Received 14 October 2022, accepted 31 October 2022, date of publication 14 November 2022, date of current version 22 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3222311

RESEARCH ARTICLE

Itao: A New Iterative Thresholding Algorithm Based Optimizer for Deep Neural Networks

MOHAMED MERROUCHI[1], KHALID ATIFI[2], MUSTAPHA SKITTOU[1], AND TAOUFIQ GADI[1]
[1]Faculty of Science and Technology, Hassan First University of Settat, Settat 26000, Morocco
[2]Faculty of Science and Technology, Cadi Ayyad University of Marrakech, Marrakesh 40000, Morocco

Corresponding author: Mohamed Merrouchi (m.merrouchi@uhp.ac.ma)

ABSTRACT In this paper, we propose a new iterative thresholding algorithm based optimizer (Itao) for deep neural networks. It is a first-order gradient-based algorithm with Tikhonov regularization for stochastic objective functions. It is fast and straightforward to implement. It acts on the parameters and their gradients, with respect to the objective function, in only one step in the backpropagation system when training a neural network. This reduces the learning time and makes it well suited for neural networks with large parameters and/or large datasets. We have experimented this algorithm on several types of loss functions such as mean squared error, mean absolute error and categorical crossentropy. Different types of models such as regression and classification are studied. The robustness of this optimizer against the noisy labels is also verified. Many of the empirical results of conducted experiments in this study show that our optimizer works well in practice. It can outperform other state-of-the-art optimizers in terms of accuracy or at least give the same results in addition to the reduction of learning time.

INDEX TERMS Iterative thresholding algorithm based optimizer (Itao), deep learning, optimizer algorithm, neural networks.

I. INTRODUCTION

Artificial neural networks are now pillars of deep learning, which is one of the main technologies of machine learning and artificial intelligence. They are able to perform nonlinear regression. Therefore, they can be adapted to solve inverse problems. An inverse problem arises whenever a physical system must be inferred using measurements [1], and then one must establish whether this problem is well-posed in order to determine its solution. Formally, a well-posed problem must meet Hadamard's three criteria: (i) the problem has a solution; (ii) the solution is unique; and (iii) the solution changes continuously on data and parameters [2]. Often inverse problems are ill-posed, because they are often indeterminate with an infinite number of solutions. Moreover, small changes in the data or parameters can lead to large variations in the accuracy of the solution, which can cause the solution to be unstable and thus not satisfy Hadamard's third criterion. A family of methods commonly used to solve the stability problem is regularization. As introduced in [3], regularization consists in adding prior information to the inferred system during the optimization process, which allows a smoothing of the function by approximating the solution. Neural networks have also benefited from the theory of regularization, which limits the overfitting problem.

Artificial neural networks are based on the backpropagation of the error gradient as a learning algorithm in multi-layer systems [4]. Classically, solving a problem via neural networks is equivalent to using an iterative optimization method. This method is based on the stochastic gradient algorithm which is a differential optimization algorithm [5]. In general, it is intended to minimize a parameterized scalar objective function with respect to its parameters using a chain of partial derivatives propagating from the output to the input of the neural network [6]. In passing, the parameters (weights of the connections between layers of the neural network) are adjusted in according to their contribution to the objective function computed at the output [7]. If the function is written as a sum of functions differentiable with respect to its parameters, the standard method of stochastic gradient descent (SGD) is done. This is due to the generally simple calculation of the first order partial derivatives with respect

The associate editor coordinating the review of this manuscript and approving it for publication was Mauro Gaggero.

to all parameters, and also to the evaluation of the objective function which is done in a random way on a subsample (minibatch) of data points.

In this paper, we propose a new approach to deep neural network optimization based on the iterative thresholding algorithm [8], [9]. This algorithm is an extension of the classical gradient algorithms with regularization [10]. Iterative thresholding algorithm with its other generalizations have proven their feasibility, mainly in many image processing applications (see [11], [12], and [13]). In addition, this algorithm has been used by inverse problems' researchers for partial differential equations [14], [15], [16]. We will use the classical Tikhonov regularization to reformulate the objective function to be minimized. The rest of this paper will be organized as follows. Section II is dedicated to related works. In section III, we present the mathematical foundations behind this approach. Then, we announce and explain the pseudo-code of the proposed algorithm. Experiments and results of this work are carried out in section IV, and concluding remarks are given in section V.

II. RELATED WORKS

SGD, for Stochastic Gradient Descent, is an optimizer algorithm used in neural network to optimize an objective function. It is an iterative method optimization. During the training phase of the neural network, the gradient of the objective function is computed on a single randomly shuffled example of the dataset used and then an update of the W parameters of the network is applied through the error backpropagation algorithm. For this, it is viewed as a stochastic approximation of the gradient descent (GD) where the gradient is computed on the whole dataset. For each iteration, the new value of a parameter w becomes:

$$w \leftarrow w - \eta \nabla_w J(w), \tag{1}$$

where $J(w)$ is a loss function (objective function) and η is a step size also called learning rate. when we compute the gradient on a randomly selected mini-batch examples, during a training iteration, we talk about mini-batch gradient descent and the equation for updating a parameter w becomes:

$$w \leftarrow w - \frac{\eta}{m} \nabla_w J(w), \tag{2}$$

where m is the size of mini-batch. The training becomes slow when the neural network is very large. To speed up the training we use optimizers faster than the SGD.

Polyakc [17] proposed a momentum optimization that takes into account the previous gradients. It computes a velocity v in which the gradient is replicated, the initial velocity is 0. Then it updates the parameters w as follows:

$$\begin{aligned} v &\leftarrow \beta v - \eta \nabla_w J(w), \\ w &\leftarrow w + v. \end{aligned} \tag{3}$$

Here the gradient can be seen as an acceleration factor and not as a velocity factor. To avoid any speed runaway, the hyperparameter β allows to simulate a friction called momentum. Frequently, we use a value close to 0.9. If the gradient remains constant, we can verify that the final speed is equal to this gradient multiplied by $\eta/(1-\beta)$. If $\beta = 0.9$, this speed is -10 times the learning rate multiplied by the gradient [18]. And so we can say in this case that the optimization with momentum can go 10 times faster than the gradient descent to reach the optimum value of the weights that leads basically to a minimum loss. Nesterove [19] proposed a Nesterove Accelerated Gradient, which is a variant of the inertial optimization and is faster than the original version.

There is a set of adaptive learning rate algorithms. Among these algorithms are AdaGrad [20], RMSProp [21] and Adam [6]. Adam for Adaptive Moment Estimation is considered better than other optimizers at least for training deep neural networks. Adam combines the ideas of optimization with inertia and RMSProp. It uses an exponential moving average m of past gradients like optimization with inertia and an exponential moving average v of the squares of past gradients like RMSProp. The steps of the Adam algorithm can be summarized as follows:

$$\begin{aligned} &1.\ m \leftarrow \beta_1 m - (1-\beta_1)\nabla_w J(w), \\ &2.\ v \leftarrow \beta_2 v - (1-\beta_2)\nabla_w J(w) \otimes \nabla_w J(w), \\ &3.\ m \leftarrow m \oslash (1-\beta_1^t), \\ &4.\ v \leftarrow v \oslash (1-\beta_2^t), \\ &5.\ w \leftarrow w + \eta m \oslash \sqrt{v+\epsilon}. \end{aligned} \tag{4}$$

where t is the iteration number. It can be seen that steps 1, 2 and 5 are similar to the optimization with momentum and RMSProp. β_1 and β_2 control the exponential decay rates of the moving averages m and v respectively. Steps 3 and 4 allow to dynamize these moments at the beginning of the training because they are initialized to 0. β_1^t and β_2^t become very small after a few tens of iterations while these two steps become negligible. Practically, β_1 is set to 0.9 and β_2 to 0.99. ϵ is a smoothing factor that is frequently set to 10^{-8}, it avoids the division by 0.

