Computational and Applied Mathematics (2023) 42:167
https://doi.org/10.1007/s40314-023-02307-9

Stochastic perturbation of subgradient algorithm for nonconvex deep neural networks

A. El Mouatasim[1] · J. E. Souza de Cursi[2] · R. Ellaia[3]

Received: 19 January 2023 / Revised: 19 January 2023 / Accepted: 15 April 2023 /
Published online: 1 May 2023

Abstract
Choosing a learning rate is a necessary part of any subgradient method optimization. With deeper models such as convolutional neural networks of image classification, fine-tuning the learning rate can quickly become tedious, and it does not always result in optimal convergence. In this work, we suggest a variation of the subgradient method in which the learning rate is updated by a control step in each iteration of each epoch. Stochastic Perturbation Subgradient Algorithm (SPSA) is our approach for tackling image classification issues with deep neural networks including convolutional neural networks. Used MNIST dataset, the numerical results reveal that our SPSA method is faster than Stochastic Gradient Descent and its variants with a fixed learning rate. However SPSA and convolutional neural network model improve the results of image classification including loss and accuracy.

Keywords Subgradient algorithm · Nonconvex nonsmooth optimization · Stochastic perturbation · Learning rate · Image classification · Deep neural networks and CNN

Mathematics Subject Classification 52B55 · 49J52 · 90C26 · 60H30 · 68T05

Communicated by Antonio José Silva Neto.

A. El Mouatasim
a.elmouatasim@uiz.ac.ma

J. E. Souza de Cursi
souza@insa-rouen.fr

R. Ellaia
rachid.ellaia@um5r.emi.ma

1 Mathematical and Management Department, Ibnou Zohr University, FPO, Ouarzazate, Morocco

2 Laboratoire de Mécanique de Normandie, Normandie Université, INSA Rouen Normandie, 685, Avenue de l'Université Saint-Etienne du Rouvray, Rouen, France

3 LERMA, Mohammadia School of Engineers, Mohammed V University in Rabat, Avenue Ibn Sina BP765, Agdal, Rabat, Morocco

1 Introduction

The learning capacities of the human brain, which is made up of neurons connected by synapses, provide the inspiration for Artificial Neural Networks (ANN) (Singh et al. 2015). In fact, they can learn any given mapping up to arbitrary accuracy (Stutz 2014), at least theoretically. They also make it simple to incorporate past task information into the network design. As a consequence, LeCun et al. (1989) proposed Convolutional Neural Networks (CNN) for computer vision including image classification and Natural Language Processing (NLP) including text classification and text generation.
We can find the major real-world applications of image classification in autonomous vehicles (Bojarski et al. 2016), face recognition (Xinhua and Qian 2015), medical diagnosis (Singh et al. 2015), and satellite image (Pelletier et al. 2019).

When discussing the learning of feature hierarchies, we are usually talking about deep learning, that is training deep neural networks (DNN) (Huang et al. 2019). Here, DNN usually refers to neural networks with more than 3 layers. To date, deep learning is still considered challenging (Bengio 2009).

Due to the constrained architecture of CNN,[1] they allow to train deep architectures with traditional methods,[2] in 2020 El Mouatasim improved gradient algorithm (El Mouatasim 2020) via Nesterov step for softmax classifier without using any activation function ReLu (Rectified Linear Units) and activation function tanh, see Sect. 3.

This allows a CNN to learn feature hierarchies, that means that the CNN is used to learn a hierarchy of task-specific features which can then be used for classification using traditional multilayer perceptrons.[3] This property is a huge advantage of CNN over fully-connected multilayer perceptrons where learning deep architectures is considered difficult (Bengio 2009).

However, when using a popular and applicable activation function ReLU (nonsmooth) or tanh (nonconvex) in the architecture of Deep Neural Networks (DNN) model (Konstantin and Johannes 2019), the training of DNN is a hard problem, since it is a nonsmooth nonconvex optimization problem; therefore, the local minimum can not be a global minimum. To find the parameters updated by the Stochastic Gradient Descent (SGD) algorithm (Tuyen and Hang-Tuan 2021) and its variants including Nesterov accelerated gradient (NAG) algorithm (Botev et al. 2017), Adagrad (Duchi et al. 2011) and Adam (Kingma and Ba 2015) are easy to converge into the local minimum of loss function slowly. The theory and numerical results show that the SGD algorithm and its variants need more improvements for nonsmooth global optimization DNN.

Therefore, the learning rate of optimization algorithm can be improved, the parameters update, such as learning rate annulling (Nakamura et al. 2021), cyclical learning rate [35] and (Liu and Liu 2019). We propose in this paper a fast iterative algorithm with a control learning rate called stochastic perturbation of subgradient algorithm (SPSA) for solving a nonsmooth nonconvex DNN including image classification.

We analyze the propose algorithm SPSA and offer some encouraging findings from numerical tests using the MNIST dataset (LeCun and Cortes 2010), demonstrating that the proposed

[1] CNN as introduced in LeCun et al. (1989) make use of weight sharing as introduced in Sect. 4 which reduces the complexity and size of the network and allows to train deep architectures.

[2] Usually this includes gradient descent (Singh et al. 2015; Tuyen and Hang-Tuan 2021) optimization as discussed in Sect. 5 as well as error backpropagation as introduced in Sect. 3 to evaluate the gradient of a chosen loss function.

[3] The multilayer perceptron is discussed in detail in Sect. 3.

classification method SPSA is more accurate and anti-overfitting than previous image classification methods such as SGD and its variants.

Although neural networks can be used to solve computer vision problems, previous knowledge should be incorporated into the network architecture to improve generalization performance (LeCun 1989). The goal of CNN is to exploit spatial information between image pixels. As a result, they rely on discrete convolution. We explore the essential components of CNN in Sect. 3.7, as reported in Jarrett et al. (2009) and LeCun et al. (2010).

This work is organized as follows: after the introduction, Sect. 2 presents notation and assumptions. Section 3 gives a brief view of deep neural network modeling. Section 4: considers the optimization of a DNN, including SPSA. Finally, in Sect. 5, the numerical results are examined, and Sect. 6 presents the conclusions and perspectives.

2 Notations and assumptions

We denote by $\mathbb{R}$ the set of the real numbers $(-\infty, +\infty)$, $\mathbb{R}^+$ the set of positive real numbers $[0, +\infty)$, $\mathbb{E} = \mathbb{R}^n$, the n-dimensional real Euclidean space. For $\mathbf{w} = (w_1, w_2, \dots, w_n)^t \in \mathbb{E}$, $\mathbf{w}^t$ denotes the transpose of $\mathbf{w}$. We denote by $\| \mathbf{w} \| = \sqrt{\mathbf{w}^t\mathbf{w}} = (w_1^2 + \cdots + w_n^2)^{1/2}$ the Euclidean norm of $\mathbf{w}$ and by $(\mathbf{w}, \mathbf{y}) = \mathbf{w}^t\mathbf{y}$ the scalar product on $\mathbb{E}$.
We shall denote $\mathbf{Id}$ the $n \times n$ Identity matrix.
Training a DNN implies the solution of an optimization problem of the kind:

$$Find\ \mathbf{w}^* \in \mathbb{E}\ such\ that\ G(\mathbf{w}^*) = G^* = \min_{\mathbf{w}\in\mathbb{E}} G(\mathbf{w}), \tag{1}$$

$\mathbf{w}^*$ is said to be a *global minimum solution* of of problem (1) and G^* is the corresponding *global minimal value*. In the context of DNN, the *objective function* $G : \mathbb{E} \longrightarrow \mathbb{R}$ is referred as *loss function* and $\mathbf{w}$ is a vector of weights. In general, the weights of a network form a list of vectors $\mathbf{w} = \left(\mathbf{w}^{1)}, \dots, \mathbf{w}^{(K)}\right)$, where K is the number of layers in the DNN (see Sect. 3) and $\mathbf{w}^{(\ell)} \in \mathbb{R}^{\eta_\ell}$, where η_ℓ is the number of units in the layer no. ℓ, $1 \le \ell \le K$. The list of vectors is brought to a vector $\mathbf{w}$ (see Sect. 3).

The problem (1) is unconstrained, since the admissible set is $\mathbb{E} = \mathbb{R}^n$. Typical loss functions are bounded from below on $\mathbb{E}$:

$$\exists m \in \mathbb{R}\ such\ that\ G(\mathbf{w}) \ge m, \forall \mathbf{w} \in \mathbb{E}\,. \tag{2}$$

Such an asumption is verified if, for instance, G continuous and coercive, i.e.,

$$G \text{ is continuous on } \mathbb{E} \text{ and } \lim_{\|\mathbf{w}\|\to+\infty} G(\mathbf{w}) = +\infty.$$

Under these asumptions, there exists a solution $\mathbf{w}^* \in \mathbb{E}$ and $G^* = G(\mathbf{w}^*)$. Let us introduce $S_\alpha = \{\mathbf{w} \in \mathbb{E} \quad G(\mathbf{w}) \le \alpha\}$. We assume also that

$$\forall \alpha > m :\ S_\alpha \text{ is not empty, closed and bounded, meas } (S_\alpha) > 0 \tag{3}$$

where meas (S_α) is the Lebesgue measure of S_α. Notice that these asumptions eliminate the trivial situation where G is constant on $\mathbb{E}$ (in such a situation, Problem (1) is trivial: any point of $\mathbb{E}$ is a global minimum). Equation (3) is verified when G is continuous and coercive.

Finally, we assume that G is uniformly Lipschitz continuous, i.e.,

$$\exists L \in \mathbb{R} \text{ such that } \forall(\mathbf{w}, \mathbf{y}) \in \mathbb{E} \times \mathbb{E} \quad |G(\mathbf{w}) - G(\mathbf{y})| \le L\|\mathbf{w} - \mathbf{y}\|_2,$$

In this work, we consider the determination of these global minima by *subgradient methods* involving *stochastic perturbations*. These methods have shown to be efficient and robust for nonconvex nonsmooth problems (El Mouatasim et al. 2006, 2011).

