nonlinear adaptive filtering using a recursive levenberg-marquardt algorithm. *IEEE Trans. Signal Process.* **2000**, *48*, 1915–1927. [CrossRef]
258. Wilamowski, B.M.; Cotton, N.J.; Kaynak, O.; Dundar, G. Computing gradient vector and Jacobian matrix in arbitrarily connected neural networks. *IEEE Trans. Ind. Electron.* **2008**, *55*, 3784–3790. [CrossRef]
259. Wilamowski, B.M.; Yu, H. Improved computation for Levenberg–Marquardt training. *IEEE Trans. Neural Netw.* **2010**, *21*, 930–937. [CrossRef]
260. Wilamowski, B.M.; Yu, H. Neural network learning without backpropagation. *IEEE Trans. Neural Netw.* **2010**, *21*, 1793–1803. [CrossRef]
261. Fairbank, M.; Alonso, E.; Schraudolph, N. Efficient calculation of the Gauss-Newton approximation of the Hessian matrix in neural networks. *Neural Comput.* **2012**, *24*, 607–610. [CrossRef]
262. Rubio, J.d.J. Stability analysis of the modified Levenberg-Marquardt algorithm for the artificial neural network training. In *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 3510–3524. [CrossRef] [PubMed]
263. Ampazis, N.; Perantonis, S.J. Two highly efficient second-order algorithms for training feedforward networks. *IEEE Trans. Neural Netw.* **2002**, *13*, 1064–1074. [CrossRef] [PubMed]
264. Lee, J. Attractor-based trust-region algorithm for efficient training of multilayer perceptrons. *Electron. Lett.* **2003**, *39*, 727–728. [CrossRef]
265. Lee, J.; Chiang, H.D. Theory of stability regions for a class of nonhyperbolic dynamical systems and its application to constraint satisfaction problems. *IEEE Trans. Circuits Syst.–I* **2002**, *49*, 196–209.
266. RoyChowdhury, P.; Singh, Y.P.; Chansarkar, R.A. Dynamic tunneling technique for efficient training of multilayer perceptrons. *IEEE Trans. Neural Netw.* **1999**, *10*, 48–55. [CrossRef]
267. Ye, H.; Luo, L.; Zhang, Z. Nesterov's acceleration for approximate Newton. *J. Mach. Learn. Res.* **2020**, *21*, 1–37.
268. Beigi, H.S.M. Neural network learning through optimally conditioned quadratically convergent methods requiring no line search. In Proceedings of the IEEE 36th Midwest Symp Circuits Systems, Detroit, MI, USA, 16–18 August 1993; pp. 109–112.
269. Nazareth, J.L. *Differentiable Optimization and Equation Solving*; Springer: New York, NY, USA, 2003.
270. McLoone, S.; Irwin, G. Fast parallel off-line training of multilayer perceptrons. *IEEE Trans. Neural Netw.* **1997**, *8*, 646–653. [CrossRef]
271. Phua, P.K.H.; Ming, D. Parallel nonlinear optimization techniques for training neural networks. *IEEE Trans. Neural Netw.* **2003**, *14*, 1460–1468. [CrossRef]
272. Shanno, D. Conjugate gradient methods with inexact searches. *Math. Oper. Res.* **1978**, *3*, 244–256. [CrossRef]
273. McLoone, S.; Irwin, G. A variable memory quasi-newton training algorithm. *Neural Process. Lett.* **1999**, *9*, 77–89. [CrossRef]
274. McLoone, S.; Asirvadam, V.S.; Irwin, G. A memory optimal bfgs neural network training algorithm. In Proceedings of the 2002 International Joint Conference on Neural Networks, Honolulu, HI, USA, 12–17 May 2002; pp. 513–518.
275. Perantonis, S.J.; Ampazis, N.; Spirou, S. Training feedforward neural networks with the dogleg method and bfgs hessian updates. In Proceedings of the International Joint Conference on Neural Networks, Como, Italy, 24–27 July 2000; pp. 138–143.