As we can notice, all these algorithms act on the gradient of the objective function and the backward system consists of several steps as in the case of Adam. This increases the complexity of the algorithm and then the training time of the neural network. In this paper, we announce Itao, a new optimizer based on iterative thresholding algorithm. It acts on both, the parameter and the error gradient in one step. Many of the experiences in this work show that our optimizer works well in practice. It gives better results in terms of accuracy and it reduces training time. Next section will present the mathematical foundations behind this optimizer.

III. MATHEMATICAL FOUNDATIONS AND PROPOSED ALGORITHM

A. MATHEMATICAL FOUNDATIONS

Let u^δ the observations relative to a dataset and A the evolution operator of a model entirely defined by the parameters w. We try to determine the parameters w from the observations

u^{δ}, it comes down to finding the solution of the equation:

$$A(w) = u^{\delta}. \tag{5}$$

The determination of the parameters w from the observations u^{δ} is called inverse problem.

Neural networks use statistical inversion techniques which are based on inference. For this purpose, we use the observations from a dataset describing the behavior of the function to be inverted A in order to extract the internal regularities. Then we look for an estimate that we note g of the inverse function A^{-1}, which will solve the inverse problem for all observations. This is the task of the learning algorithm [4], it allows to estimate the optimal parameters W of the estimator g_w of the inverse function A^{-1}.

The dataset used during the learning process consists of a number of examples and can be written as follows:

$$D = \{z^i; i = 1, \ldots, n\}, \tag{6}$$

when it is a supervised learning, $z^i = (x^i, y^i)$ where x^i is the input and y^i is the output with $y^i = A(x^i)$. For an unsupervised learning the examples represent only the inputs of A and thus $z^i = x^i$, here the role of learning is to provide a representation of the output space from a statistical or topological view's point(for example clustering, features extraction, etc ...). These examples are used to find the best estimator g_w and then the best estimator A_w of the function A.

The estimator A_w is a regression model with parameters W. For an input x, the value predicted by the estimator is $y(w, x) = A_w(x)$.

The problem for the estimation of optimum parameters W is an inverse problem, that we can reformulate as follows:

Inverse Source Problem (ISP). Determine the matrix w such that $A(w) = u^{\delta}$, where $u^{\delta} \in \mathbf{R}^n$ is the real data and the operator A is defined as follows:

$$\begin{cases} A : \mathcal{M}_{m,n}(\mathbf{R}) \longrightarrow \mathbf{R}^n \\ \qquad w \qquad\quad \longrightarrow y(w). \end{cases} \tag{7}$$

Here $y(w)$ is the predicted output. We treat the ISP by interpreting its solution as a minimizer of the following problem:

$$\text{find } w^{\star} \in \mathcal{U} \text{ such that } E(w^{\star}) = \min_{w \in \mathcal{U}} E(w), \tag{8}$$

where E is the cost function whose result will decrease as the values predicted by the model get closer to the real values (observations), and $\mathcal{U}$ is the set of unknown admissible parameters w defined in the following way:

$$\mathcal{U} := \{h \in \mathcal{M}_{m,n}(\mathbf{R}) : \|h\|_{\mathcal{M}_{m,n}(\mathbf{R})} \leq r, r > 0\}. \tag{9}$$

Evidently, the set $\mathcal{U}$ is a bounded, closed, and convex subset of $\mathcal{M}_{m,n}(\mathbf{R})$.

The problem (8) is ill-posed in the sense of Hadamard. Even if there is a unique solution w_{true}, it may be unstable: a small variation ε on the observation $u^{\delta}{}_{\varepsilon} = u^{\delta} + \varepsilon$ can cause large variations in the restitution of w. Thus, the simple minimization of $\left\|y(w) - u^{\delta}{}_{\varepsilon}\right\|$ does not always ensure a good estimation of the solution w, to solve it we propose an approach based on a classical Tikhonov regularization technique in order to guarantee numerical stability of the computational procedure. The problem thus consists in minimizing a functional of the form:

$$J(w) = E(w) + \frac{1}{2}\lambda \|w\|^2_{\mathcal{M}_{m,n}(\mathbf{R})}, \tag{10}$$

here, λ being a small positive regularizing coefficient that provides extra convexity to the functional J.

Next, we reformulate the minimization problem (8) to reconstruct the parameters w according to the least squares error with the Tikhonov regularization as follows:

$$\min_{w \in \mathcal{M}_{m,n}(\mathbf{R})} J(w), J(w) := \frac{1}{2}\left\|y(w) - u^{\delta}\right\|^2_2 + \frac{1}{2}\lambda \|w\|^2_{\mathcal{M}_{m,n}(\mathbf{R})}, \tag{11}$$

To solve nonlinear optimizations, almost all iterative methods use the derivatives of the concerned objective functional. The gradient of J is:

$$\partial_w J = \partial_w E + \lambda w. \tag{12}$$

Proposition 1: $w^{\star} \in \mathcal{U}$ is a minimizer of the functional $J(w)$ only if it satisfies the equation:

$$\partial_{w^{\star}} J = \partial_{w^{\star}} E + \lambda w^{\star} = 0. \tag{13}$$

To solve the nonlinear equation (13) for $w^{\star}$, we can use the iteration:

$$w_{i+1} = \frac{K}{\lambda + K} w_i - \frac{1}{\lambda + K} \partial_{w_i} E \qquad (i = 0, 1, \ldots) \tag{14}$$

Indeed, $w^{\star}$ is the fixed point of (14) ($w^{\star}$ is the limit of the sequence $(w)_i$).

here $K > 0$ is a tuning parameter, it acts as a weight between the previous step and the iterative update. The iteration (14) coincides with the iterative thresholding algorithm, which can be derived from the minimization problem of a surrogate functional. In their papers, Jiang et al. [10] and Daubechies et al. [22] introduce a surrogate functional that we exploit to discuss the choice of K guaranteeing the convergence. The surrogate functional $J^s(w, h)$ of $J(w)$ can be written as:

$$J^s(w, h) := J(w) + \frac{1}{2}K \|w - h\|^2_{\mathcal{M}_{m,n}(\mathbf{R})} - \frac{1}{2}\|y(w) - y(h)\|^2_2. \tag{15}$$

The positivity of J^s is maintained by:

$$K \|w\|^2_{\mathcal{M}_{m,n}(\mathbf{R})} \geq \|y(w)\|^2_2 \text{ for all } w \in \mathcal{M}_{m,n}(\mathbf{R}). \tag{16}$$

This is achieved by choosing:

$$K \geq \|A\|^2_{op}, \tag{17}$$

here A is a linear evolution operator, it is defined as:

$$\begin{cases} A : \mathcal{M}_{m,n}(\mathbf{R}) \longrightarrow \mathbf{R}^n \\ \qquad w \qquad\quad \longrightarrow y(w). \end{cases} \tag{18}$$