Subgradient methods are descent methods, id est, methods that generate a sequence $\{\mathbf{w}_{(n)} : n \in \mathbb{N}\}$, from a given initial vector $\mathbf{w}_{(0)} \in \mathbb{E}$, using iterations:

$$\mathbf{w}_{(k+1)} = \mathbf{Q}\left(\mathbf{w}_{(k)}, \mathbf{d}_{(k)}, \rho_{(k)}\right) = \mathbf{w}_{(k)} + \rho_{(k)}\mathbf{d}_{(k)} \ , \ \forall k \geq 0 \,. \tag{4}$$

In the framework of descent methods, $\mathbf{d}_{(k)}$ is the *descent direction* and $\rho_{(k)}$ is the *step*, referred as *learning rate* in the context of DNN. For subgradient methods, the descent direction is given by (notice the sign minus preceding $\mathbf{d}_{(k)}$):

$$-\,\mathbf{d}_{(k)} \in \partial G(\mathbf{w}_{(k)}). \tag{5}$$

where $\partial G(\mathbf{w})$ is the Clarke's subdifferential at point $\mathbf{w} \in \mathbb{E}$, see for instance (Bagirov et al. 2013).
To simplify the analysis, we assume that there is a constant $M \in \mathbb{R}^+$, such that

$$\parallel \mathbf{d}_{(k)} \parallel \leq M. \tag{6}$$

The determination of the step $\rho_{(k)} \geq 0$ involves often one-dimensional search with a previously established maximal step $\rho_{\max}$. For instance, the optimal step is determined by

$$\rho_{(k)} = \arg\min \left\{ f(\rho) = G\left(\mathbf{w}_{(k)} + \rho\mathbf{d}_{(k)}\right) : 0 \leq \rho \leq \rho_{\max} \right\} \tag{7}$$

So, the step is given by a function $\rho : \mathbb{E} \times \mathbb{E} \to \mathbb{R}$ and reads as

$$\rho_{(k)} = \rho\left(\mathbf{w}_{(k)}, \mathbf{d}_{(k)}\right); 0 \leq \rho\left(\mathbf{w}_{(k)}, \mathbf{d}_{(k)}\right) \leq \rho_{\max}. \tag{8}$$

Stochastic perturbations are introduced to prevent from convergence to local minima and the difficulties. They consist of a controlled random search (see, for instance El Mouatasim et al. 2006, 2011, 2014 and El Mouatasim (2018)). In such an approach, $\left\{\mathbf{w}_{(k)} : k \in \mathbb{N}\right\}$ becomes a sequence of random vectors $\left\{\mathbf{W}_{(k)} : k \in \mathbb{N}\right\}$, with $\mathbf{W}_{(0)} = \mathbf{w}_{(0)}$ and the iterations (4) modified as follows (again, notice the sign minus preceding $\mathbf{D}_{(k)}$):

$$\mathbf{W}_{(k+1)} = \mathbf{Q}\left(\mathbf{W}_{(k)}, \mathbf{D}_{(k)}, \rho_{(k)}\right) + \mathbf{P}_{(k)}, \ , \ \ -\mathbf{D}_{(k)} \in \partial G(\mathbf{W}_{(k)}) \,. \tag{9}$$

Here, $\left\{\mathbf{P}_{(k)} : k \in \mathbb{N}^*\right\}$ is a suitable sequence of random vectors—the stochastic perturbations. A convenient choice of $\{\mathbf{P}_{(k)} : k \in \mathbb{N}^*\}$ ensures the convergence of $\mathbf{W}_{(k)}$ to $\mathbf{w}^\star$ almost surely. The step (or learning rate) is given by

$$\rho_{(k)} = \rho\left(\mathbf{W}_{(k)}, \mathbf{D}_{(k)}\right) \ ; \ 0 \leq \rho\left(\mathbf{W}_k, \mathbf{D}_{(k)}\right) \leq \rho_{\max} \,. \tag{10}$$

The practical implementation of (9)–(10) involves finite samples of $\mathbf{P}_{(k)}$ (see Sect. 5).

3 Deep neural network modeling

The goal of this section is to present briefly the mathematical modeling of DNN including the graph of DNN, activation functions, loss function, forward evaluation, backpropagation and regularization. The contents of this section are classical in the literature and our presentation is largely based on Bishop (1995), Stutz (2014) and Haykin (2005).

The basic element of an ANN is a unit usually referred as *neuron*, *perceptron* or *unit*. Indeed, at the origins of ANN, the units were intended to represent a *neuron* and simulate its activity (Marvin 1954). Later, *perceptrons* were designed to simulate the behavior of the eye

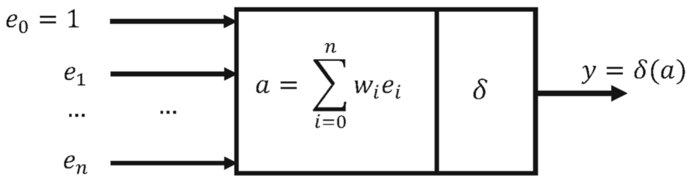


Fig. 1 Basic unit of a neural net. The basic unit transforms the inputs $\mathbf{e} = (e_1, \ldots, e_n)^t$ into a value a by aggregation involving weights $\mathbf{w} = (w_0, w_1, \ldots, w_n)^t$: $a = w_0 + w_1 e_1 + \cdots + w_n e_n$ (w_0 is the bias). Using a dummy input $e_0 = 1$, we have $a = \alpha(\mathbf{e}, \mathbf{w}) = \mathbf{w}^t \mathbf{e}$. The value of a is modified by an activation function δ to produce the output $y = \delta(a)$

(Rosenblatt 1958). Nowadays, the terminology *unit* is often preferred since it is neutral and does not imply analogies with the human organism.

Indeed, the basic element is a processing unity that computes an output y, using given a vector of n input values $\mathbf{e} = (e_1, \ldots, e_n)^t$. Usually, the output is determined as a function of unknown parameters brought to a *row* vector $\mathbf{w} = (w_0, w_1, \ldots, w_n)$:

$$y = \delta(a) \ , \ a = \alpha(\mathbf{e}, \mathbf{w}) \ . \tag{11}$$

The unknown parameters $\mathbf{w}$ must be determined from available observed data given in a dataset $\mathcal{D} = \{(\mathbf{e}_s, y_s), 1 \leq s \leq n_s\}$—observations that form a sample from the $(\mathbf{e}, y)$. The determination of the unknown values $\mathbf{w}$ is called *training*. It is usually carried by dividing $\mathcal{D}$ is divided in three parts: a training set T, a validation set V and a test set S. training usually consists in minimizing the gap between the predictions of the ANN and the observations on T, with verifications on V. After the training, the performance of the ANN is evaluated on S. In the framework of ANN, w_0 is usually referred as *bias*—it can be seen as an external element, independent from the inputs; $(w_1, \ldots, w_n)^t$ are the *weights*; δ is the *activation function* and α is a weighted sum of the input values (Haykin 2005):

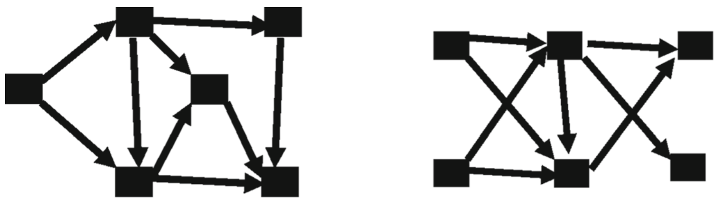
$$\alpha(\mathbf{e}, \mathbf{w}) = w_0 + w_1 e_1 + \cdots + w_n e_n \ . \tag{12}$$

It is usual to introduce a dummy input $e_0 = 1$, so that the bias can be treated as a weight (see Fig. 1):

$$\alpha(\mathbf{e}, \mathbf{w}) = \mathbf{w}^t \mathbf{e} = \sum_{i=0}^{n} w_i e_i \ . \tag{13}$$

To generate an ANN, different units are interconnected, so that the output of a unit becomes one of the inputs of another unit (Bishop 1995). The units are organized according to a directed graph[4]—the network graph—where each unit becomes a node (generally labeled according to its output) and the directed edges represent the information flow in the network: an arrow from a first unit to a second one, indicates that the output of the first unit is used as input for the second unity (see Figs. 2, 3)

[4] A directed graph is an ordered pair $G = (V, E)$, where V is a set of nodes and E is a set of edges linking the nodes in its most general form: Within the graph, $(u, v) \in E$ denotes the presence of a directed edge from node u to node v. Given two units u and v in a network graph, a directed edge from u to v indicates that the output of unit u is used as input by unit v.



(a) An example of network (b) A second example of network

Fig. 2 Using a directed graph to generate an ANN. Each ■ is an unity. Arrows show the flow of information

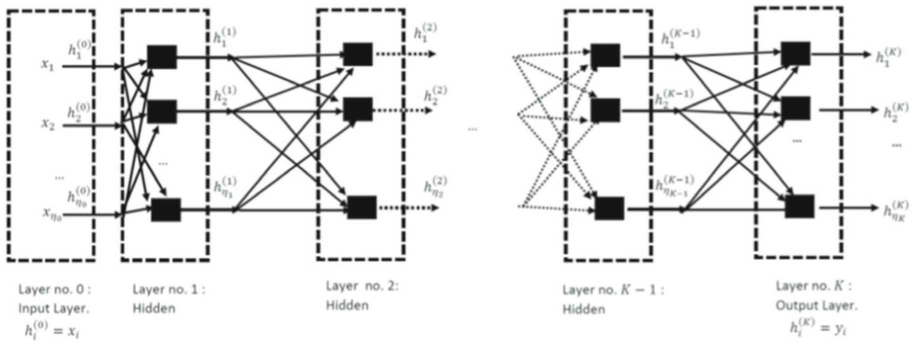


Fig. 3 DNN graph of a K-layer perceptron. The ℓ^{th} hidden layer contains η_ℓ hidden units. As usual in this kind of diagram, the bias is not represented

3.1 Multilayer perceptrons

The units of an ANN are generally organized in layers—i. e., in subsets of units receiving the same inputs and producing different outputs each, which will form the inputs of the next layer. Each subset is visualized as a stack of units (see Fig. 3) and referred by indexes (ℓ, k) indicating their positions (stack ℓ, position k from the top). The first layer receives the input and the last layer furnishes the output of the ANN—these layers are particularly referred as being the *input layer* and the *output layer*. The other intermediating layers are called *hidden layers* and are generally invisible to the user.

When there are more than three hidden layers, we refer to the multilayer perceptron as a Deep Neural Network (DNN). Training a DNN—or Deep Learning—is considered as challenging (Bengio 2009).

Let us consider a multilayer perceptron[5] formed by the input layer (numbered as 0) and K other layers, where K is a strictly positive integer. Layers $1, \dots, K-1$ are hidden ones, layer K is the output layer. Layer number ℓ contains η_ℓ units and produces outputs $h_i^{(\ell)}$, $1 \le i \le \eta_\ell$, as shown in Fig. 3. The input layer performs the identity: $h_i^{(0)} = x_i$. With these notations,the net receives η_0 inputs and generates η_K outputs.

Let us denote, for $\ell \ge 0$, $\mathbf{h}^{(\ell)} = \left(h_1^{(\ell)}, \dots, h_{\eta_\ell}^{(\ell)}\right)^t$. The vector $\mathbf{h}^{(\ell)}$ contains the outputs of the layer ℓ which are the inputs of the layer $\ell + 1$. Analogously, let us introduce, for

[5] A K-layer perceptron, on the other hand, is made up of $(K+1)$ layers, including the input layer. The input layer remains uncounted (or is numbered to zero), since it does not perform processing: the input units compute the identity (Bishop 1995, 2006).