276. Bortoletti, A.; Fiore, C.D.; Fanelli, S.; Zellini, P. A new class of quasi-newtonian methods for optimal learning in MLP-networks. *IEEE Trans. Neural Netw.* **2003**, *14*, 263–273. [CrossRef] [PubMed]
277. Hestenes, M.R.; Stiefel, E. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.* **1953**, *49*, 409–436. [CrossRef]
278. Charalambous, C. Conjugate gradient algorithm for efficient training of artificial neural networks. *IEE Proc. G* **1992**, *139*, 301–310. [CrossRef]
279. Dixon, L.C.W. Conjugate gradient algorithms: Quadratic termination properties without linear searches. *J. Inst. Math. Appl.* **1975**, *15*, 9–18. [CrossRef]
280. Goryn, D.; Kaveh, M. Conjugate gradient learning algorithms for multilayer perceptrons. In Proceedings of the IEEE 32nd Midwest Symp Circuits Systems, Champaign, IL, USA, 14–16 August 1989; pp. 736–739.
281. Fletcher, R.; Reeves, C.W. Function minimization by conjugate gradients. *Comput. J.* **1964**, *7*, 148–154.
282. Polak, E. *Computational Methods in Optimization: A Unified Approach*; Academic Press: New York, NY, USA, 1971.

283. Dai, Y.H.; Yuan, Y. A nonlinear conjugate gradient method with a strong global convergence property. *SIAM J. Optim.* **1999**, *10*, 177–182. [CrossRef]
284. A, A.B.; Kaszkurewicz, E. Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method. *Neural Netw.* **2004**, *17*, 65–71.
285. Towsey, M.; Alpsan, D.; Sztriha, L. Training a neural network with conjugate gradient methods. In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australian, 27 November–1 December 1995; pp. 373–378.
286. Liu, C.S.; Tseng, C.H. Quadratic optimization method for multilayer neural networks with local error-backpropagation. *Int. J. Syst. Sci.* **1999**, *30*, 889–898. [CrossRef]
287. Kostopoulos, A.E.; Grapsa, T.N. Self-scaled conjugate gradient training algorithms. *Neurocomputing* **2009**, *72*, 3000–3019. [CrossRef]
288. Ruck, D.W.; Rogers, S.K.; Kabrisky, M.; Maybeck, P.S.; Oxley, M.E. Comparative analysis of backpropagation and the extended kalman filter for training multilayer perceptrons. *IEEE Trans. Pattern. Anal. Mach. Intell.* **1992**, *14*, 686–691. [CrossRef]
289. Iguni, Y.I.; Sakai, H.; Tokumaru, H. A real-time learning algorithm for a multilayered neural network based on the extended kalman filter. *IEEE Trans. Signal Process.* **1992**, *40*, 959–967. [CrossRef]
290. Leung, C.S.; Chan, L.W. Dual extended kalman filtering in recurrent neural networks. *Neural Netw.* **2003**, *16*, 223–239. [CrossRef] [PubMed]
291. Singhal, S.; Wu, L. Training feedforward networks with the extended kalman algorithm. In Proceedings of the IEEE ICASSP-89, Glasgow, Scotland, 23–26 May 1989; pp. 1187–1190.
292. Zhang, Y.; Li, X. A fast u-d factorization-based learning algorithm with applications to nonlinear system modeling and identification. *IEEE Trans. Neural Netw.* **1999**, *10*, 930–938. [CrossRef] [PubMed]
293. Rivals, I.; Personnaz, L. A recursive algorithm based on the extended kalman filter for the training of feedforward neural models. *Neurocomputing* **1998**, *20*, 279–294. [CrossRef]
294. Shah, S.; Palmieri, F. Meka–a fast, local algorithm for training feedforward neural networks. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), San Diego, CA, USA, 17–21 June 1990; pp. 41–46.