Indeed, $K \geq \|A\|^2_{op}$ implies that:
$K \geq (sup(\frac{\|y(w)\|_2}{\|w\|_{\mathcal{M}_{m,n}(\mathbf{R})}}))^2$ thereafter $K \geq sup(\frac{\|y(w)\|_2^2}{\|w\|^2_{\mathcal{M}_{m,n}(\mathbf{R})}})$.
Therefore $K \geq \frac{\|y(w)\|^2}{\|w\|^2_{\mathcal{M}_{m,n}(\mathbf{R})}}$ and finally,
$K\,\|w\|^2_{\mathcal{M}_{m,n}(\mathbf{R})} \geq \|y(w)\|_2^2$ for all $w \in \mathcal{M}_{m,n}(\mathbf{R})$.
Which demonstrates the equation 16. Then we can write:

$$J(w) = J^s(w,w) \leq J^s(w,h)$$

$J^s(w,h)$ can be regarded as a small perturbation of $J(w)$ when h is close to w. Using the notion of scalar product, we can rewrite $J^s(w,h)$ as below:

$$\begin{aligned}
J^s(w,h) &= J(w) + \frac{1}{2}K\,\|w-h\|^2_{\mathcal{M}_{m,n}(\mathbf{R})} \\
&\quad - \frac{1}{2}\,\|y(w)-y(h)\|_2^2 \\
&= \frac{1}{2}\left\|y(w)-u^\delta\right\|_2^2 + \frac{1}{2}\lambda\,\|w\|^2_{\mathcal{M}_{m,n}(\mathbf{R})} \\
&\quad + \frac{1}{2}K\,\|w-h\|^2_{\mathcal{M}_{m,n}(\mathbf{R})} - \frac{1}{2}\,\|y(w)-y(h)\|_2^2 \\
&= \frac{1}{2}\langle y(w)-u^\delta, y(w)-u^\delta\rangle_2 + \frac{1}{2}\lambda\langle w,w\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} \\
&\quad + \frac{1}{2}K\langle w-h, w-h\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} \\
&\quad - \frac{1}{2}\langle y(w)-y(h), y(w)-y(h)\rangle_2 \\
&= \frac{1}{2}\langle y(w), y(w)\rangle_2 - \langle y(w), u^\delta\rangle_2 + \frac{1}{2}\langle u^\delta, u^\delta\rangle_2 \\
&\quad + \frac{1}{2}\lambda\langle w,w\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} + \frac{1}{2}K\langle w,w\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} \\
&\quad - K\langle w,h\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} + \frac{1}{2}K\langle h,h\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} \\
&\quad - \frac{1}{2}\langle y(w), y(w)\rangle_2 + \langle y(w), y(h)\rangle_2 \\
&\quad - \frac{1}{2}\langle y(h), y(h)\rangle_2 \\
&= \langle y(w), y(h)-u^\delta\rangle_2 + \frac{1}{2}\langle u^\delta, u^\delta\rangle_2 \\
&\quad + \frac{1}{2}(K+\lambda)\langle w,w\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} - K\langle w,h\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} \\
&\quad + \frac{1}{2}K\langle h,h\rangle_{\mathcal{M}_{m,n}(\mathbf{R})} - \frac{1}{2}\langle y(h), y(h)\rangle_2
\end{aligned}$$

It is a quadratic form with regard to w when u^δ and h are fixed. We have:

$$\partial_w J^s(w,h) = \langle \partial_w y(w), y(h)-u^\delta\rangle + (K+\lambda)w - Kh$$

We have also:

$$\begin{aligned}
E(w) &= \frac{1}{2}\left\|y(w)-u^\delta\right\|_2^2 \\
&= \frac{1}{2}\langle y(w)-u^\delta, y(w)-u^\delta\rangle_2 \\
&= \frac{1}{2}\langle y(w), y(w)\rangle_2 - \langle y(w), u^\delta\rangle_2 + \frac{1}{2}\langle u^\delta, u^\delta\rangle_2, \\
\partial_w E(w) &= \langle \partial_w y(w), y(w)\rangle - \langle \partial_w y(w), u^\delta\rangle \\
&= \langle \partial_w y(w), y(w)-u^\delta\rangle.
\end{aligned}$$

$\partial_w J^s(w,h) = 0$ leads to:

$$\underset{w\in\mathcal{M}_{m,n}(\mathbf{R})}{\arg\min}\; J^s(w,h) = \frac{K}{K+\lambda}h - \frac{1}{\lambda+K}\langle \partial_w y(w), y(h)-u^\delta\rangle \tag{19}$$

As a result, the iterative update (14) is equivalent to solving the minimization problem $\min_{w\in\mathcal{M}_{m,n}(\mathbf{R})} J^s(w,h)$ with $h = w_i$. Indeed, we can write:

$$\begin{aligned}
w_{i+1} &= \frac{K}{K+\lambda}w_i - \frac{2}{K+\lambda}\partial_{w_i} y\langle y(w_i) - u^\delta\rangle \\
&= \frac{K}{K+\lambda}w_i - \frac{1}{K+\lambda}\partial_{w_i} E.
\end{aligned} \tag{20}$$

Moreover, Khoramian [9] proves the convergence of the iteration (14) for any bounded linear operator A when K is chosen according to the condition (17). Next, we will state The basic algorithm used for the numerical reconstruction.

B. PROPOSED ALGORITHM

Algorithm 1 presents the pseudo-code of our proposed algorithm. It aims at minimizing the stochastic cost function $E(w)$ which is a scalar and differentiable function w.r.t parameters w. $E_i(w)$ represents the stochastic realization of cost function at step i. The term stochastic comes from the fact that we randomly evaluate a subsample (minibatch) of data points. At each iteration step we need to solve the forward system by calculating $y(w_i)$, then calculate the error $J(w_i)$. In the backward system, we update the w_i parameters using our optimizer. This is described by lines 4 and 5 in the Algorithm 1. In the line 4, g_i denotes the gradient of E_i w.r.t w computed in step i. The parameters w are updated according to (14) as shown in the line 5. The iteration is stopped when w_i converged. Generally the descent algorithm is infinite (does not give a solution in a finite number of iterations), for that we specify a stopping condition. There are several conditions to do this. For example, the iteration is stopped when the variation of the parameters w is not considerable after

Algorithm 1 Stochastic Optimizer Based on Iterative Thresholding Algorithm

```
Require: $\lambda$: A Tikhonov regularization parameter.
Require: $K > 0$: A tuning constant chosen according to (17).
Require: $E(w)$: Objective function with parameter $w$.
Require: $w_0$: Give an initial guess of $w$.
1: $i \leftarrow 0$: Initialize the steps number.
2: while $w_i$ not converged do
3:     $i \leftarrow i + 1$
4:     $g_i \leftarrow \nabla_w E_i(w_{i-1})$    ▷ Get gradients w.r.t objective function at step i.
5:     $w_i \leftarrow (K/(K+\lambda)).w_{i-1} - g_i/(K+\lambda)$    ▷ Update parameters.
6: end while
7: return $w_i$    ▷ Resulting parameters.
```

updating. For this, we can estimate that there is a convergence towards the solution of the minimization problem when $\|w_i - w_{i-1}\|_2 / \|w_{i-1}\|_2 \leq \varepsilon$, where ε is a tolerance chosen at the beginning of the algorithm. In practice, neural networks are trained for a number of iterations after which we stop iterating.

In the coming parts, we will do experiments for different types of neural networks and cost functions. During these experiments, we will be able to compare the results achieved by the proposed optimizer Itao with those obtained by SGD, Adam, RMSProp and AdaGrad.

IV. EXPERIMENTS AND RESULTS

In this section, we experiment with our proposed optimizer in different examples of deep learning algorithms using different classical datasets. In this perspective, we use models including nonlinear regression and logistic regression using both the multi-layer perceptron, deep convolutional and deep residual neural networks. Loss functions such as mean squared error (MSE), mean absolute error (MAE) and categorical crossentropy are used. Although we did not analyze the convergence of our algorithm for the last two cost functions, we empirically found that our optimizer works well for both. Many of these experiments will be subject to performance comparisons between our optimizer and other state-of-the-art optimizers for the same models and datasets.