$\ell > 0$, $\mathbf{w_i}^{(\ell)} = (w_{i,0}^{(\ell)}, \dots, w_{i,\ell-1}^{(\ell)})$: $\mathbf{w_i}^{(\ell)}$ is a *row* vector containing the weigths for the unit (ℓ, i)—id est, $w_{i,k}^{(\ell)}$ is the weight corresponding to the oriented edge going from unit $(\ell-1, k)$ to unit (ℓ, i) (from the k^{th} unit in layer $(\ell-1)$ to the i^{th} unit in layer ℓ), and $w_{i,0}^{(\ell)}$ is the bias. We have (see Eq. (12))

$$h_i^{(\ell)} = \delta_i^{(\ell)}(a_i^{(\ell)}) \ , \ a_i^{(\ell)} = \alpha_i^{(\ell)}(\mathbf{h}^{(\ell-1)}, \mathbf{w_i}^{(\ell)}) \ , 1 \le i \le \eta_\ell \ , 1 \le \ell \le K; \tag{14}$$

with (see Eq. (13))

$$\alpha_i^{(\ell)}(\mathbf{h}^{(\ell-1)}, \mathbf{w_i}^{(\ell)}) = \mathbf{w_i}^{(\ell)}\mathbf{h}^{(\ell-1)} \ , 1 \le \ell \le K \ . \tag{15}$$

Let us introduce $\mathbf{w}^{(\ell)} = \left(w_{i,k}^{(\ell)} : 1 \le i \le \eta_\ell, 0 \le k \le \eta_{\ell-1}\right)$—a matrix containg the weights of layer ℓ; $\boldsymbol{\alpha}^{(\ell)}\left(\mathbf{h}^{(\ell-1)}, \mathbf{w}^{(\ell)}\right) = \left(\alpha_1^{(\ell)}\left(\mathbf{h}^{(\ell-1)}, \mathbf{w_1}^{(\ell)}\right), \dots, \alpha_{\eta_\ell}^{(\ell)}\left(\mathbf{h}^{(\ell-1)}, \mathbf{w}_{\eta_\ell}^{(\ell)}\right)\right)$, $\mathbf{a}^{(\ell)} = \left(a_1^{(\ell)}, \dots, a_{\eta_\ell}^{(\ell)}\right)^t$—vectors containing all the agregations of layer ℓ; and $\boldsymbol{\delta}^{(\ell)}(\mathbf{a}) = \left(\delta_1^{(\ell)}(a_1^{(\ell)}), \dots, \delta_{\eta_\ell}^{(\ell)}(a_{\eta_\ell}^{(\ell)})\right)$—a vector containing all the activations of layer ℓ. Then, Eqs. (14)-(15) can be written in a concise form

$$\mathbf{h}^{(\ell)} = \boldsymbol{\delta}^{(\ell)}(\mathbf{a}) \ , \ \mathbf{a}^{(\ell)} = \boldsymbol{\alpha}^{(\ell)}(\mathbf{h}^{(\ell-1)}, \mathbf{w}^{(\ell)}) = \mathbf{w}^{(\ell)}\mathbf{h}^{(\ell-1)} \ 1 \le \ell \le K \ . \tag{16}$$

Let $\mathbf{x} = \left(x_1, \dots, x_{\eta_0}\right)$ be the vector of inputs, and $\mathbf{w} = \left(\mathbf{w}^{(\ell)} : 1 \le \ell \le K\right)$ represent the list of all weights in the network: $\mathbf{w}$ is brought to a vector $\mathbf{w}$:

$$\mathbf{w} = \left(v^{(1)}, \dots, v^{(K)}\right)$$

where

$$v^{(i)} = \left(w_{1,0}^{(i)}, \dots, w_{1,\eta_{i-1}}^{(i)}, \dots, w_{\eta_i,0}^{(i)}, \dots, w_{\eta_i \eta_{i-1}}^{(i)}\right).$$

Then

$$\mathbf{h}^{(K)} = \boldsymbol{\tau}(\mathbf{x}, \mathbf{w}) \ , \tag{17}$$

where

$$\boldsymbol{\tau}(\mathbf{x}, \mathbf{w}) = \boldsymbol{\delta}^{(K)}\left(\boldsymbol{\delta}^{(K-1)}\left(\boldsymbol{\delta}^{(K-2)}\left(\dots, \mathbf{w}^{(K-2)}\right), \mathbf{w}^{(K-1)}\right), \mathbf{w}^{(K)}\right) \ , \tag{18}$$

and the multilayer perceptron can be considered as a function

$$\mathbf{h}(\bullet, \mathbf{w}) : \mathbb{R}^{\eta_0} \to \mathbb{R}^{\eta_K}, \mathbf{x} \mapsto \mathbf{h}(\mathbf{x}, \mathbf{w}), \tag{19}$$

associating to the inputs $\mathbf{x}$ the outputs $\mathbf{h}(\mathbf{x}, \mathbf{w}) = \left(h_1^{(K)}, \dots, h_{\eta_K}^{(K)}\right)$.

From the standpoint of mathematical modeling, the training of a DNN summarizes as

(a) Data: a set of observations $\mathcal{D} = \{(\mathbf{x}_s, \mathbf{y}_s) \ , \ 1 \le s \le n_s\} \subset \mathbb{R}^{\eta_0} \times \mathbb{R}^{\eta_K}$ is given. A loss function $\mathcal{L} : \mathbb{R}^{\eta_k} \times \mathbb{R}^{\eta_K} \mapsto \mathbb{R}^+$ is given to measure the gap between two elements from $\mathbb{R}^{\eta_K}$ (for instance, a distance or a pseudo-distance). Examples of standard loss functions are given in 3.3.

(b) The decision variables are the weights and bias, grouped in $\mathbf{w}$—these are the model parameters to be determined.

(c) The performance of a given set of model parameters $\mathbf{w}$ is evaluated by the aggregation of the values of the gaps between the outputs $\boldsymbol{\tau}(\mathbf{x}_s, \mathbf{w})$ and the observed values $\mathbf{y}_s$, for $1 \le s \le n_s$. The aggregation is performed by a function $\mathcal{A} : \mathbb{R}^{n_s} \mapsto \mathbb{R}^+$, transforming the vector of distances $\mathbf{d}(\mathbf{w}) = \big(\mathcal{L}(\boldsymbol{\tau}(\mathbf{x}_1, \mathbf{w}), \mathbf{y}_1), \ldots, \mathcal{L}(\boldsymbol{\tau}(\mathbf{x}_{n_s}, \mathbf{w}), \mathbf{y}_{n_s})\big)$ into a non-negative real number. The global loss function to be minimized is

$$G\left(\mathbf{w}\right) = \mathcal{A}\left(\mathbf{d}(\mathbf{w})\right) \ . \tag{20}$$

For instance, we can use the arithmetic mean of the gaps:

$$\mathcal{A}\left(\mathbf{d}(\mathbf{w})\right) = \frac{1}{n}\sum_{s=1}^{n_s} d_s(\mathbf{w}) = \frac{1}{n}\sum_{s=1}^{n_s} \mathcal{L}(\boldsymbol{\tau}(\mathbf{x}_s), \mathbf{y}_s) \ . \tag{21}$$

(d) Find the model parameters that minimize the global loss function f, id est, find

$$\mathbb{w}^* = \arg\min\left\{G\left(\mathbf{w}\right) : \mathbf{w}\right\} \ . \tag{22}$$

In the sequel, we use this model for the training of a DNN on a daset of images $\mathbf{X}$ and labels Y: $\mathcal{D} = \{(\mathbf{X}_s, Y_s)\,, i = 1, \ldots n_s\}$.

3.2 Activation functions

Although the activation functions are, in principle, specific to each unit—as indicated in the preceding—it is usual to consider a single activation function for all the hidden layers: $\delta_i^{(\ell)} = \delta$, for $1 \leq i \leq \eta_\ell$ and $1 \leq \ell \leq K - 1$. Often, a distinct activation function is used for the output layer: $\delta_i^{(K)} = \overline{\delta}$, for $1 \leq i \leq \eta_K$, with a choice adapted to the problem under consideration. The choice of δ and $\overline{\delta}$ is considered as important, since it modifies the behavior of the network. Today, the choice for δ is mostly dictated by experience and trial (Feng and Lu 2019; Szandała yyy), while the choice of $\overline{\delta}$ results from the characteristics of the expected output (binary, discrete, bounde, unbounded). Concerning the hidden layers, a popular choice is a family of *Rectified Linear Units*—activation functions usually referred as ReLU:

$$\delta(z) = \mathrm{ReLU}(z) = \max(0, z). \tag{23}$$

Notice that $\mathrm{ReLU}(\mathbb{R}) = (0, +\infty)$, so that the outputs are positive but unbounded (Stutz 2014), (Krizhevsky et al. 2012). A variant of ReLU is PReLU or LeakyReLU (El Jaafari et al. 2021), given by ($a > 0$ is a parameter to be chosen by the user):

$$\delta(z) = \mathrm{PReLU}(z) = \max(0, z) + a\ \min(0, z) \quad (a > 0). \tag{24}$$

$\mathrm{PReLU}(\mathbb{R}) = (-\infty, +\infty)$, so that the outputs are unbounded, and can take negative values.

For the output layer, a popular family of activation functions is the *sigmoid* family. For instance, the *logistic sigmoid*, given by

$$\overline{\delta}(z) = \sigma(z) = \frac{1}{1 + \exp(-z)}, \quad z \in \mathbb{R}. \tag{25}$$

This function has a bounded range: we have $\sigma(\mathbb{R}) = (0, 1)$. From the same family, the *hyperbolic tangent* $\overline{\delta}(z) = th(z) = \tanh(z)$ takes negative values, since $th(\mathbb{R}) = [-1, 1]$, so that th can be used if the user desires bounded outputs including negative values (Stutz 2014), (Duda et al. 2001). These *sigmoid* family is often used when we are interested in binary outputs. However, according to Stutz (2014), Glorot and Bengio (2010) both the *logistic sigmoid* and the *hyperbolic tangent* perform badly in deep learning. These authors recommend the use of the *softsign* activation function:

$$\bar{\delta}(z) = s(z) = \frac{1}{1+|z|}. \tag{26}$$

Nevertheless, some works tend to show that the combination of the family ReLU with the hyperbolic tangent activation produces good results (Jarrett et al. 2009).

When considering classification in η_K classes with neural networks,[6] the user to choose a most relevant output among $\mathbf{h}^{(K)} = (h_1^{(K)}, \dots, h_{\eta_K}^{(K)})$.

Thus, we can design a DNN destined to classify data $\mathbf{X}$ in one among η_K classes, by producing an output that can be interpreted in terms of probability: the class i, $1 \le i \le \eta_K$ corresponds to the unit (K, i) in the output layer and $p_i = h_i^{(K)}$ is a prediction of the probability of the event " $\mathbf{X} \in$ class i" (id est, " $\mathbf{X}$ is member of class i "). Let us denote by Y the class number: $Y \in \{1, \dots, \eta_K\}$: we look for a DNN such that

$$p_i = P(Y = i|\mathbf{X}) \Rightarrow \mathbf{p} = (p_1, \dots, p_{\eta_K})^t \in \mathbb{R}^{\eta_K} : \sum_{i=1}^{\eta_K} p_i = 1, \tag{27}$$

where $P(\bullet|\bullet)$ denotes the conditional probability.

In this case, a popular choice is the *softmax* activation functions, whose outputs can be interpreted as probabilities:

$$\delta_i^{(K)}(\mathbf{a}^{(K)}) = \text{softmax}(\mathbf{a}^{(K)}, i) = \frac{\exp(a_i^{(K)})}{\sum_{j=1}^{\eta_K} \exp(a_j^{(K)})}. \tag{28}$$

Indeed, $p_i = \delta_i^{(K)}(\mathbf{a}^{(K)})$ verifies $p_i > 0$ and $\sum_{j=1}^{\eta_K} p_i = 1$, so that the outputs $p_i = h_i^{(K)}$, $1 \le i \le \eta_K$, can be interpreted as probabilities (Bishop 2006).

3.3 Loss functions for classification

Let be given $\mathbf{X}$ belonging to the class Y: we have as exact probability $p_Y^* = 1$, so that the exact vector of probabilities is $\mathbf{p}^*(Y) = (p_1, \dots, p_{\eta_K})$ such that $p_Y^* = 1$ and $p_i^* = 0$, if $i \ne Y$—it is a *one-hot* vector.[7]

The gap between the probabilities of the classes furnished by $\mathbf{p} = \boldsymbol{\tau}(\mathbf{X}, \mathbf{w})$ and the exact probabilities $\mathbf{p}^*$ can be evaluated by the cross-entropy loss function

$$\mathcal{L}\left(\mathbf{p}, \mathbf{p}^*(Y))\right) = -\sum_{j=0}^{\eta_K - 1} p_j^*(Y) \log(p_j). \tag{29}$$

We can consider this loss function under the form

$$\mathcal{L}(\mathbf{p}, Y)) = -\sum_{j=1}^{\eta_K} \mathbf{1}_{Y=j} \log(p_j) \ , \tag{30}$$

[6] The objective is to assign $\mathbf{x}$ to one among η_K discrete classes, using the outputs $\mathbf{h}^{(K)}$ (Bishop 2006; Stutz 2014).