295. Puskorius, G.V.; Feldkamp, L.A. Decoupled extended kalman filter training of feedforward layered networks. In Proceedings of the International Joint Conference on Neural Networks, Seattle, WA, USA, 8–12 July 1991; pp. 771–777.
296. Nishiyama, K.; Suzuki, K. H_∞-learning of layered neural networks. *IEEE Trans. Neural Netw.* **2001**, *12*, 1265–1277. [CrossRef]
297. Azimi-Sadjadi, R.; Liou, R.J. Fast learning process of multilayer neural networks using recursive least squares method. *IEEE Trans. Signal Process.* **1992**, *40*, 446–450. [CrossRef]
298. Bilski, J.; Rutkowski, L. A fast training algorithm for neural networks. *IEEE Trans. Circuits Syst.–II* **1998**, *45*, 749–753. [CrossRef]
299. Leung, C.S.; Tsoi, A.C.; Chan, L.W. Two regularizers for recursive least square algorithms in feedforward multilayered neural networks. *IEEE Trans. Neural Netw.* **2001**, *12*, 1314–1332. [CrossRef]
300. Xu, Y.; Wong, K.-W.; Leung, C.-S. Generalized RLS approach to the training of neural networks. *IEEE Trans. Neural Netw.* **2006**, *17*, 19–34. [CrossRef]
301. Stan, O.; Kamen, E. A local linearized least squares algorithm for training feedforward neural networks. *IEEE Trans. Neural Netw.* **2000**, *11*, 487–495. [CrossRef] [PubMed]
302. Parisi, R.; Claudio, E.D.D.; Orlandim, G.; Rao, B.D. A generalized learning paradigm exploiting the structure of feedforward neural networks. *IEEE Trans. Neural Netw.* **1996**, *7*, 1450–1460. [CrossRef]
303. Ma, S.; Ji, C.; Farmer, J. An efficient em-based training algorithm for feedforward neural networks. *Neural Netw.* **1997**, *10*, 243–256. [CrossRef] [PubMed]
304. Amari, S.I. Information geometry of the em and em algorithms for neural networks. *Neural Netw.* **1995**, *8*, 1379–1408. [CrossRef]
305. Kosko, B.; Audhkhasi, K.; Osoba, O. Noise can speed backpropagation learning and deep bidirectional pretraining. *Neural Netw.* **2020**, *129*, 359–384 [CrossRef]
306. Adigun, O.; Kosko, B. Noise-boosted bidirectional backpropagation and adversarial learning. *Neural Netw.* **2019**, *120*, 1–204. [CrossRef] [PubMed]
307. Martens, J. New insights and perspectives on the natural gradient method. *J. Mach. Learn. Res.* **2020**, *21*, 1–76.
308. Gonzalez, A.; Dorronsoro, J.R. Natural conjugate gradient training of multilayer perceptrons. *Neurocomputing* **2008**, *71*, 2499–2506. [CrossRef]
309. Baermann, F.; Biegler-Koenig, F. On a class of efficient learning algorithms for neural networks. *Neural Netw.* **1992**, *5*, 139–144. [CrossRef]
310. Scalero, R.S.; Tepedelenlioglu, N. A fast new algorithm for training feedforward neural networks. *IEEE Trans. Signal Process.* **1992**, *40*, 202–210. [CrossRef]
311. Ergezinger, S.; Thomsen, E. An accelerated learning algorithm for multilayer perceptrons: Optimization layer by layer. *IEEE Trans. Neural Netw.* **1995**, *6*, 32–42. [CrossRef]
312. Hunt, S.D.; Deller, J.R., Jr. Selective training of feedforward artificial neural networks using matrix perturbation theory. *Neural Netw.* **1995**, *8*, 931–944. [CrossRef]
313. Rubanov, N.S. The layer-wise method and the backpropagation hybrid approach to learning a feedforward neural network. *IEEE Trans. Neural Netw.* **2000**, *11*, 295–305. [CrossRef] [PubMed]