A. EXPERIMENT: NONLINEAR REGRESSION

In this experiment we use the Auto-MPG [23] dataset as support. This dataset contains physical data related to cars produced between 1970 and 1982 such as power, displacement and weight as well as fuel consumption in miles per gallon. The objective is to predict the fuel consumption of a car from its physical parameters. After a phase of data preprocessing which includes the deletion of rows with forgotten data and the conversion of categorical columns into numerical data, we obtain a dataset with 392 rows and 10 columns. The column named "mpg" representing the number of miles per gallon is the label to predict after training the future model. Since the 9 features use different scales and ranges, we proceed to a normalization. Indeed, one of the reasons for the importance of normalization is that the features are multiplied by the model weights. Thus, the scale of the outputs, the value of the objective function and the scale of the gradients are affected by the scale of the inputs. Also the normalization makes the training much more stable. To normalize features, we use a standard scaler. For a sample x the standard score z is calculated as:

$$z = (x - u)/s, \tag{21}$$

where u and s are the mean and standard deviation respectively of a set samples. this operation is achieved for both training and test features.

Predicting car consumption in miles per gallon falls to a regression problem. To do this, we opt for a deep neural network as used in [24] with:

- The input layer with 9 neurons (9 features);
- Two hidden dense layers, with 64 neurons each, using the ReLu as activation function;
- A linear single-output layer (for mpg label).

As loss function we use two common functions for regression problems: Mean Squared Error (MSE) and Mean Absolute Error (MAE), both with Tikhonov regularization. We experiment with the proposed optimizer Itao as well as Adam, RMSprop, Adagrad and SGD as state-of-the-art optimizers. To evaluate the performance of the proposed model, we use a k-fold cross-validation approach since the number of records in the dataset is not large. Also, there is a trade-off between bias and variance when choosing the k-fold cross-validation [25]. We split this dataset into training set and test set in 80:20 ratio. Then, using 5-fold cross-validation, the training set was randomly divided into five equal sized subsets. Four of them are used as training data, while the fifth is used as validation data to evaluate the model. This operation is repeated five times by shifting the training and the validation subsets. Each time the performance of the model is reported. At the end, we calculate the average of error validation between the predicted values and the true values. We use a mini-batch of size 64 and we train the model for 200 epochs.

When using each optimizer, we opt for the grid search algorithm [26] to find the optimal hyperparameters (regularization parameter: λ, K for Itao, learning rate lr for the other optimizers) giving the best performances of the created model using the cross-validation method. Finally, we take the combination of hyperparameters that gives the lowest validation average error. These selected values are then used to train the entire training dataset, after which we proceed to test the model using the unseen data from the test set. We perform these experiments for the two loss functions mentioned above. Fig. 1 shows the loss curves during the training phase for all optimizers. We notice that for the MSE, the training loss curves converge quickly for all optimizers. The results of this experiment are reported in Table 1 and Table 2 for the two cost functions. For the MSE, the results obtained during the test phase are close to each other for the five optimizers with a slight advantage for Itao. The same remark as for MAE loss function with a slight advantage for Adagrad and Itao. The average errors obtained with MAE are considerably lower than those obtained with MSE because the MAE function is less sensitive to outliers.

Next, we plot the average error during the test phase as a function of the number of epochs. This will allow us to see the behavior with respect to the model learning for the five optimizers when the number of epochs increases. During the training phase, we take as a batch size the total number of the training dataset records. The Fig. 2 shows the results found. Here, we can clearly see that with Itao the average error is monotonic and continues to decrease with each epoch for both MSE and MAE. It becomes significantly lower than those obtained by the other optimizers when the number of epochs increases. The decrease of the average error using Itao

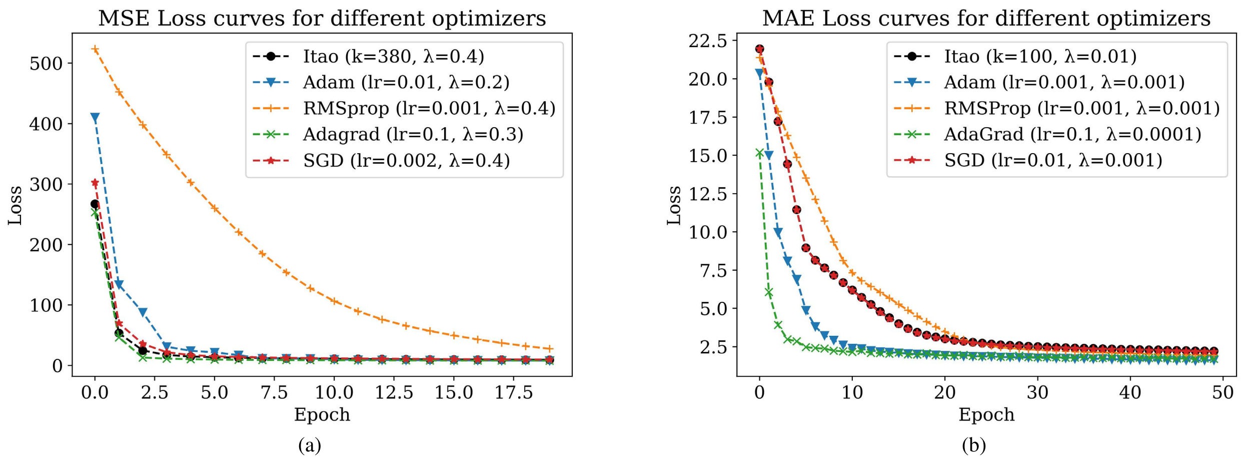


FIGURE 1. **Regression training loss using Auto-MPG dataset. (a) MSE loss function. (b) MAE loss function.**

TABLE 1. **Average error in test phase for MSE loss function using 5-fold cross-validation.**

Optimizer	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average	Test
Itao(K=380.0, λ=0.4)	2.451	2.926	2.649	3.416	2.745	2.837	2.291
Adam(lr=0.01, λ=0.2)	2.425	2.810	2.660	3.350	2.727	2.794	2.357
RMSProp(lr=0.001, λ=0.4)	2.505	2.873	2.661	3.430	2.775	2.849	2.398
AdaGrad(lr=0.1, λ=0.3)	2.511	2.825	2.686	3.454	2.686	2.833	2.451
SGD(lr=0.002, λ=0.4)	2.465	2.882	2.674	3.405	2.725	2.830	2.358

TABLE 2. **Average error in test phase for MAE loss function using 5-fold cross-validation.**

Optimizer	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average	Test
Itao(K=100.0, λ=0.1)	1.757	2.200	1.781	2.324	2.175	2.048	1.738
Adam(lr=0.005, λ=0.001)	1.646	1.956	1.728	2.308	2.053	1.938	1.850
RMSProp(lr=0.001, λ=0.001)	1.849	2.102	1.857	2.412	2.138	2.072	1.960
AdaGrad(lr=0.1, λ=0.0001)	1.748	2.118	1.723	2.321	2.044	1.991	1.709
SGD(lr=0.01, λ=0.1)	1.743	2.184	1.878	2.317	2.142	2.053	1.758