[7] A one-hot vector v is then a binary vector with a single non-zero component, which takes the value 1.

where

$$\mathbf{1}_{Y=j} = \begin{cases} 1 \text{ if } Y = j \\ 0 \text{ otherwise.} \end{cases}$$

The global loss function becomes

$$G(\mathbf{w}) = \frac{1}{n_s} \sum_{s=1}^{n_s} \mathcal{L}(\boldsymbol{\tau}(\mathbf{X}_s, \mathbf{w}), Y_s)), \tag{31}$$

id est,

$$G(\mathbf{w}) = -\frac{1}{n_s} \sum_{s=1}^{n_s} \sum_{j=1}^{\eta_K} \mathbf{1}_{Y_s=j} \log(\tau_j(\mathbf{X}_s, \mathbf{w})), \tag{32}$$

with $h_j^{(K)} = \tau_j(\mathbf{X}, \mathbf{w})$.

3.4 Forward evaluation

The evaluation of the objective function f requests the determination of the global outputs $\mathbf{h}_s^{(K)} = \boldsymbol{\tau}(\mathbf{X}_s, \mathbf{w})$, for $1 \le s \le n_s$. As usual in the framework of DNN, such a determination is carried by *forward propagation*, which consists in the successive determination of the outputs $\mathbf{h}^{(\ell)}$, for $0 \le \ell \le K$. Forward propagation implements the evaluation of $\boldsymbol{\tau}(\mathbf{X}, \mathbf{w})$, by generating the finite sequence $\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(K)}$, defined by Eqs. (14)–(15), with the initial data $\boldsymbol{h}^{(0)} = \mathbf{X}$. For the classification of a image, $\mathbf{X}$ is a $\eta \times \eta$ square matrix containing values of the pixels—for a black-and-white image, the pixels take either the value 0 either the value 1, so that $\mathbf{X} \in \{0, 1\}^{\eta \times \eta}$. The final outputs of the net are the probabilities of membership, as described in the preceding section: we have $0 \le h_j^{(K)} \le 1$, so that $\boldsymbol{\tau}(\mathbf{X}, \mathbf{w}) \in [0, 1]^{\eta_k}$ and the forward propagation implements a map

$$\boldsymbol{\tau}_{\mathbf{w}} : \{0, 1\}^{\eta \times \eta} \mapsto [0, 1]^{\eta_k}, \ \boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}) = \boldsymbol{\tau}(\mathbf{X}, \mathbf{w}) . \tag{33}$$

3.5 Backpropagation

To implement the optimization methods described in Sect. 2, we need to evaluate the gradient of the global loss function f with respect to the weigths $\mathbf{w}$. In the framework of DNN, such an evaluation is performed by *backpropagation*., which corresponds to the successive application of the chain rule for the derivation of the composition of functions. The rule is successively applied from the last layer to the first one, to generate $\partial_{\mathbf{w}} G = \partial G \big/ \partial \mathbf{w}$, id est, the derivatives $\partial_{w_{ij}^{(\ell)}} G = \partial G / \partial w_{ij}^{(\ell)}$, corresponding to each weight $w_{ij}^{(\ell)}$ of the net: $0 \le j \le \eta_{\ell-1}, 1 \le i \le \eta_\ell, 1 \le \ell \le K$. As previously observed, the global loss functions are defined by Eqs. (20), (21), (31) and involves a loss function $\mathcal{L}$ destined to evaluate the gap between the prediction $\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X})$ and the real value $\mathbf{p}^*(Y)$. From Eq. (31):

$$\partial_{\mathbf{w}} G = \frac{1}{n_s} \sum_{s=1}^{n_s} \partial_{\mathbf{w}} \mathcal{L}(\boldsymbol{\tau}(\mathbf{X}_s, \mathbf{w}), Y_s)) \quad , \tag{34}$$

Thus, we must evaluate the derivatives $\partial_{\mathbf{w}} \mathcal{L}$, id est, $\partial_{w_{ij}^{(\ell)}} \mathcal{L}$, for $0 \le j \le \eta_{\ell-1}, 1 \le i \le \eta_\ell$, $1 \le \ell \le K$. The derivatives can be evaluated by recurrence, using Eqs. (14)–(15) or Eq.

(16). Indeed, we observe that

$$\partial_{h_s^{(\ell-1)}} h_t^{(\ell)} = D_t^{(\ell)} w_{ts}^{(\ell)} \ , \ D_t^{(\ell)} = \left(\delta_t^{(\ell)}\right)' (a_t^{(\ell)}). \tag{35}$$

Moreover,

$$\partial_{w_{ij}^{(\ell)}} h_t^{(\ell)} = D_t^{(\ell)} \ \partial_{w_{ij}^{(\ell)}} a_t^{(\ell)} = \sum_{s=0}^{\eta_{\ell-1}} D_t^{(\ell)} w_{ts}^{(\ell)} \partial_{w_{ij}^{(\ell)}} h_s^{(\ell-1)} \ , \tag{36}$$

so that

$$\partial_{w_{ij}^{(\ell)}} h_t^{(\ell)} = \sum_{s=0}^{\eta_{\ell-1}} \partial_{h_s^{(\ell-1)}} h_t^{(\ell)} \partial_{w_{ij}^{(\ell)}} h_s^{(\ell-1)} \ , \tag{37}$$

and

$$\partial_{w_{ij}^{(\ell)}} \mathbf{h}_t^{(\ell)} = \partial_{\mathbf{h}^{(\ell-1)}} \mathbf{h}^{(\ell)} \partial_{w_{ij}^{(\ell)}} \mathbf{h}^{(\ell-1)} \ . \tag{38}$$

Thus,

$$\partial_{\mathbf{w}} \mathbf{h}_t^{(\ell)} = \partial_{\mathbf{h}^{(\ell-1)}} \mathbf{h}^{(\ell)} \partial_{\mathbf{w}} \mathbf{h}^{(\ell-1)} \ . \tag{39}$$

We have

$$\partial_{\mathbf{w}} \mathcal{L}(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}), Y) = \partial_{\boldsymbol{\tau}_{\mathbf{w}}} \mathcal{L}(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}), Y) \ \partial_{\mathbf{w}} \boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}) \ , \tag{40}$$

id est,

$$\partial_{\mathbf{w}} \mathcal{L}(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}), Y) = \partial_{\boldsymbol{\tau}_{\mathbf{w}}} \mathcal{L}(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}), Y) \ \partial_{\mathbf{w}} \mathbf{h}^{(K)}. \tag{41}$$

Thus, Eq. (39) yields that, for $0 \le \ell \le K-1$

$$\partial_{\mathbf{w}} \mathcal{L}(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}), Y) = \partial_{\mathbf{h}^{(K)}} \mathcal{L}(\mathbf{h}^{(K)}, Y) \left[\prod_{i=0}^{K-\ell-1} \partial_{\mathbf{h}^{(K-i-1)}} \mathbf{h}^{(K-i)} \right] \partial_{\mathbf{w}} \mathbf{h}^{(\ell)} \ . \tag{42}$$

Equations (35) and (42) furnish $\partial_{\mathbf{w}} \mathcal{L}$. Then, Eq. (34) furnishes $\partial_{\mathbf{w}} G$. Practical implementation is made by the algorithms of backpropagation, such as:

Algorithm 1 Error backpropagation algorithm.

External Data: a sample $\mathcal{D} = \{(\mathbf{X}_s, Y_s)\ , i = 1, \ldots n_s\}$.
Input: weights $\mathbf{w}$.
Output: subgradient of loss function for $\mathbf{w}$.

```
(1) For a sample $\mathcal{D}$, propagate the inputs $\mathbf{X}_s$ through the network to compute the outputs $(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}_s)$ (in topological order).
(2) Compute the loss $\mathcal{L}_s := \mathcal{L}(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}_s), Y_s)$
(3) Compute the subgradient $\partial_{\mathbf{w}} G(\mathbf{w})$ using Eqs. (35), (42) and (34).
```

3.6 Regularization

Multilayer perceptrons with at least one hidden layer have been shown to approximate any target mapping to arbitrary precision. As a result, the training data may be overfitted, with a low training error on the training set but a large training error on unknown data (Bengio 2009). Regularization is the job of avoiding overfitting to improve generalization performance, which means that the trained network should also perform well on unknown data (Haykin 2005).

As a result, the training set is frequently divided into two parts: real training and validation. The neural network is subsequently trained with the new training set, and its generalization performance is assessed using the validation set (Duda et al. 2001).

Regularization can be done in a variety of ways. The training set is frequently supplemented to include particular invariances that the network is supposed to learn (Krizhevsky et al. 2012). Other methods include a regularization component in the error measure to manage the solution's complexity and form (Bishop 1995)—the objective function to be minimized is modified by the adjonction of a penalty term:

$$G_\lambda(\mathbf{w}) = G(\mathbf{w}) + \lambda\varphi(\mathbf{w}), \tag{43}$$

where $\varphi(\mathbf{w})$ affects the solution's shape and λ is a balancing parameter. A popular φ is the ℓ_2-regularization[8]

$$\varphi(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{\ell=1}^{K}\sum_{i=1}^{\eta_\ell}\sum_{j=1}^{\eta_{\ell-1}} \left(w_{ij}^{(\ell)}\right)^2.$$

The goal is to punish big weights because they are associated with overfitting (Bishop 1995). More generally, we can consider ℓ_p-regularization. For instance, for $p = 1$, ℓ_1-regularization[9] is used in El Mouatasim (2015, 2019) to enforce sparsity of the weights, that is many of the weights should vanish.

3.6.1 Early stopping

While the error on the training set tends to decrease with the number of iterations, once the network starts to overfit the training set, the error on the validation set usually starts to climb again. To minimize overfitting, training should be halted as soon as the error on the validation set reaches a minimum, i.e. before the error on the validation set rises again (Bishop 1995). This technique is known as early stopping.

3.6.2 Dropout

Another regularization strategy based on human brain observation is proposed in Hinton et al. (2012). Each hidden unit is skipped with probability $P = \frac{1}{2}$ whenever the neural network is given a training sample (Hinton et al. 2012). This method can be interpreted in a variety of ways. Units cannot, for starters, rely on the presence of other units. Second, this strategy allows for the simultaneous training of numerous distinct networks. As a result, dropout can be equated to model averaging.[10]
In this situation, nearly half of the neurons are inactive and are not regarded to be part of the neural network. As you can see, the neural network grows more basic.

Overfitting can be reduced by using a simpler version of the neural network with less complexity. At each forward propagation and weight update step, neurons with a particular probability P are deactivated.

[8] Weight decay is a term used to describe the ℓ_2-regularization; see Bishop (1995) for more information.

[9] For $p = 1$, the norm $\|\cdot\|_1$ is defined as $\|\mathbf{w}\|_1 = \sum_{\ell=1}^{K}\sum_{i=1}^{\eta_\ell}\sum_{j=1}^{\eta_{\ell-1}} |w_{ij}^{(\ell)}|$.

[10] By averaging the predictions of different models, model averaging attempts to reduce inaccuracy (Hinton et al. 2012).