314. Manry, M.T.; Apollo, S.J.; Allen, L.S.; Lyle, W.D.; Gong, W.; Dawson, M.S.; Fung, A.K. Fast training of neural networks for remote sensing. *Remote Sens. Rev.* **1994**, *9*, 77–96. [CrossRef]

315. Chen, H.H.; Manry, M.T.; Chandrasekaran, H. A neural network training algorithm utilizing multiple sets of linear equations. *Neurocomputing* **1999**, *25*, 55–72. [CrossRef]
316. Yu, C.; Manry, M.T.; Li, J.; Narasimha, P.L. An efficient hidden layer training method for the multilayer perceptron. *Neurocomputing* **2006**, *70*, 525–535. [CrossRef]
317. Li, Y.; Zhang, D.; Wang, K. Parameter by parameter algorithm for multilayer perceptrons. *Neural Process. Lett.* **2006**, *23*, 229–242. [CrossRef]
318. Yu, X.; Efe, M.O.; Kaynak, O. A general backpropagation algorithm for feedforward neural networks learning. *IEEE Trans. Neural Netw.* **2002**, *13*, 251–254. [PubMed]
319. Behera, L.; Kumar, S.; Patnaik, A. On adaptive learning rate that guarantees convergence in feedforward networks. *IEEE Trans. Neural Netw.* **2006**, *17*, 1116–1125. [CrossRef]
320. Man, Z.; Wu, H.R.; Liu, S.; Yu, X. A new adaptive backpropagation algorithm based on Lyapunov stability theory for neural networks. *IEEE Trans. Neural Netw.* **2006**, *17*, 1580–1591.
321. Brouwer, R.K. Training a feed-forward network by feeding gradients forward rather than by back-propagation of errors. *Neurocomputing* **1997**, *16*, 117–126. [CrossRef]
322. Shawe-Taylor, J.S.; Cohen, D.A. Linear programming algorithm for neural networks. *Neural Netw.* **1990**, *3*, 575–582. [CrossRef]
323. Stoeva, S.; Nikov, A. A fuzzy backpropagation algorithm. *Fuzzy Sets Syst.* **2000**, *112*, 27–39. [CrossRef]
324. Nikov, A.; Stoeva, S. Quick fuzzy backpropagation algorithm. *Neural Netw.* **2001**, *14*, 231–244. [CrossRef]
325. Tao, P.; Cheng, J.; Chen, L. Brain-inspired chaotic backpropagation for MLP. *Neural Netw.* **2022**, *155*, 1–13. [CrossRef] [PubMed]
326. Delgado, M.; Mantas, C.J.; Moraga, C. A fuzzy rule based backpropagation method for training binary multilayer perceptron. *Inf. Sci.* **1999**, *113*, 1–17. [CrossRef]
327. Castro, J.L.; Delgado, M.; Mantas, C.J. A fuzzy rule-based algorithm to train perceptrons. *Fuzzy Sets Syst.* **2001**, *118*, 359–367. [CrossRef]
328. Wang, D.; Chaudhari, N.S. Binary neural network training algorithms based on linear sequential learning. *Int. J. Neural Syst.* **2003**, *5*, 333–351. [CrossRef] [PubMed]
329. Burr, J. Digital neural network implementations. In *Neural Networks, Concepts, Applications, and Implementations*; Prentice Hall.: Englewood Cliffs, NJ, USA, 1991; Volume III.
330. Holt, J.; Hwang, J. Finite precision error analysis of neural network hardware implementations. *IEEE Trans. Comput.* **1993**, *42*, 281–290. [CrossRef]
331. Bolt, G.R. Fault models for artifical neural networks. In Proceedings of the IJCNN'91, Singapore, 18–21 November 1991; pp. 1373–1378.
332. Bolt, G.R.; Austin, J.; Morgan, G. *Fault Tolerant Multi-Layer Perceptron Networks*; Tech. Rep. YCS-92-180; Department of Computer Science, University of York: York, UK, 1992.