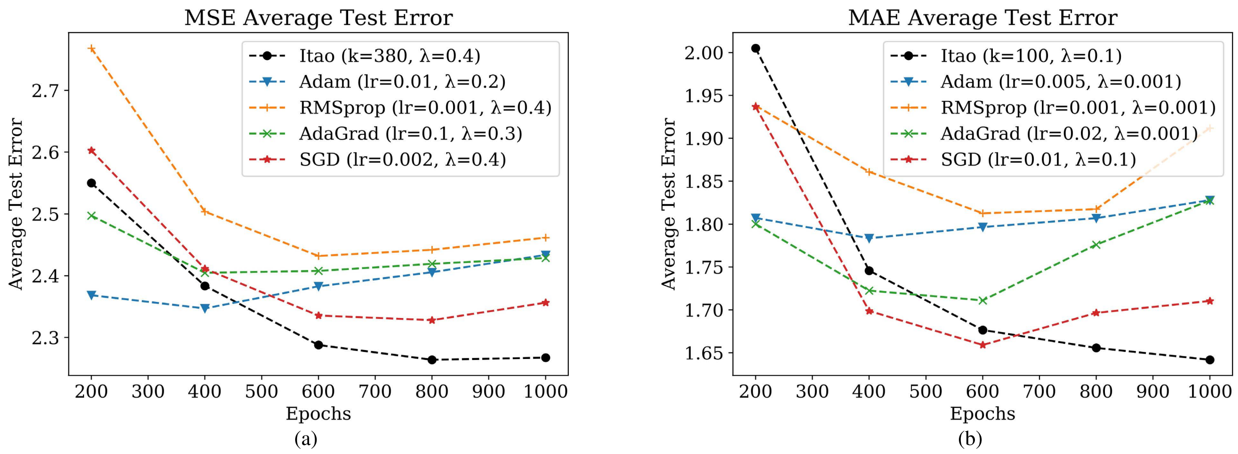


FIGURE 2. **Average test error, using Auto-MPG dataset, as function of epochs' number for all optimizers. (a) with MSE loss function. (b) with MAE loss function.**

implies that it converges in a direct way to the minimal error leading to identify with precision the parameters w^* of (13).

B. EXPERIMENT: LOGISTIC REGRESSION

In this experiment, we use the loss crossentropy function with Tikhonov regularization. The dataset supporting this experiment is MNIST [27] (handwritten Digit Images) which gathers 60,000 images for training and 10,000 images for testing. These grayscale images of 28 pixels on each side belong to 10 classes (0 to 9). We use a model including:

- The input layer with 784 neurons (28 * 28);
- Two hidden dense layers with 256 et 64 neurones respectively, using the ReLu as activation function;
- An output layer with 10 neurons (10 classes), this layer uses softmax as activation function.

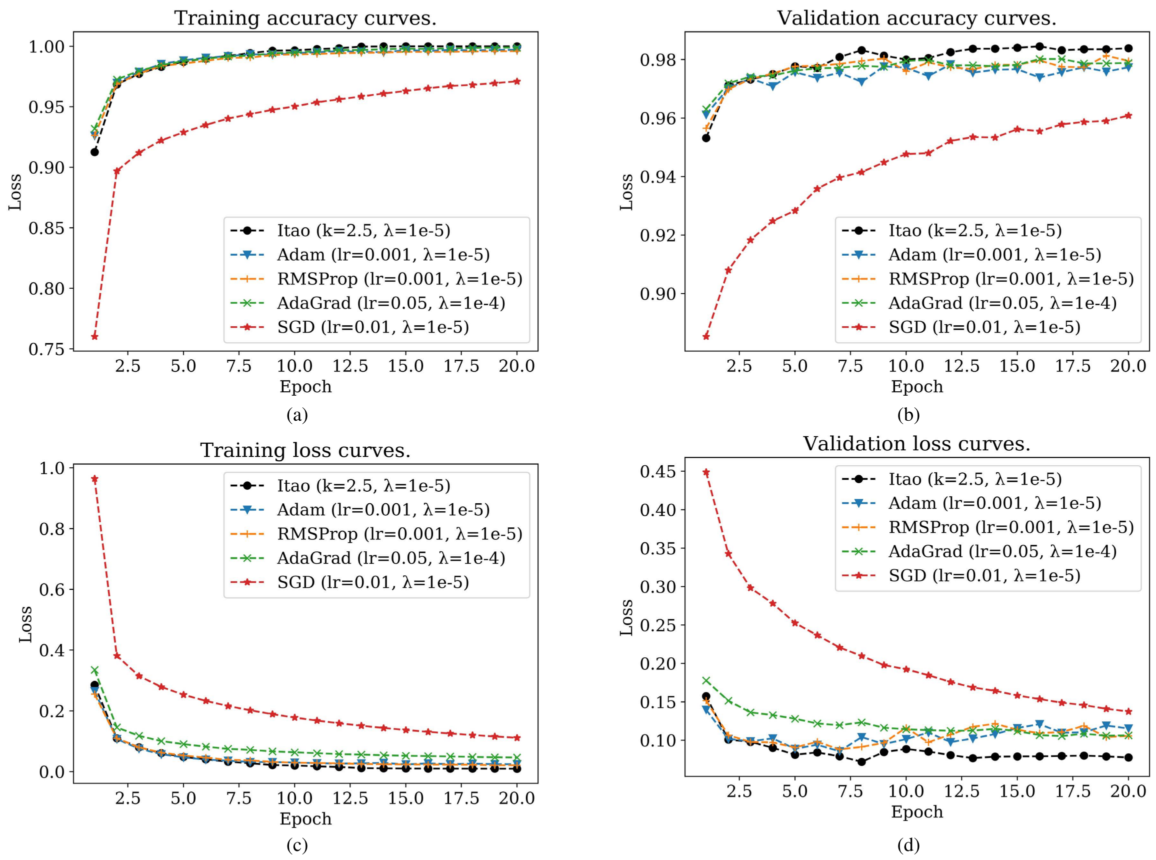


FIGURE 3. Logistic regression training and validation curves on MNIST images. (a) Training accuracy. (b) Validation accuracy. (c) Training loss. (d) Validation loss.

TABLE 3. Performance in test phase and training time for crossentropy loss function using MNIST dataset.

Optimizer	Training time(s)	Test accuracy(%)	Test error
Itao (K=2.5, λ=0.00001)	64.86	98,41	0.076
AdaGrad (lr=0.05, λ=0.0001)	69.80	98,17	0.110
Adam (lr=0.001, λ=0.00001)	69.35	97,96	0.104
RMSProp (lr=0.001, λ=0.00001)	83.86	97,85	0.110
SGD (lr=0.01, λ=0.00001)	62.43	96,62	0.119

In the training phase we take 54,000 images as training set and 6,000 images as validation set. The model is trained for 20 epochs with a batch size of 64. We use the grid serach algorithm to find the hyperparameters giving the best performance of the model created for each of the used optimizers. Fig. 3 shows the accuracy and loss curves during the training and validation phases for the five optimizers: our proposed Itao, Adam, AdaGrad, RMSProp and SGD. We repeat this experiment for 10 times. Table 3 shows the average of the results found for training time, test accuracy and test loss.

Similarly, in this logistic regression experiment with the crossentropy cost function, our optimizer Itao outperforms the other optimizers in terms of accuracy and error when classifying the test images of the MNIST dataset. This also appears in the training and validation curves for both the accuracy and loss. For 20 epochs we arrive in the test phase, on the MNIST dataset with Itao, at an accuracy of 98.41% and an error of 0.076. In terms of accuracy, Itao, Adagrad, Adam and RMSProp are far ahead of SGD which only achieves 96.62%. For the model training time, SGD achieves the lowest time followed by our optimizer Itao. This is because both corresponding algorithms are straightforward and contain fewer steps in the backpropagation phase.