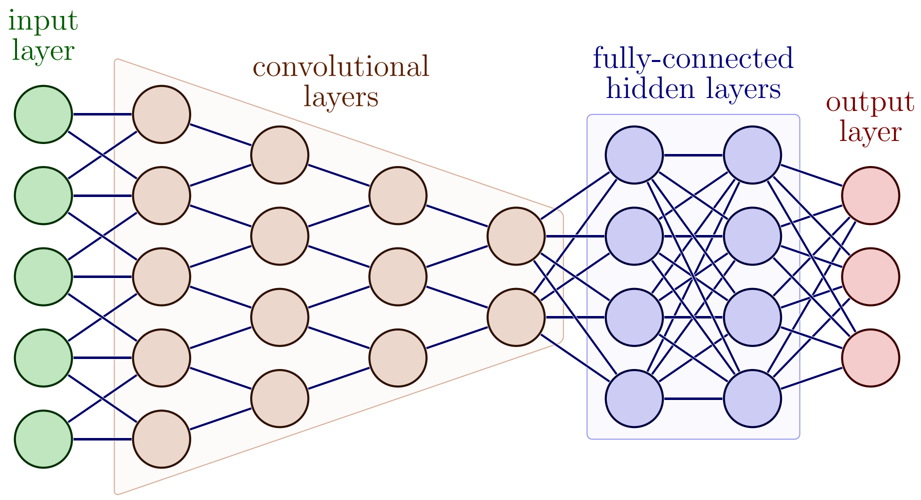


Fig. 4 Flow diagram of deep CNN (Figure from Izaak (2022), under licence CC BY-SA 4.0)

3.7 Convolutional neural networks

The mathematical modeling of CNN including convolution, convolutional layer, non-linearity layer, local contrast normalization layer, fully connected layer and architectures is given in Stutz (2014).

We look at the architecture employed in Krizhevsky et al. (2012) as an example of a modern CNN that performs well on the ImageNet Dataset (Zeiler and Fergus 2013). Five convolutional layers are followed by a rectified linear unit non-linearity layer, brightness normalization, and overlapping pooling in the architecture. Three additional fully-connected layers are used for classification. Krizhevsky et al. (2012) employs dropout as a regularization strategy to avoid overfitting.

The authors of Ciresan et al. (2012) integrate many deep CNN with comparable architectures and average their classification/prediction results. The multi-column deep CNN is the name for this architecture.

We summarized the description of the CNNs by the flow diagram in Fig. 4.

4 Optimization DNN

The goal of this section is given the mathematical modeling of optimization DNN including analysis of the optimization problem, SGD, Subgradient algorithm, control learning rate and SPSA.

4.1 Nonconvex optimization DNN problem

Piecewise affine activation functions provide a dilemma since the resultant mathematical issue is a multicomposite optimization problem with associated nonconvexity and non differentiability. The variable linear transformation from one layer to the next, where the transformation matrix is multiplied by the variable input of that layer, which is determined from the previous layer, causes the nonconvexity. By gradually aggregating the resultant bilinearity over all layers, a highly nonconvex composite objective function may be reduced. This property of linked nonconvexity and nondifferentiability poses significant hurdles for solving the optimization issue using a mathematically rigorous approach with a proved guarantee of convergence to some form of stable solution. Indeed, there is no known solution that can achieve the latter stated aim as a stand-alone deterministic optimization problem (Cui et al. 2020). In particular, usually SGD algorithms converges to suitable local minimum even though the problem itself is highly nonconvex optimization, for more about analysis of the optimization error, see for instance (Kutyniok 2022).

4.2 Stochastic gradient descent (SGD)

SGD techniques are prominent strategies for estimating the parameters $\mathbf{w}$ of DNN.

(I) **Forward**: Let $\mathbf{w}_{(k)}$ be the current weights.
For $\ell = 1$ to K

- Calculate the predicted values $(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}_s), s = 1, \ldots, n_s$.
- Calculate the corresponding values of $(\mathbf{a}_s^{(\ell)}, \mathbf{h}_s^{(\ell)}), s = 1, \ldots, n_s$.

(II) **Backpropagation**:

- Calculate the gradient of the output

$$\nabla_{a^{(K)}(x)} \mathcal{L}(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}_s), Y_s) = \boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}_s) - \mathbf{p}^*(Y_s),$$

  where $\mathbf{p}^*(Y_s) \in \mathbb{R}^{\eta_K}$ is a *one-hot* vector such that $p^*_{Y_s} = 1$, id est, $p^*_i = \mathbf{1}_{i=Y_s}$, $1 \leq \imath \leq \eta_K$.
- For $\ell = K$ to 1 - Calculate the gradient at hidden layer ℓ using Eqs. (35), (42) and (34)

(III) **Mini-Batch**: The technique employed in SGD algorithms is to replace the Eq. (31) of classification error with

$$f(\mathbf{w}, s) = \frac{-1}{|M_s|} \sum_{i:z_i \in M_s} \sum_{j=1}^{\eta_K} \mathbf{1}_{Y_s=j} \log(\boldsymbol{\tau}_{\mathbf{w}}(\mathbf{X}_s)), \tag{44}$$

for each iteration s, where M_s is mini-batch and $|M_s|$ is mini-batch size.
Then

$$\mathbf{E}[G(\mathbf{w}, s)] = G(\mathbf{w}),$$

where $\mathbf{E}[.]$ is mathematical expectation, and

$$\mathbf{E}[\nabla_{\mathbf{w}} G(\mathbf{w}, t)] = \nabla_{\mathbf{w}} G(\mathbf{w}).$$

are satisfied by Eq. (44).

However, prior to testing, the sample is randomly mixed, sorted into batches of a defined size, and each batch is used one at a time until all of the samples have been gathered. This is then repeated in the same order as the previous batch. Then, starting with the first batch and finishing with the last, each epoch is made up of sequential batches that encompass all of the training data.

Algorithm 2 The general gradient descent algorithm.

Input: initial weights $\mathbf{w}_{(0)}$, maximum number of iterations $k_{\max}$
Output: final weights $\mathbf{w}_{(k_{\max})}$

```
1. for $k = 0$ to $k_{\max} - 1$
2.     estimate $\nabla G\left(\mathbf{w}_{(k)}\right)$
3.     compute $\mathbf{g}_{(k)} = -\nabla G\left(\mathbf{w}_{(k)}\right)$
4.     compute learning rate $\rho_{(k)} = \arg\min_{\rho}\left\{G\left(\mathbf{w}_{(k)} - \rho\mathbf{g}_{(k)}\right)\right\}$.
5.     $\mathbf{w}_{(k+1)} := \mathbf{w}_{(k)} + \rho_{(k)}\mathbf{g}_{(k)}$
6. return $\mathbf{w}_{(k_{\max})}$
```

4.3 Subgradient algorithm

To determine the global optimum, the DNN algorithm employs an optimization strategy. The SGD algorithm, a well-referenced artificial intelligence function to model a first-order optimization technique that helps us identify a local minimum, was employed in the neural network algorithm.

A subgradient algorithm is an unconstrained minimization method that's used to reduce the size of a function, in this case the loss function. In this section, the loss function (43) can be presented the objective function of the nonconvex nonsmooth optimization problem 1.

The goal was to find a $\mathbf{w}$ value that produced the least amount of error and allowed the loss function to attain a local minimum. In this strategy, each iteration seeks to find a new $\mathbf{w}$ value that gives a somewhat smaller error than the preceding iteration.

Typically, (1) computational methods rely on the usage of subgradient iteration algorithms, which supply information at any point $\mathbf{w}$, the value of the loss function $G(\mathbf{w})$, as well as a subgradient $\mathbf{g}$ from the subdifferential set $\partial G(\mathbf{w})$.
Remember that every vector $\mathbf{g}$ that fulfills the inequality

$$G(\mathbf{y}) \geq G(\mathbf{w}) + \mathbf{g}^t(\mathbf{y} - \mathbf{w}) \quad \forall \mathbf{y} \in \mathbb{R}^n, \tag{45}$$

is a subgradient of $G(\mathbf{w})$ at $\mathbf{w}$.

Since the subgradient algorithm with learning rate ρ starts with some initial $\mathbf{w}_{(0)}$ and updates it repeatedly:

$$\mathbf{w}_{(k+1)} = \mathbf{w}^k - \rho^k\mathbf{g}^k, \quad \mathbf{g}^k \in \partial G(\mathbf{w}^k), \quad k = 0, 1, \ldots \tag{46}$$

which have been the subject of extensive research since the 1960s.

A convex function is known to be subdifferentiable at all points in its domain. The subdifferential is also a non-empty convex, closed, and bounded set (Dem'vanov and Vasil'ev 1985). At all locations in its domain, a convex function isn't necessarily differentiable. If a convex function is differentiable at a point $\mathbf{w}$, then $\nabla G(\mathbf{w})$ is a subgradient of G at $\mathbf{w}$. Also we have

$$\partial G(\mathbf{w}) = \{\nabla G(\mathbf{w})\}.$$

Subgradient can be thought of as a generalized gradient of a convex function in this context.

For nonconvex function $\partial G(\mathbf{w})$ is Clarke subdifferential at point $\mathbf{w} \in \mathbb{E}$ (Bagirov et al. 2013). A learning rate is also employed to govern the size of the downward step we take throughout each iteration. For learning rate $\rho_{(k)}$ meeting the "divergence series" criterion, it was proven that (46) converges under relatively mild conditions:

$$\rho_{(k)} \xrightarrow{k\to\infty} 0+, \quad \sum_{k} \rho_{(k)} = \infty. \tag{47}$$

Theorem 1 *Let G satisfy the asumptions on the objective function of optimization problem 1. Let the sequence $\{\mathbf{w}_{(k)} : k \in I\!N\}$ given by the formula (46) and the learning rate $\rho_{(k)}$ satisfy the conditions (47). Then $G(\mathbf{w}_{(k)}) \to G^*$.*

Proof To demonstrate the theorem, we must show that

$$\forall \epsilon > 0, \exists n_k \in \mathbb{N} \text{ such that } k \geq n_k \Rightarrow G\left(\mathbf{w}_{(k)}\right) - G\left(\mathbf{w}^*\right) < \epsilon.$$

Let us proof it by *reductio ad absurdum*: assume that $\exists \epsilon > 0$ such that

$$\forall k \in \mathbb{N} : \quad G(\mathbf{w}_{(\sigma_k)}) - G(\mathbf{w}^*) \geq \epsilon, \tag{48}$$

where $G(\mathbf{w}_{(\sigma_k)})$ is a subsequence of $G(\mathbf{w}_{(k)})$.