333. Chiu, C.T.; Mehrotra, K.; Mohan, C.K.; Ranka, S. Modifying training algorithms for improved fault tolerance. In Proceedings of the ICNN'94, Orlando, FL, USA, 28 June 1994–2 July 1994; pp. 333–338.
334. Murray, A.F.; Edwards, P.J. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Trans. Neural Netw.* **1994**, *5*, 792–802. [CrossRef] [PubMed]
335. Phatak, D.S.; Koren, I. Complete and partial fault tolerance of feedforward neural nets. *IEEE Trans. Neural Netw.* **1995**, *6*,446–456. [CrossRef]
336. Zhou, Z.H.; Chen, S.F.; Chen, Z.Q. Improving tolerance of neural networks against multi-node open fault. In Proceedings of the IJCNN'01, Washington, DC, USA, 15–19 July 2001; pp. 1687–1692.
337. Sequin, C.H.; Clay, R.D. Fault tolerance in feedforward artificial neural networks. *Neural Netw.* **1990**, *4*, 111–141.
338. Cavalieri, S.; Mirabella, O. A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks. *Neural Netw.* **1999**, *12*, 91–106. [CrossRef] [PubMed]
339. Hammadi, N.C.; Ito, H. A learning algorithm for fault tolerant feedforward neural networks. *IEICE Trans. Inf. Syst.* **1997**, *80*, 21–26.
340. Emmerson, M.D.; Damper, R.I. Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application. *IEEE Trans. Neural Netw.* **1993**, *4*, 788–793. [CrossRef]
341. Neti, C.; Schneider, M.H.; Young, E.D. Maximally fault tolerance neural networks. *IEEE Trans. Neural Netw.* **1992**, *3*, 14–23. [CrossRef] [PubMed]
342. Simon, D.; El-Sherief, H. Fault-tolerance training for optimal interpolative nets. *IEEE Trans. Neural Netw.* **1995**, *6*, 1531–1535. [CrossRef]
343. Phatak, D.S.; Tchernev, E. Synthesis of fault tolerant neural networks. In Proceedings of the IJCNN'02, Honolulu, HI, USA, 12–17 May 2002; pp. 1475–1480.
344. Zhou, Z.H.; Chen, S.F. Evolving fault-tolerant neural networks. *Neural Comput. Appl.* **2003**, *11*, 156–160. [CrossRef]
345. Leung, C.S.; Sum, J. A fault-tolerant regularizer for rbf networks. *IEEE Trans. Neural Netw.* **2008**, *19*, 493–507. [CrossRef]
346. Bernier, J.L.; Ortega, J.; Rodriguez, M.M.; Rojas, I.; Prieto, A. An accurate measure for multilayer perceptron tolerance to weight deviations. *Neural Process. Lett.* **1999**, *10*, 121–130. [CrossRef]
347. Bernier, J.L.; Ortega, J.; Rojas, I.; Ros, E.; Prieto, A. Obtaining fault tolerance multilayer perceptrons using an explicit regularization. *Neural Process. Lett.* **2000**, *12*, 107–113. [CrossRef]

348. Bernier, J.L.; Ortega, J.; Rojas, I.; Ros, E.; Prieto, A. A quantitative study of fault tolerance, noise immunity and generalization ability of MLPs. *Neural Comput.* **2000**, *12*, 2941–2964. [CrossRef]
349. Bernier, J.L.; Ortega, J.; Rojas, I.; Ros, E.; Prieto, A. Improving the tolerance of multilayer perceptrons by minimizing the statistical sensitivity to weight deviations. *Neurocomputing* **2000**, *31*, 87–103. [CrossRef]
350. Bernier, J.L.; Diaz, A.F.; Fernandez, F.J.; Canas, A.; Gonzalez, J.; Martin-Smith, P.; Ortega, J. Assessing the noise immunity and generalization of radial basis function networks. *Neural Process. Lett.* **2003**, *18*, 35–48. [CrossRef]
351. Stevenson, M.; Winter, R.; Widrow, B. Sensitivity of feedfoward neural networks to weight errors. *IEEE Trans. Neural Netw.* **1990**, *1*, 71–80. [CrossRef]
352. Piche, S.W. The selection of weight accuracies for madalines. *IEEE Trans. Neural Netw.* **1995**, *6*, 432–445. [CrossRef] [PubMed]
353. Zeng, X.; Wang, Y.; Zhang, K. Computation of adalines' sensitivity to weight perturbation. *IEEE Trans. Neural Netw.* **2006**, *17*, 515–519. [CrossRef] [PubMed]
354. Catala, M.A.; Parra, X.L. Fault tolerance parameter model of radial basis function networks. In Proceedings of the IEEE ICNN'96, Washington, DC, USA, 3–6 June 1996; pp. 1384–1389.