C. EXPERIMENT: CONVOLUTIONAL NEURAL NETWORKS

Today, Convolutional Neural Networks (CNNs) have recognized a great success in computer vision tasks. A convolutional neural network consists of multiple layers of convolution, pooling and nonlinear units. A convolution layer is the main component of CNN architecture, it is based on mathematical operation of convolution. It uses a stack of filters also called kernels, whose parameters can be learned to detect the common features of all the images in the dataset. The principle of CNNs is that lower convolution layers learn low-level features of the input image and which are then grouped together to form more complex features at the high layers. Unlike fully connected layers, the neurons

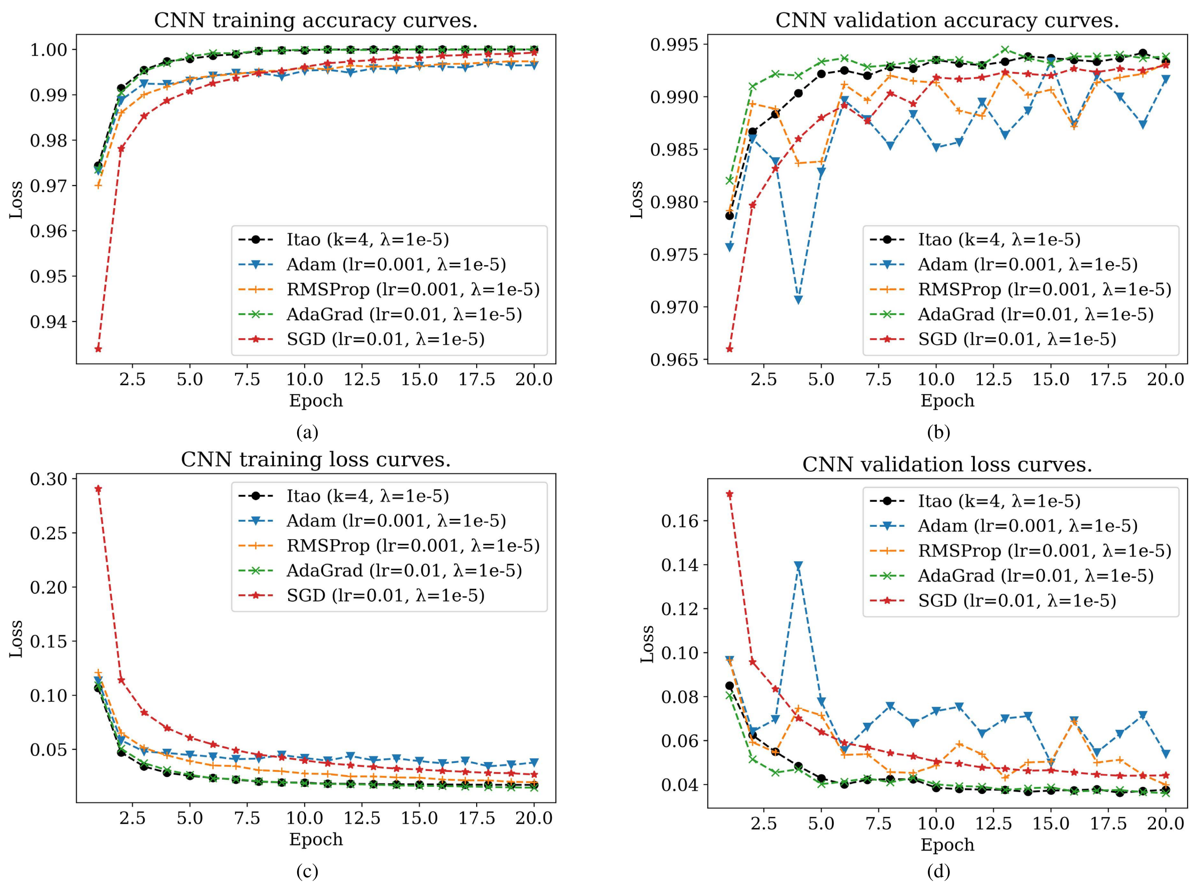


FIGURE 4. Convolutional Neural Network training and validation curves on MNIST images. (a) Training accuracy. (b) Validation accuracy. (c) Training loss. (d) Validation loss.

of a convolution layer are not connected to every neuron of the previous layer, but each of them is connected only to the neurons located in a small rectangle, in the previous layer, of dimensions defined by the filter. Then, all the neurons of a layer share the same parameters' weights. This weight sharing can lead to very different gradients in the different layers [6]. It is therefore often useful in practice to use a small learning rate for the convolution layers when applying the SGD. In this experiment we will show the efficiency of our proposed optimizer based on the iterative thresholding approach. It updates the parameters' weights according to (14) by acting via coefficients on the old values of the parameters' weights themselves and the value of the propagated error's gradient.

1) MNIST CLASSIFICATION USING A TAILORED CNN

As in the previous experiment, we use the MNIST dataset. The architecture of the convolutional neural network is as follows:

- Two cascaded convolution layers, each with 16 filters of 3×3;
- A max pooling layer of stride of 2;
- A convolution layer of 32 filters of 3×3;
- A dense layer of 1024 neurons;
- A dense layer of 128 neurons;
- A dense output layer of 10 neurons with softmax as activation function.

For all layers preceding the output layer, we use LeakyRelu as an activation function. we trained the model for 20 epochs with a batch size of 100. The loss function used is the crossentropy with Tikhonov regularization. As for the previous experiments, we opt for the grid search algorithm to find the hyperparameters' values giving the best performance of the model created for the five optimizers: our proposed Itao, Adam, AdaGrad, RMSProp and SGD. In this experiment we use a GPU Tesla T4 provided by Google Colab [28]. Since some routines of the cuDNN library of NVIDIA GPUs do not guarantee the reproducibility of the results across runs [29], we do the training and the test 10 times then we take the average of results. Fig. 4 shows the accuracy and loss curves during the training and validation phases for all optimizers. Table 4 shows the average results found during the test phase.

Once again, the results of this experiment using a convolutional network clearly show that Itao performs well in terms of accuracy and error in the test phase for the classification of images corresponding to the MNIST dataset. In 20 epochs, we reach with Itao an accuracy of 99.37% for an error of 0.041. Here, we notice that the performances with SGD surpass, with the conditions of this convolutional network, those obtained with Adam and RMSProp. This

TABLE 4. Performances of the convolutional neural network using the MNIST dataset.

Optimizer	Training time(s)	Accuracy (%)	Error
Itao (K=4, λ=0.00001)	89.30	99,37	0.041
AdaGrad (lr=0.01, λ=0.00001)	95.12	99.20	0.048
SGD (lr=0.01, λ=0.00001)	89.01	99.08	0.052
RMSProp (lr=0.001, λ=0.00001)	115.62	98.88	0.059
Adam (lr=0.001, λ=0.00001)	94.97	98.80	0.068

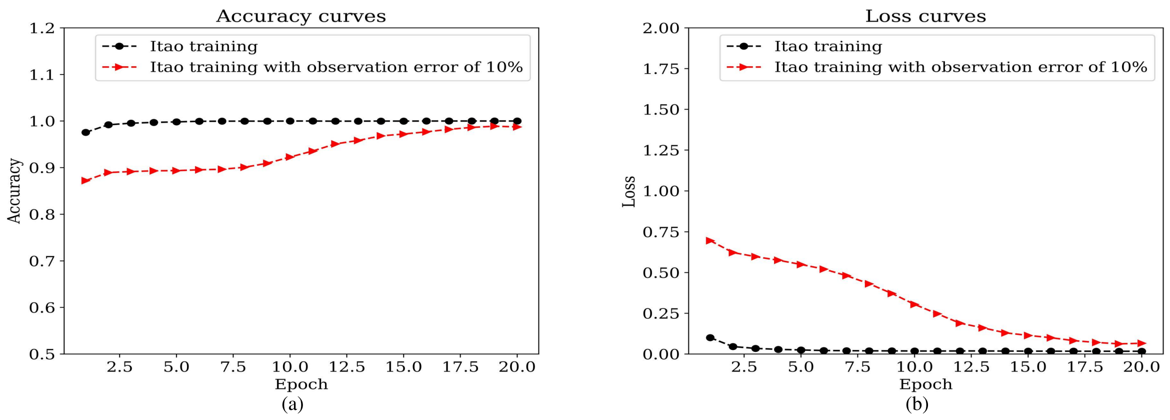


FIGURE 5. Convolutional Neural Network training on MNIST dataset with and without noisy labels. (a) Accuracy curves. (b) Loss curves.