Let $\mathbf{w} = \mathbf{w}_{(\sigma_k)}$, $y = \mathbf{w}^*$ in (45) then:

$$G(\mathbf{w}^*) \geq G(\mathbf{w}_{(\sigma_k)}) + \mathbf{g}^t_{(\sigma_k)}\left(\mathbf{w}^* - \mathbf{w}_{(\sigma_k)}\right), \tag{49}$$

from (48) and (49) we have:

$$\mathbf{g}^t_{(\sigma_k)}\left(\mathbf{w}_{(\sigma_k)} - \mathbf{w}^*\right) \geq \epsilon. \tag{50}$$

Since $\rho_{(\sigma_k)} > 0$, multiplying (50) by $-2\rho_{(\sigma_k)}$ yields

$$-2\rho_{(\sigma_k)}\mathbf{g}^t_{(\sigma_k)}\left(\mathbf{w}_{(\sigma_k)} - w^*\right) \leq -2\rho_{(\sigma_k)}\epsilon.$$

As a consequence, Eq. (46) is used in the estimations that follow:

$$\begin{aligned} \|\mathbf{w}_{(\sigma_k+1)} - \mathbf{w}^*\|^2 &= \|\mathbf{w}_{(\sigma_k)} - \rho_{(\sigma_k)}\mathbf{g}_{(\sigma_k)} - \mathbf{w}^*\|^2 \\ &= \|\mathbf{w}_{(\sigma_k)} - \mathbf{w}^*\|^2 + (\rho_{(\sigma_k)})^2\|\mathbf{g}_{(\sigma_k)}\|^2 \\ &\quad - 2\rho_{(\sigma_k)}\mathbf{g}^t_{(\sigma_k)}\left(\mathbf{w}_{(\sigma_k)} - \mathbf{w}^*\right) \\ &\leq \|\mathbf{w}_{(\sigma_k)} - \mathbf{w}^*\|^2 + (\rho_{(\sigma_k)})^2\|\mathbf{g}_{(\sigma_k)}\|^2 - 2\rho_{(\sigma_k)}\epsilon. \end{aligned}$$

Since $\partial G(\mathbf{w}_{(\sigma_k)})$ is a bounded set (Dem'vanov and Vasil'ev 1985), then $\exists M > 0$ such that $\|\mathbf{g}_{(\sigma_k)}\|^2 \leq M$.
Using the first condition of (47) $\exists n_k \in \mathbb{N}$ such that

$$\forall k \geq n_k : \quad \rho_{(\sigma_k)} \leq \frac{\epsilon}{M} \Leftrightarrow \rho_{(\sigma_k)}M \leq \epsilon.$$

Then we have:

$$\begin{aligned} \forall k \geq n_k : \ \|\mathbf{w}_{(\sigma_k)} - \mathbf{w}^*\|^2 &\leq \|\mathbf{w}_{(\sigma_k)} - \mathbf{w}^*\|^2 + \rho_{(\sigma_k)}(\rho_{(\sigma_k)}M - 2\epsilon) \\ &\leq \|\mathbf{w}_{(\sigma_k)} - \mathbf{w}^*\|^2 - \rho_{(\sigma_k)}\epsilon. \end{aligned}$$

Recursively expressed, this last inequality produces for every arbitrary integer $n > n_k$:

$$\|\mathbf{w}_{(n+1)} - \mathbf{w}^*\|^2 \leq \|\mathbf{w}_{(n_k)} - \mathbf{w}^*\|^2 - \epsilon \sum_{k=n_k}^{n} \rho_{(\sigma_k)}. \tag{51}$$

Using the second condition of (47), the right side of (51) tends to $-\infty$, so that a contradiction with $0 \leq \|\mathbf{w^{n+1}} - \mathbf{w}^*\|^2 \leq -\infty$, what is a contradiction. Thus, we conclude that the theorem is true. □

4.4 Control learning rate

However, numerical experimentation and theoretical research revealed that this learning rate rule leads to delayed convergence as a rule, and future development followed in the footsteps of the subgradient technique (El Mouatasim 2015).

Let $\Psi(\rho) = G(\mathbf{w} - \rho\mathbf{g})$, $\mathbf{g}G \in \partial G(\mathbf{w})$. Let us consider ψ a smooth quadratic approximation of Ψ, verifying the following conditions: $\Psi(0) = \psi(0)$, $\Psi(\rho) = \psi(\rho)$ and $\psi'(0) \in \partial\Psi(0)$, where $\psi'(\mathbf{w})$ is the derivative of ψ at point $\mathbf{w}$. For instance, ψ can be defined as follows:

$$\psi(h) = G(\mathbf{w}) - \|\mathbf{g}G\|^2 h + \nu_\rho \frac{\|\mathbf{g}\|^2}{\rho} h^2,$$

where $\nu_\rho = \dfrac{\Psi(\rho) - (G(\mathbf{w}) - \rho\|\mathbf{g}\|^2)}{\rho\|\mathbf{g}\|^2}$. Then

$$\psi'(\rho) = -\|\mathbf{g}\|^2 + 2\nu_\rho \frac{\|\mathbf{g}\|^2}{\rho}\rho = \|\mathbf{g}\|^2(2\nu_\rho - 1).$$

This suggests that if $\nu_\rho \leq 0.5$; the derivative of function ψ at point ρ is negative, we should increasing the learning rate to reduce loss function G. If, on the other hand, $\nu_\rho \geq 0.5$; the derivative of function ψ at point ρ is positive, and so the learning rate should be decreasing to reduce loss function G (Wójcik1 et al. 2018).

Let us suppose a learning rate of $\rho_{(k)}$ was used at iteration k and let us identify at $\mathbf{w}_{(k+1)}$ relation between $\rho_{(k)}$ and the learning rate providing minimization of G in the direction $-\mathbf{g}^k$ (El Mouatasim 2020; Uryas'ev 1991).

In this paper, we propose this control learning rate:

$$\rho_{(k+1)} = \begin{cases} \rho_{\text{incr}}\rho_{(k)}, & \text{if } \nu_\rho \leq 0.5, \\ \rho_{\text{decr}}\rho_{(k)}, & \text{if } \nu_\rho \geq 0.5, \end{cases} \tag{52}$$

where:

$\rho_{(0)}0$ is starting learning rate,
$\rho_{\text{incr}} > 1$ is a increasing coefficient,
$\rho_{\text{decr}} = \dfrac{1}{\rho_{\text{incr}}} < 1$ is a decreasing coefficient.

Theorem 2 *If the sequence* $\{\mathbf{w}_{(k)} : k \in I\!N\}$ *given by the formula* (46) *and the learning rate* $\rho_{(k)}$ *given by* (52)*, then* $G(\mathbf{w}_{(k)}) \to G^*$.

Proof Let $\left\{\rho_{(k_j)} | \rho_{(k_j)} = \rho_{(0)}\rho_{\text{incr}}^{k_j}\right\}$ be a subsequence of a sequence $\{\rho_{(k)} : k \in \mathbb{N}\}$, since $\rho_{incr} > 1$, we have $\sum_{j=1}^{\infty} \rho_{(k_j)} = +\infty$. The problem (1) is nonconvex and we assume $\exists M > 0$ such that $\rho^k \leq M\rho_{\text{decr}}^k$. Since $\rho_{\text{decr}} < 1$ then $\rho_{\text{decr}}^k \to +0$. Thus, we have also

$$\rho_{(k)} \xrightarrow{k\to\infty} +0. \tag{53}$$

The conditions of (47) are satisfied by (53). So we can apply the Theorem 1. □

4.5 Stochastic perturbation of subgradient algorithm (SPSA)

The main difficulty remains the lack of convexity: if G is nonconvex, the Kuhn–Tucker points may not correspond to global minimum (El Mouatasim et al. 2006, 2011). In the sequel, we shall improve this point by using an appropriate random perturbation: as previously observed, the real quantities are replaced by random variables (Eqs. (9)–(10)). Since

$$\mathbf{Q}\left(\mathbf{W}_{(k)}, \mathbf{D}_{(k)}, \rho_{(k)}\right) + \mathbf{P}_{(k)} = \mathbf{Q}\left(\mathbf{W}_{(k)}, \mathbf{D}_{(k)} + \mathbf{P}_{(k)}/\rho_k, \rho_k\right), \tag{54}$$

the stochastic iterations may be considered as perturbations of the descent direction $\mathbf{D}_{(k)}$. In the sequel, we describe the general properties of these elements leading to convenient sequences and we show that sequences of Gaussian vectors may be used.

A simple way for the generation of a convenient sequence of perturbations $\left\{\mathbf{P}_{(k)} : k \in_N\right\}$ is

$$\mathbf{P}_{(k)} = \xi_{(k)}\mathbf{Z}_{(k)},$$

where

1. $\left\{\xi_{(k)} : k \in \mathbb{N}\right\}$ is a nonincreasing sequence of *strictly positive* real numbers *converging to zero* and such that $\xi_{(0)} \leq 1$.
2. $\left\{\mathbf{Z}_{(k)} : k \in \mathbb{N}\right\}$ is a sequence of random vectors taking their values on E.

Let us introduce $U_{(k)} = G(\mathbf{W}_{(k)})$. Since, at each iteration number $k \geq 0$, the step $\rho_{(k)}$ is determined into a way that reduces the value of the objective function (Eq. (7)), the sequence $\left\{ U_{(k)} : k \in \mathbb{N}\right\}$ is decreasing by construction. Moreover, it has a lower bound given by G^*.

$$\forall k \geq 0 \ : \ G^* \leq U_{(k+1)} \leq U_{(k)}. \tag{55}$$

Thus, $\left\{U_{(k)} : k \in \mathbb{N}\right\}$ is nonincreasing and bounded from below by m: there exists $U \geq m$ such that $U_{(k)} \to U$ for $k \to +\infty$. The aim is to establish that $U = G^*$.

Theorem 3 *Let $\mathbf{Z}_{(k)} = \mathbf{Z}$, where $\mathbf{Z}$ is a random variable following $N(\mathbf{0}, \sigma\mathbf{Id})$, $(\sigma > 0)$ and let*

$$\xi_{(k)} = \sqrt{\frac{a}{\log(k+d)}}, \tag{56}$$

where $a > 0$, $d > 0$ and k is the iteration number. Then, for a large enough, $U = l^*$ almost surely.

Proof See, for instance, Pogu and Souza de Cursi (1994) and El Mouatasim et al. (2006). □

5 Computational experiment

All of the tests were run on a personal PC with an HP i5 CPU processor running at 1.20GHz, 4 GB of RAM, and Python 3.9 for Windows 10 installed.

In this paper, we implement CNN (LeNet-5) using PyTorch3, an open source Python library for deep learning classification.

5.1 Algorithm of SPSA

The pseudocode of SPSA is given as follows:

The pytorch implementation of SPSA algorithm is available in public repository https://github.com/el-mouatasim/SPSA.

Algorithm 3 Stochastic perturbation of subgradient algorithm (SPSA).

Input: initial weights $\mathbf{w}_{(0)}$, number of global iterations k_{glob}number of local iterations $k_{\max}$ number of stochastic perturbations n_{sto}.
Output: final optimal weights $\mathbf{w}_{(k_{opt})}$.

```
1.  set $\mathbf{w}_{(k_{opt})} = \mathbf{w}_{(0)}$
2.  for $k_g = 0$ to $k_{glob} - 1$
3.  set $\mathbf{w}_{ac}^{0} = \mathbf{w}_{(k_{opt})}$ and $\mathbf{w}_{ac}^{opt} = \mathbf{w}_{(k_{opt})}$
4.  for $k = 0$ to $k_{\max} - 1$
5.      estimate the subgradient $\mathbf{g}_{(k)} \in \partial G(\mathbf{w}_{ac}^{k})$
6.      select control learning rate $\rho_{(k)}$ by Eq. (52)
7.      calculate $\mathbf{w}_{ac}^{k+1} := \mathbf{w}_{ac}^{k} - \rho^{k}\mathbf{g}^{k}$
8.      if $G(\mathbf{w}_{ac}^{k+1}) \leq G(\mathbf{w}_{ac}^{opt})$
9.          set $\mathbf{w}_{ac}^{opt} = \mathbf{w}_{ac}^{k+1}$
10. return $\mathbf{w}_{ac}^{opt}$
11. set $\mathbf{w}_{(k_{opt})} = \mathbf{w}_{ac}^{opt}$
12. set $\mathbf{w}_{sto}^{0} = \mathbf{w}_{(k_{opt})}$ and $\mathbf{w}_{sto}^{opt} = \mathbf{w}_{(k_{opt})}$
13. for $k = 1$ to $n_{sto}$
14.     add perturbation stochastic $\mathbf{w}_{sto}^{k} = \mathbf{w}_{sto}^{opt} + \xi_{(k)}\mathbf{Z}_{(k)}$ by Theorem (3)
15.     estimate the subgradient $\mathbf{g}_{(k)} \in \partial G(\mathbf{w}_{sto}^{k})$
16.     select control learning rate $\rho_{(k)}$ by Eq. (52)
17.     calculate $\mathbf{w}_{sto}^{k+1} := \mathbf{w}_{sto}^{k} - \rho_{(k)}\mathbf{g}_{(k)}$
18.     if $G(\mathbf{w}_{sto}^{k+1}) \leq G(\mathbf{w}_{sto}^{opt})$
19.         chose $\mathbf{w}_{sto}^{opt} = \mathbf{w}_{sto}^{k+1}$
20. return $\mathbf{w}_{sto}^{opt}$
21. set $\mathbf{w}_{(k_{opt})} = \mathbf{w}_{sto}^{opt}$
22. return $\mathbf{w}_{(k_{opt})}$
```

Fig. 5 Samples from the MNIST dataset (Figure from Steppan (2022), under licence CC BY-SA 4.0)

5.2 MNIST dataset

MNIST dataset (LeCun and Cortes 2010), consisting 70,000 images 28 × 28 grayscale of handwritten digits in the range of 0 to 9, for a total of 10 classes, which includes 60,000 training and validation, and 10,000 test, Fig. 5.