355. Yang, S.H.; Ho, C.H.; Siu, S. Sensitivity analysis of the split-complex valued multilayer perceptron due to the errors of the i.i.d. inputs and weights. *IEEE Trans. Neural Netw.* **2007**, *18*, 1280–1293. [CrossRef]
356. Kamiura, N.; Isokawa, T.; Hata, Y.; Matsui, N.; Yamato, K. On a weight limit approach for enhancing fault tolerance of feedforward neural networks. *IEICE Trans. Inf. Syst.* **2000**, *83*, 1931–1939.
357. Simon, D. Distributed fault tolerance in optimal interpolative nets. *IEEE Trans. Neural Netw.* **2001**, *12*, 1348–1357. [CrossRef]
358. Parra, X.; Catala, A. Fault tolerance in the learning algorithm of radial basis function networks. In Proceedings of the IJCNN 2000, Como, Italy, 24–27 July 2000; pp. 527–532.
359. Sum, J.; Leung, C.S.; Ho, K. On objective function, regularizer and prediction error of a learning algorithm for dealing with multiplicative weight noise. *IEEE Trans. Neural Netw.* **2009**, *20*, 124–138. [CrossRef] [PubMed]
360. Hinton, G.E.; Osindero, S.; Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput.* **2006**, *18*, 1527–1554. [CrossRef] [PubMed]
361. Le Cun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*; Touretzky, D.S., Ed.; Morgan Kaufmann: San Mateo, CA, USA, 1989; Volume 2, pp. 396–404.
362. Mohamed, A.; Dahl, G.; Hinton, G. Deep belief networks for phone recognition. In Proceedings of the NIPS Workshop on Deep Learning for Speech Recognition and Related Applications, Whistler, BC, Canada, 12 December 2009.
363. Larochelle, H.; Bengio, Y.; Louradour, J.; Lamblin, P. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.* **2009**, *1*, 1–40.
364. Erhan, D.; Bengio, Y.; Courville, A.; Manzagol, P.-A.; Vincent, P.; Bengio, S. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.* **2010**, *11*, 625–660.
365. Bengio, Y. Learning deep architectures for AI. *Found. Trends Mach. Learn.* **2009**, *2*, 1–127. [CrossRef]
366. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
367. Bejani, M.M.; Ghatee, M. Theory of adaptive SVD regularization for deep neural networks. *Neural Netw.* **2020**, *128*, 33–46. [CrossRef] [PubMed]
368. Nair, V.; Hinton, G.E. Rectified linear units improve restricted Boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML), Haifa, Israel, 21–24 June 2010; pp. 807–814.
369. Cheng, Q.; Li, H.; Wu, Q.; Ma, L.; Ngan, K.N. Parametric deformable exponential linear units for deep neural networks. *Neural Netw.* **2020**, *125*, 281–289. [CrossRef]
370. Mhaskar, H.N.; Poggio, T. An analysis of training and generalization errors in shallow and deep networks. *Neural Netw.* **2020**, *121*, 229–241. [CrossRef]
371. Zou, D.; Cao, Y.; Zhou, D.; Gu, Q. Gradient descent optimizes over-parameterized deep ReLU networks. *Mach. Learn.* **2020**, *109*, 467–492. [CrossRef]
372. Martin, C.H.; Mahoney, M.W. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *J. Mach. Learn. Res.* **2021**, *22*, 1–73.