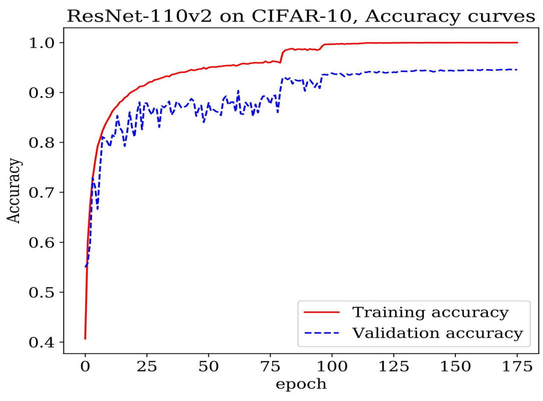
explains that there are some cases where SGD can be better than Adam and RMSprop. Itao's training loss and validation curves still retain the monotonicity of previous experiments, which implies the convergence of the Itao algorithm in the case of the convolutional networks. For the training time, SGD and Itao achieve the lowest time with 89.01s and 89.30s respectively. We can say that when the model becomes more and more complex SGD and Itao give almost the same training time. Because during training we have both forward and backward systems; the time of the forward system is the same. The difference resides in the time of the backward system, Itao takes a little bit longer because it acts on the parameters and their gradients with respect to the cost function during the backpropagation phase. On the contrary, SGD acts only on the gradients. For the other optimizers, we have the same ranking as the previous experiments. RMSprop gives the longest training time.

Next, we will see the robustness of the previous convolutional model using Itao, with the same hyperparameters, against the noisy labels. To do so, we train this model using the 60,000 examples of the MNIST training dataset, then proceed to the evaluation with the 10,000 examples of the MNIST test dataset and this in two trials. In the first one, we use the MNIST training dataset without modification. In the second trial, we take 6,000 examples from the training set, i.e. 10% of the examples in this set, and then we randomly change their classes. With this, we can simulate an observation error of 10%. Fig. 5 shows the loss and accuracy training curves for the two trials. The Table 5 shows the results of the test phase.

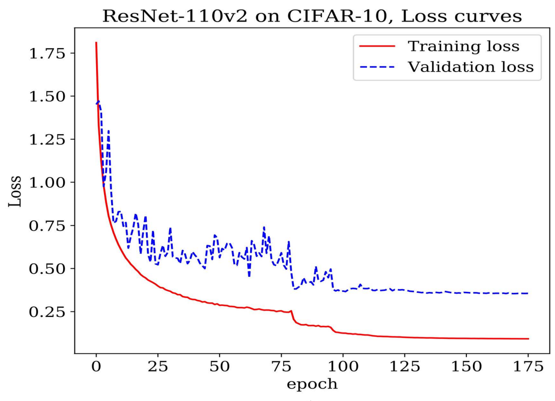
The results obtained show that for 10% of the training examples with noisy labels, the model achieves 94% in the test phase as accuracy against 99.38% in the case of training with original dataset. We have a decrease of only 5.41% in accuracy, moreover the training curves with noisy labels converge towards those with original dataset over the epochs. This may imply that the model used with the Itao optimizer remains robust to the noisy labels.

2) CIFAR-10 AND CIFAR-100 CLASSIFICATION USING DEEP RESIDUAL NEURAL NETWORKS

In this experiment, we use the CIFAR-10 and CIFAR-100 [30] as a datasets, they each contain 60,000 32×32 color images that belong to 10 and 100 classes respectively. Each dataset is divided into training and test datasets with 50,000 and 10,000 images respectively. As a CNN, we use an implementation of Deep Residual Neural Networks version 2 that are a family of common deep CNN architectures [31], [32]. We focus only on the base models with limited data augmentation, we only use translation and horizontal flipping as mentioned in the original paper [33]. Also we train these models with only our optimizer Itao and compare achieved results with those obtained in [33]. For this experiment, we use Google Colab [28] with GPU. On CIFAR-10, we experiment with ResNet-56v2, ResNet-110v2 and ResNet-164v2 models. We train these models on 176 epochs. The Tikhonov regularization parameter λ is fixed at 10^{-5}. The coefficient K is scheduled to be increased. For the first 80 epochs, we start with 2.5, then at each next 16 epochs, we multiply the coefficient K by 2. The Fig. 6 shows training and validation curves for ResNet-110v2 model. The Table 6 groups the test results obtained for the different models and those of the original paper and the work [31], [32]. On CIFAR-100, we experiment with a model based on ResNet-164v2. we train the model on 180 epochs. Similarly, we perform a scheduling for the coefficient K when training



(a)



(b)

FIGURE 6. **ResNet-110v2 on CIFAR-10 using the proposed optimizer Itao. (a) Accuracy curves. (b) Loss curves.**

TABLE 5. **Convolutional Neural Network test phase on MNIST dataset with and without noisy labels.**

Optimizer	**Accuracy (%)**	Error
Itao (K=4, λ=0.00001)	99,38	0.042
Itao (K=4, λ=0.00001) with observation error of 10%	94,00	0.257

TABLE 6. **Classification accuracy(%) on CIFAR-10/100 test set using Deep Residual Networks and Itao optimizer.**

Dataset	Network	Work [32]	Original paper [33]	Proposed
CIFAR-10	ResNet-56v2	93.01	NA	94.23
	ResNet-110v2	93.15	93.63	94.63
	ResNet-164v2	NA	94.54	94.93
CIFAR-100	ResNet-164v2	NA	75.67	76.08

the model. For the first 60 epochs, we set k at 2.5 and we multiply it by 2 each 20 next epochs. The test results are presented at the Table 6. We notice improvements in accuracy for all models studied when using Itao optimizer. The scheduling for the coefficient K by increasing its value is justified by the fact that if its initial value verifies (17), values higher than this one also verify this equation. Indeed, when setting the value of λ; a very large value of K results in a long learning time and does not guarantee the best accuracy. On the other hand, we schedule the value of this coefficient to start with a low value (but checks (17)) then increase it over epochs. This makes the learning faster at the beginning and gives better performances at the end. This can be explained by the fact that when approaching the optimal parameters W of the studied neural network, a smoothness in the learning process must be favored. This is clearly shown in Fig. 6.

V. CONCLUSION

In this paper, we have introduced Itao; a new and efficient optimizer for deep neural networks. It is based on iterative thresholding algorithm which is an extension of the gradient algorithm with regularization. Multi stochastic objective functions are tested for different types of deep learning models. The proposed algorithm is simple, straightforward and easy to implement. We have made a comparison with other state-of-the-art optimizers such as SGD, Adam, AdaGrad and RMSProp. Some experiments realized in this study confirms that Itao can outperform these optimizers or at least give the same performances in terms of accuracy. In addition, the training times, for the models experimented, realized with Itao are close to those achieved by SGD which is the most basic of the optimizers. This is because they both do less steps in the backward system. We report some improvements when classifying CIFAR-10 and CIFAR-100 by ResNet version 2 models compared to the original paper. Also, we verified that Itao remains robust to noisy labels which simulates observation errors. Nevertheless, we hope to bring some enhancements to this algorithm in the future works, especially for the convergence speed when minimizing objective functions.