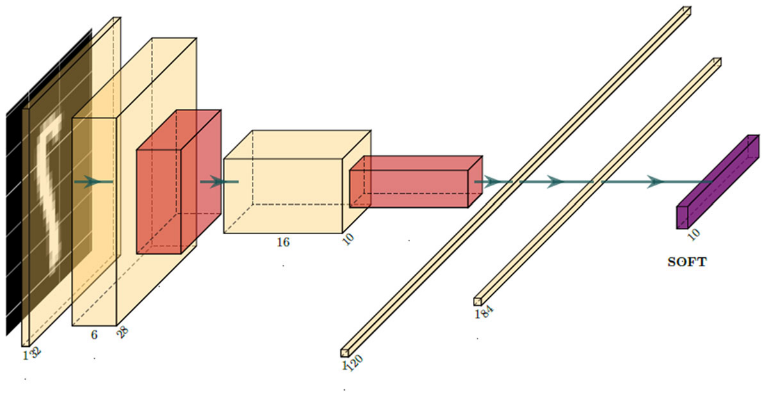


Fig. 6 Architect of the LeNet model

We present digits from the 54,000 MNIST training set to the network LeNet5 to train it, 6000 images for the validation set and 10,000 for test set. A 32 mini-batch size was used.

5.3 LeNet-5 model

Deep learning models might have a lot of hyperparameters that need to be adjusted. The number of layers to add, the number of filters to apply to each layer, whether to subsample, kernel size, stride, padding, dropout, learning rate, momentum, batch size, and so on are all options. Because the number of possible choices for these variables is unlimited, using cross-validation to estimate any of these hyperparameters without specialist GPU technology to expedite the process is extremely challenging.

As a result, we suggest a model the LeNet-5 model (Fig. 6). The LeNet architecture pad the input image with to make it 32 $\times$ 32 pixels, then convolution and subsampling with tanh activation in two layers. The next two layers are completely connected linear layers with tanh activation, followed by a layer of Gaussian connections, which are fully connected nodes that use mean squared-error as the loss function.

5.4 Comparing results

We start learning rate of SPSA by $lr = 1e - 2$ and the coffecient of update $\rho_{\text{incr}} = 1.05$ are applied, We used $K_{\max} = 20$ epochs and additional $K_{\text{sto}} = 5$ stochastic perturbation epochs applied to the best epoch with minimal train loss of subgradient algorithms with control learning rate. also we comparing with popular algorithms such as Adam Kingma and Ba (2015) (with initial learning rate $lr = 1e - 3$), SGD (with initial learning rate $lr = 1e - 2$) and Adagrad. Table 1 give the results of the algorithms in 25 training epochs. Figures 7, 8, 9, 10 exhibit the results of comparing approaches: training loss, training accuracy, validation loss, and validation accuracy, respectively.

The network's classification accuracy was then measured using the class-assigned neuron response in the 10,000 MNIST test set and SPSA for LeNet-5 model. Figure 11 for confusion matrix show how to determine the estimated number by multiplying each neuron's responses by class and then choosing the class with the largest average frequency; also the classification report in Table 2, and Fig. 12 give the predict of random sample in test set.

Table 1 LeNet-5 model: comparison between SPSA, Adam, Adagrad and SGD in terms of accuracy and loss

Algorithm	Training loss	Training accuracy (%)	Validation loss	Validation accuracy (%)	Test loss	Test accuracy (%)
SPSA	0.007	99.75	0.004	98.73	0.043	98.74
Adam	0.008	99.74	0.005	98.83	0.05	98.74
Adagrad	0.015	99.65	0.004	98.78	0.04	98.72
SGD	0.033	99.01	0.004	98.90	0.042	98.65

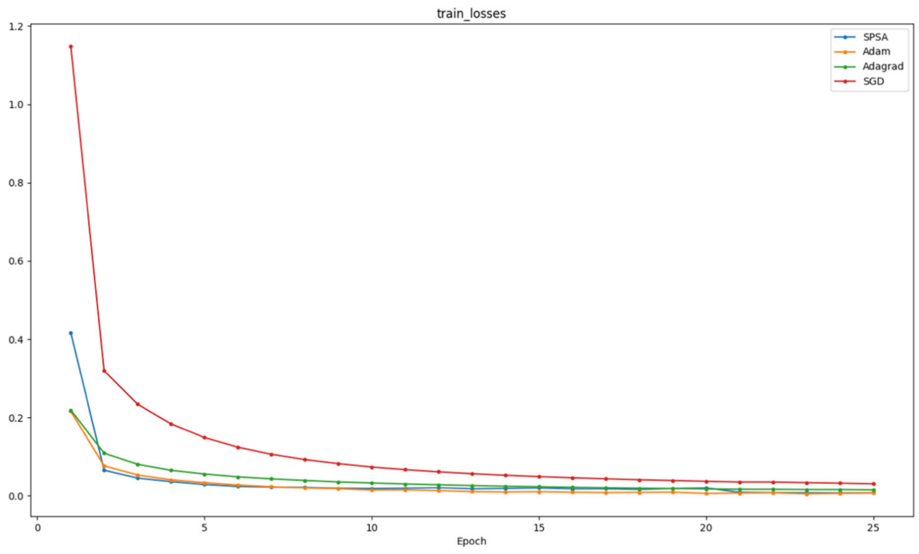


Fig. 7 LeNet-5 model. Comparing training losses results of SPSA, Adam, Adagrad and SGD

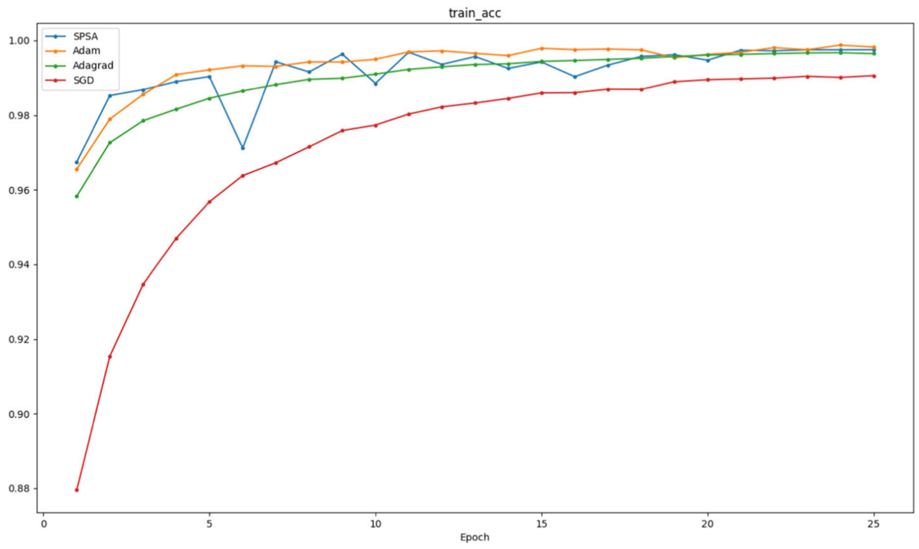


Fig. 8 LeNet-5 model. Comparing training accuracy results of SPSA, Adam, Adagrad and SGD

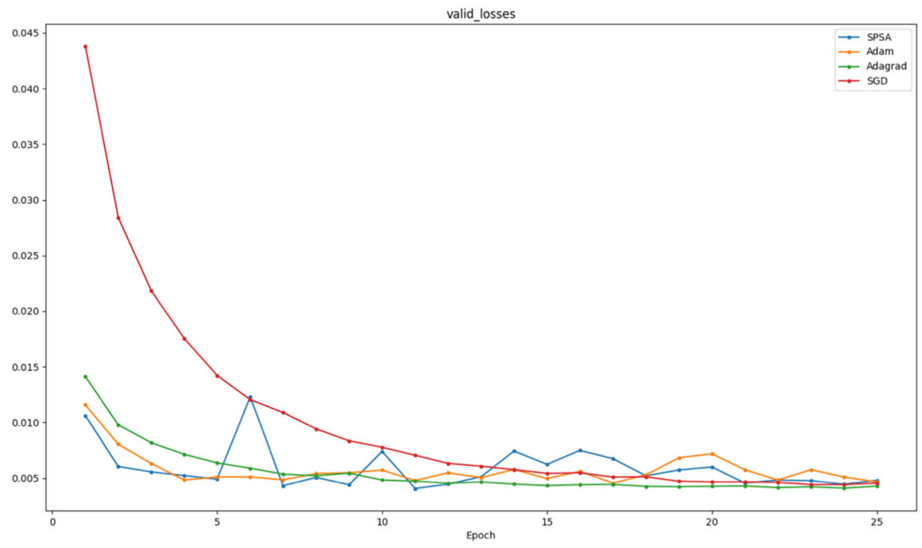


Fig. 9 LeNet-5 model. Comparing evaluation losses results of SPSA, Adam, Adagrad and SGD

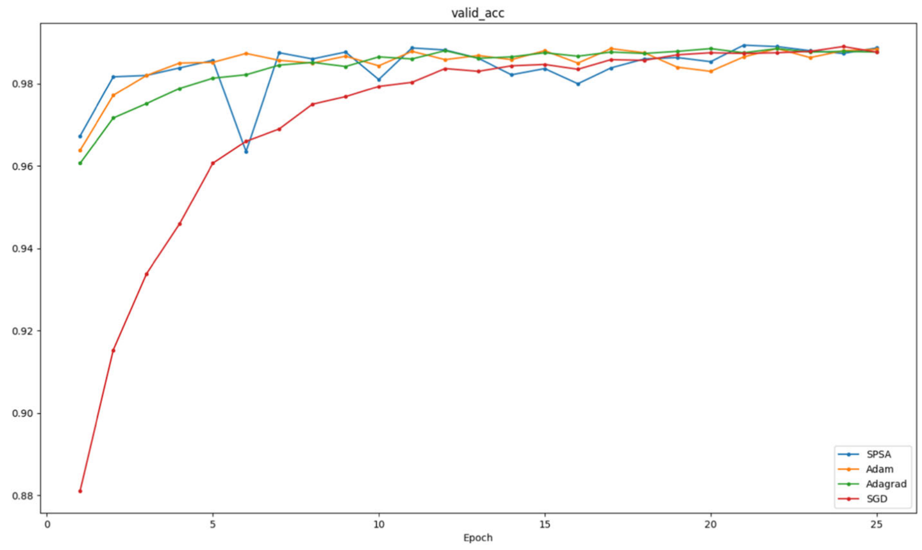


Fig. 10 LeNet-5 model. Comparing evaluation accuracy results of SPSA, Adam, Adagrad and SGD

6 Concluding remarks

We considered the training of DNN including multilayer perceptrons (MP), and CNN for image classification with tanh activation function, ReLU activation function or ℓ_1 regularization. In such a context, training leads to a nonsmooth nonconvex optimization problem, which has been solved by subgradient techniques—SGD with with controlled learning rate, using locally optimal conditions. The results were improved when compared with the literature.