373. Semenova, N.; Larger, L.; Brunner, D. Understanding and mitigating noise in trained deep neural networks. *Neural Netw.* **2022**, *146*, 151–160. [CrossRef] [PubMed]
374. Liu, B.; Liu, Z.; Zhang, T.; Yuan, T. Non-differentiable saddle points and sub-optimal local minima exist for deep ReLU networks. *Neural Netw.* **2021**, *144*, 75–89. [CrossRef] [PubMed]
375. Petzka, H.; Sminchisescu, C. Non-attracting regions of local minima in deep and wide neural networks. *J. Mach. Learn. Res.* **2021**, *22*, 1–34.
376. Mingard, C.; Valle-Perez, G.; Skalse, J.; Louis, A.A. Is SGD a Bayesian sampler? Well, almost. *J. Mach. Learn. Res.* **2021**, *22*, 1–64.
377. Chester, D.L. Why two hidden layers are better than one. In Proceedings of the International Joint Conference on Neural Networks, Washington, DC, USA, 15–19 January 1990; pp. 265–268.
378. Trenn, S. Multilayer perceptrons: Approximation order and necessary number of hidden units. *IEEE Trans. Neural Netw.* **2008**, *19*, 836–844. [CrossRef]

379. Huang, C. ReLU networks are universal approximators via piecewise linear or constant functions. *Neural Comput.* **2020**, *32*, 2249–2278. [CrossRef]
380. Yarotsky, D. Error bounds for approximations with deep ReLU networks. *Neural Netw.* **2017**, *94*, 103–114. [CrossRef]
381. Dung, D.; Nguyen, V.K. Deep ReLU neural networks in high-dimensional approximation. *Neural Netw.* **2021**, *142*, 619–635. [CrossRef]
382. Elbrachter, D.; Perekrestenko, D.; Grohs, P.; Bolcskei, H. Deep Neural Network Approximation Theory. *IEEE Trans. Inf. Theory* **2021**, *67*, 2581–2623 [CrossRef]
383. Wiatowski, T.; Bolcskei, H. A Mathematical Theory of Deep Convolutional Neural Networks for Feature Extraction. *IEEE Trans. Inf. Theory* **2018**, *64*, 1845–1866. [CrossRef]
384. Baldi, P.; Vershynin, R. The capacity of feedforward neural networks. *Neural Netw.* **2019**, *116*, 288–311. [CrossRef]
385. Mhaskar, H.N. Dimension independent bounds for general shallow networks. *Neural Netw.* **2020**, *123*, 142–152. [CrossRef]
386. Ryck, T.D.; Lanthaler, S.; Mishra, S. On the approximation of functions by tanh neural networks. *Neural Netw.* **2021**, *143*, 732–750. [CrossRef] [PubMed]
387. Illing, B.; Gerstner, W.; Brea, J. Biologically plausible deep learning—But how far can we go with shallow networks? *Neural Netw.* **2019**, *118*, 90–101. [CrossRef] [PubMed]
388. Nemoto, I.; Kubono, M. Complex associative memory. *Neural Netw.* **1996**, *9*, 253–261. [CrossRef]
389. Xu, D.; Zhang, H.; Liu, L. Convergence analysis of three classes of split-complex gradient algorithms for complex-valued recurrent neural networks. *Neural Comput.* **2010**, *22*, 2655–2677. [CrossRef] [PubMed]
390. Zhang, Y.; Huang, H. Adaptive complex-valued stepsize based fast learning of complex-valued neural networks. *Neural Netw.* **2020**, *124*, 233–242. [CrossRef] [PubMed]
391. Du, K.-L.; Swamy, M.N.S. *Search and Optimization by Metaheuristics*; Springer: New York, NY, USA, 2016.