DATA AVAILABILITY

Source codes and models used to support the findings of this study are available from the corresponding author upon request.

CONFLICTS OF INTEREST

The authors have no conflict of interest to disclose.

FUNDING STATEMENT

There is no funding to declare for this work.

REFERENCES

[1] A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, vol. 2005, doi: 10.1137/1.9780898717921.

[2] J. Hadamard, *Lectures on Cauchy's Problem in Linear Partial Differential Analysis*. New Haven, CT, USA: Yale Univ. Press, 1923.

[3] A. N. Tikhonov, "On the regularization ofill-posed problems," *Dokl. Akad. Nauk SSSR*, vol. 153, no. 1, pp. 49–52, 1963.
[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA, USA: MIT Press, 1986, Ch. 8, pp. 318–362.
[5] S. I. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, nos. 4–5, pp. 185–196, 1993, doi: 10.1016/0925-2312(93)90006-o.
[6] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent., (ICLR) Conf. Track*, 2015, pp. 1–15.
[7] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015, doi: 10.1016/j.neunet.2014.09.003.
[8] M. Fornasier and H. Rauhut, "Iterative thresholding algorithms," *Appl. Comput. Harmon. Anal.*, vol. 25, no. 2, pp. 187–208, Sep. 2008, doi: 10.1016/j.acha.2007.10.005.
[9] S. Khoramian, "An iterative thresholding algorithm for linear inverse problems with multi-constraints and its applications," *Appl. Comput. Harmon. Anal.*, vol. 32, no. 1, pp. 109–130, Jan. 2012, doi: 10.1016/j.acha.2011.03.004.
[10] D. Jiang, Y. Liu, and M. Yamamoto, "Inverse source problem for the hyperbolic equation with a time-dependent principal part," *J. Differ. Equ.*, vol. 262, no. 1, pp. 653–681, Jan. 2017, doi: 10.1016/j.jde.2016.09.036.
[11] I. Daubechies, G. Teschke, and L. Vese, "Iteratively solving linear inverse problems," *Inverse Problems Imag.*, vol. 1, no. 1, pp. 29–46, 2007.
[12] J. M. Bioucas-Dias and M. A. T. Figueiredo, "A new TwIST: Two-step iterative shrinkage/thresholding algorithms for image restoration," *IEEE Trans. Image Process.*, vol. 16, no. 12, pp. 2992–3004, Dec. 2007, doi: 10.1109/TIP.2007.909319.
[13] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm," *Soc. Ind. Appl. Math. J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, 2009.
[14] D. Jiang, H. Feng, and J. Zou, "Overlapping domain decomposition methods for linear inverse problems," *Inverse Problems Imag.*, vol. 9, no. 1, pp. 163–188, 2015, doi: 10.3934/ipi.2015.9.163.
[15] Y. Liu, D. Jiang, and M. Yamamoto, "Inverse source problem for a double hyperbolic equation describing the three-dimensional time cone model," *SIAM J. Appl. Math.*, vol. 75, no. 6, pp. 2610–2635, Jan. 2015, doi: 10.1137/15M1018836.
[16] V. Isakov, *Inverse Problems for Partial Differential Equations*. New York, NY, USA: Springer, 2006.
[17] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Comput. Math. Math. Phys.*, vol. 4, no. 5, pp. 1–17, 1964, doi: 10.1016/0041-5553(64)90137-5.
[18] G. Aurélien, *Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. Sebastopol, CA, USA: O'Reilly, 2019, p. 455.
[19] Y. E. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$," *Soviet Math. Doklady*, vol. 27, no. 2, pp. 372–376, 1983.
[20] J. C. Duchi, H. Elad, and S. Yoram, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011.
[21] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," *COURSERA, Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.
[22] I. Daubechies, M. Defrise, and C. D. Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *J. Commun. Pure Appl. Math.*, vol. 11, no. 57, pp. 1413–1457, 2004, doi: 10.48550/arXiv.math/0307152.
[23] *UCI Machine Learning Repository, Auto MPG Dataset*. Accessed: Mar. 14, 2021. https://archive.ics.uci.edu/ml/datasets/auto+mpg
[24] *Basic Regression: Predict Fuel Efficiency | TensorFlow Core*. Accessed: Aug. 16, 2021. [Online]. Available: https://www.tensorflow.org/tutorials/keras/regression
[25] G. James, D. Witten, T. Hastie, and R. Tibshirani, *Resampling Methods*. New York, NY, USA: Springer, 2013, pp. 175–201, doi: 10.1007/978-1-4614-7138-7_5.
[26] *Grid Search—An Overview | ScienceDirect Topics*. Accessed: Aug. 3, 2021. [Online]. Available: https://www.sciencedirect.com/topics/mathematics/grid-search
[27] Y. LeCun and C. Cortes, "MNIST handwritten digit database," ATT Labs, 2010, vol. 2. [Online]. Available: http://yann.lecun.com/exdb/mnist
[28] *Bienvenue dans Colaboratory—Colaboratory*. Accessed: Sep. 16, 2021. [Online]. Available: https://colab.research.google.com
[29] *Developer Guide: NVIDIA Deep Learning cuDNN Documentation*. Accessed: Jan. 24, 2022. [Online]. Available: https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html#reproducibility
[30] *CIFAR-10 and CIFAR-100 Datasets*. Accessed: Jan. 20, 2022. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html
[31] *CIFAR-10 ResNet Keras*. Accessed: Apr. 16, 2022. [Online]. https://keras.io/zh/examples/cifar10_resnet/
[32] R. Atienza, *Advanced Deep Learning With TensorFlow 2 and Keras_Apply DL, GANs, VAEs, Deep RL, Unsupervised Learning, Object Detection and Segmentation, and More*, 2nd ed. Birmingham, U.K.: Packt Publishing, 2020.
[33] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision ECCV 2016* (Lecture Notes in Computer Science), vol. 9908. Cham, Switzerland: Springer, 2016, pp. 630–645, doi: 10.1007/978-3-319-46493-0_38.

MOHAMED MERROUCHI received the M.S. degree in computer science (distributed information systems) from University Hassan II, Casablanca, Morocco, in 2017. He is currently pursuing the Ph.D. degree with the Faculty of Science and Technology, University Hassan I, Settat, Morocco.

He is a member of the Research Laboratory in Mathematics, Computer Science and Engineering with University Hassan I. He currently works as a Teacher for the Ministry of National Education in Morocco. His research interests include machine learning, deep learning, and big data.

KHALID ATIFI received the M.S. degree in applied mathematics and the Ph.D. degree from the Hassan First University of Settat, Morocco, in 2011 and 2018, respectively.

He is currently a Full Professor of mathematics with the Faculty of Science and Technologies, Cadi Ayyad University, Marrakech, Morocco. His research interests include inverse problem, optimization, control theory, machine learning, and deep learning.

MUSTAPHA SKITTOU received the master's degree in computer science from the Faculty of Sciences and Technics, Hassan First University, Settat, Morocco, in 2016, where he is currently pursuing the Ph.D. degree in computer science. His research interests include big data and artificial intelligence.

TAOUFIQ GADI received the M.S. degree in computer science and the Ph.D. degree from Sidi Mohamed Ben Abdellah, Fes, Morocco, in 1994 and 1999, respectively. He is currently a Full Professor of computer science with the Faculty of Science and Technologies, Hassan First University of Settat, Morocco. He has conducted more than 20 Ph.D. theses and written a 70 of scientific articles in the domain of 3D models analysis, models driving architecture, datamining and database analysis, modeling of complex systems, and machine learning.