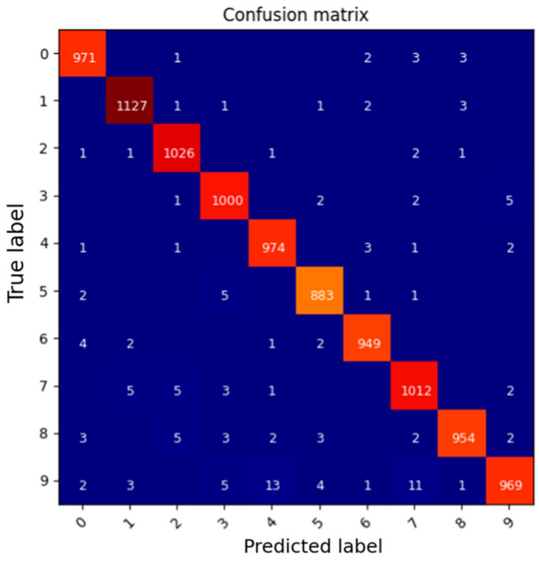


Fig. 11 Average confusion matrix of the testing results was calculated over ten presentations of the MNIST test set digits, using the SPSA for LeNet-5 model

Table 2 LeNet-5 model: classification report for classifier SPSA

	Precision	Recall	f1-score	Support
0	0.99	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.98	0.99	0.99	892
6	0.99	0.98	0.99	958
7	0.98	0.99	0.98	1028
8	0.98	0.98	0.98	974
9	0.99	0.98	0.98	1009
Accuracy			0.99	10,000
Macro avg	0.99	0.99	0.99	10,000
Weighted avg	0.99	0.99	0.99	10,000

Additional stochastic perturbation epochs are applied to find a global solution for nonconvex CNN cases. The convergence theorem of the stochastic perturbation subgradient algorithm with control learning rate (SPSA) was established, and numerical results of SPSA compared to the SGD and its variant algorithms in the dataset MNIST for LeNet-5 model revealed that SPSA is more robust and faster than other approaches when a large number of stochastic perturbations was used.

In future work, we shall consider Hilbert networks (Khalij 2021).

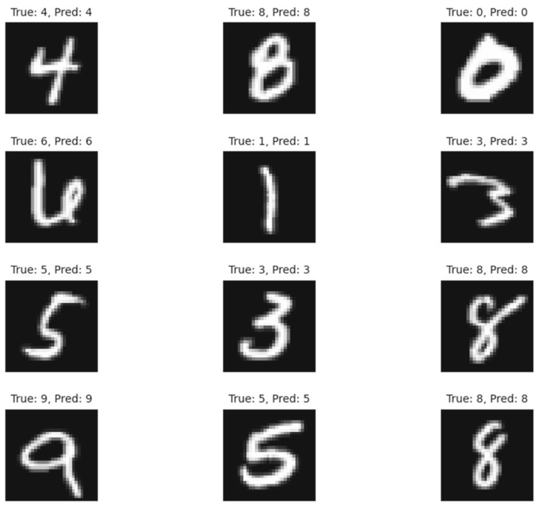


Fig. 12 Predict sample in MNIST test set, using the SPSA for LeNet-5 model

Acknowledgements We are indebted to the anonymous Reviewers and Editors for their many helpful recommendations and insightful remarks that helped us improve the original article.

Declarations

Conflict of interest The authors have no conflicts of interest to declare. All co-authors have seen and agree with the contents of the manuscript "Stochastic Perturbation of Subgradient Algorithm for Nonconvex Deep Neural Networks" and there is no financial interest to report. We certify that the submission is original work and is not under review at any other publication.

References

Bagirov AM, Jin L, Karmitsa N, Al Nuaimat A, Sultanova N (2013) Subgradient method for nonconvex nonsmooth optimization. J Optim Theory Appl 157:416–435
Bengio Y (2009) Learning deep architectures for AI. Found Trends Mach Learn 2(1):1–127
Bishop C (1995) Neural networks for pattern recognition. Clarendon Press, Oxford
Bishop C (2006) Pattern recognition and machine learning. Springer, New York
Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel LD, Monfort M, Muller U, Zhang J et al (2016) End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316
Botev A, Lever G, Barber D (2017) Nesterov's accelerated gradient and momentum as approximations to regularised update descent. In: Neural networks (IJCNN) 2017 international joint conference on, pp 1899–1903
Ciresan DC, Meier U, Schmidhuber J (2012) Multi-column deep neural networks for image classification. Comput Res Repos. arXiv:abs/1202.2745
Cui Y, He Z, Pang J (2020) Multicomposite nonconvex optimization for training deep neural networks. SIAM J Optim 30(2):1693–1723
Dem'vanov VF, Vasil'ev LV (1985) Nondifferentiable optimization. Optimization Software, Inc., Publications Division, New York
Duchi JC, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
Duda R, Hart P, Stork D (2001) Pattern classification. Wiley, New York
El Jaafari I, Ellahyani A, Charfi S (2021) Parametric rectified nonlinear unit (PRenu) for convolution neural networks. J Signal Image Video Process (SIViP) 15:241–246
El Mouatasim A (2018) Implementation of reduced gradient with bisection algorithms for non-convex optimization problem via stochastic perturbation. J Numer Algorithms 78(1):41–62

El Mouatasim A (2019) Control proximal gradient algorithm for ℓ_1 regularization image. J Signal Image Video Process (SIViP) 13(6):1113–1121
El Mouatasim A (2020) Fast gradient descent algorithm for image classification with neural networks. J Signal Image Video Process (SIViP) 14:1565–1572
El Mouatasim A, Wakrim M (2015) Control subgradient algorithm for image regularization. J Signal Image Video Process (SIViP) 9:275–283
El Mouatasim A, Ellaia R, Souza de Cursi JE (2006) Random perturbation of variable metric method for unconstraint nonsmooth nonconvex optimization. Appl Math Comput Sci 16(4):463–474
El Mouatasim A, Ellaia R, Souza de Cursi JE (2011) Projected variable metric method for linear constrained nonsmooth global optimization via perturbation stochastic. Int J Appl Math Comput Sci 21(2):317–329
El Mouatasim A, Ellaia R, Souza de Cursi JE (2014) Stochastic perturbation of reduced gradient & GRG methods for nonconvex programming problems. J Appl Math Comput 226:198–211
Feng J, Lu S (2019) Performance analysis of various activation functions in artificial neural networks. J Phys Conf Ser. https://doi.org/10.1088/1742-6596/1237/2/022030
Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: International conference on artificial intelligence and statistics, pp 249–256
Haykin S (2005) Neural networks a comprehensive foundation. Pearson Education, New Delhi
Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov R (2012) Improving neural networks by preventing co-adaptation of feature detectors. Comput Res Repos. arXiv:abs/1207.0580
Huang K, Hussain A, Wang Q, Zhang R (2019) Deep learning: fundamentals, theory and applications. Springer, Berlin
Jarrett K, Kavukcuogl K, Ranzato M, LeCun Y (2009) What is the best multi-stage architecture for object recognition? In: International conference on computer vision, pp 2146–2153
Josef S (2022) A few samples from the MNIST test dataset. https://commons.wikimedia.org/wiki/File:MnistExamples.png. Accessed 12 Dec. Under Creative Commons Attribution-ShareAlike 4.0 International License
Khalij L, de Cursi ES (2021) Uncertainty quantification in data fitting neural and Hilbert networks. In: Proceedings of the 5th international symposium on uncertainty quantification and stochastic modelling, pp 222–241. https://doi.org/10.1007/978-3-030-53669-5_17
Kingma DP, Ba JL (2015) Adam: a method for stochastic optimization. In: Proceedings of the 3rd international conference on learning representations, San Diego, CA
Konstantin E, Johannes S (2019) A comparison of deep networks with ReLU activation function and linear spline-type methods. Neural Netw 110:232–242
Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. Adv Neural Inf Process Syst 60:1097–1105
Kutyniok G (2022) The mathematics of artificial intelligence. arXiv preprint arXiv:2203.08890
LeCun Y (1989) Generalization and network design strategies. Connect Perspect 19:143–155
LeCun Y, Cortes C (2010) MNIST handwritten digit database
LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551
LeCun Y, Kavukvuoglu K, Farabet C (2010) Convolutional networks and applications in vision. In: International symposium on circuits and systems, vol 5, pp 253–256
Liu Z, Liu H (2019) An efficient gradient method with approximately optimal stepsize based on tensor model for unconstrained optimization. J Optim Theory Appl 181:608–633
Li J, Yang X (2020) A cyclical learning rate method in deep learning training. In: International conference on computer, information and telecommunication systems (CITS), pp 1–5
Minsky ML (1954) Theory of neural-analog reinforcement systems and its application to the brain-model problem. Ph.D. dissertation, Princeton University
Nakamura K, Derbel B, Won K-J, Hong B-W (2021) Learning-rate annealing methods for deep neural networks. Electronics 10:2029
Neutelings I (2022) Graphics with TikZ in LaTeX. Neural networks. https://tikz.net/neura_networks. Accessed 12 Dec. Under Creative Commons Attribution-ShareAlike 4.0 International License
Pelletier C, Webb GI, Petitjean F (2019) Temporal convolutional neural network for the classification of satellite image time series. Remote Sens 11(5):523
Pogu M, Souza de Cursi JE (1994) Global optimization by random perturbation of the gradient method with a fixed parameter. J Global Optim 5:159–180
Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386–408. https://doi.org/10.1037/h0042519
Singh BK, Verma K, Thoke AS (2015) Adaptive gradient descent backpropagation for classification of breast tumors in ultrasound imaging. Procedia Comput Sci 46:1601–1609

Stutz D (2014) Understanding convolutional neural networks. Seminar report, Fakultät für Mathematik, Informatik und Naturwissenschaften

Szandała T (2021) Review and comparison of commonly used activation functions for deep neural networks. In: Bhoi A, Mallick P, Liu CM, Balas V (eds) Bio-inspired neurocomputing. Studies in computational intelligence, vol 903. Springer, Singapore. https://doi.org/10.1007/978-981-15-5495-7_11

Tuyen TT, Hang-Tuan N (2021) Backtracking gradient descent method and some applications in large scale optimisation. Part 2. Appl Math Optim 84:2557–2586

Uryas'ev SP (1991) New variable-metric algorithms for nondifferentiable optimization problems. J Optim Theory Appl 71(2):359–388

Wójcik B, Maziarka L, Tabor J (2018) Automatic learning rate in gradient descent. Schedae Inf 27:47–57

Xinhua L, Qian Y (2015) Face recognition based on deep neural network. Int J Signal Process Image Process Pattern Recogn 8(10):29–38

Zeiler MD, Fergus R (2013) Visualizing and understanding convolutional networks. Comput Res Repos. arXiv:abs/1311.2901