J. Nonlinear Funct. Anal. 2022 (2022) 10 https://doi.org/10.23952/jnfa.2022.10

Journal of Nonlinear Functional Analysis
Available online at http://jnfa.mathres.org

RECURRENT NEURAL NETWORK WITH $L_{1/2}$ REGULARIZATION FOR REGRESSION AND MULTICLASS CLASSIFICATION PROBLEMS

LIN LI, QINWEI FAN*, LI ZHOU

School of Science, Xi'an Polytechnic University, Xi'an 710048, China

Abstract. Recurrent neural network (RNN) is introduced to solve the dynamic system problem. In this paper, a new RNN of the gradient method with $L_{1/2}$ regularization learned sequential behavior is presented. $L_{1/2}$ regularization can drive redundant weight vectors of nodes to zero efficiently. The usual $L_{1/2}$ regularization involves a non-smooth absolute value function, which causes the oscillation of the norm of gradient and the error function in the numerical computation. However, by smoothing techniques, those drawbacks can be well addressed. Simulation results of regression and multiclass classification problems demonstrate that our algorithm has better performance than three other algorithms.

Keywords. Gradient method; $L_{1/2}$ regularization; Recurrent neural networks; Smoothing approximation.

1. INTRODUCTION

An artificial neural network (ANN) comprises interconnected artificial neurons. A simple artificial feed-forward neural network usually has one input layer, one output layer, and one hidden layer; see, e.g., [16, 32, 33]. As an information processing science, ANN has been developed for many years. After experiencing the tortuous road of rising, climax, depression, and revival, it has progressed steadily; see [4, 10, 21]. However, with the development of ANN, researchers discovered that it does not consider the correlation between the data in practical applications, and the output of the network is only related to the input at current [19]. When solving practical problems, there is a lot of sequential data, such as text, voice, and video, etc [2, 13]. These data are often time dependent, and the output of the network is not only related to the input at the current but also related to the previous output [11]. ANN cannot handle this kind of correlation well, because it has no memory ability, it cannot feedback the output of the previous moment to the next moment.

In order to solve this problem, researchers proposed an ANN combined with recurrent connections, which is called the recurrent neural network (RNN); see [15, 30, 31]. RNN can model

*Corresponding author.
E-mail address: ll_1112@163.com (L. Li), qinweifan@xpu.edu.cn (Q. Fan), xpuzhouli@126.com (L. Zhou).
Received December 11, 2021; Accepted March 15, 2022.

sequence data for sequence recognitions and predictions. The structure of the RNN's hidden state serves as the memory of the network [17], and the state of the hidden layer depends on its previous state each time. The RNN has a variety of network structures. Of course, it also supports single output and multi-output; see, e.g., [35, 36]. In the single output mode, the state of the recurrent node is obtained through an output function (such as sigmoid). The output of the recurrent neural network in the multi-output mode depends on its topology. The structure of the RNN used in this paper is a single layer with multi-output recurrent connections [36].

Similar to the ANN, the RNN training also uses the gradient descent method, which is a simple and fast learning algorithm. However, when using the gradient descent algorithm to train the RNN, gradient explosion or gradient disappearance often occurs; see, e.g., [1, 18, 20, 22]. Recently, many researchers focuses on solving this problem, which indeed is a thorny problem in neural network learning. A common way to solve this problem is to add a regularization term to the traditional error function; see [27]. Penalties can drive unnecessary weight to zero, even remove some very small weight, or prevent the weights becoming larger; see [5, 6, 14, 23, 26]. Generally, several regularization methods have been proposed for optimizing neural networks, such as L_p regularization. The error function with L_p regularization is generally defined as follows:

$$Cost = Error + \lambda \phi(w),$$

where Error is the sum of square error function, $\lambda \phi(w) = \lambda \|w\|_p^p$ is the regular term, $\|w\|_p^p = (\sum_i |w_i|^p)$ is the L_p penalty, $\lambda > 0$ is the penalty coefficient, and $||\cdot||$ stands for the Euclidean norm.

The value of p corresponds to different properties of the error function. When $p = 0$, it is called L_0 regularization term (see [7, 39]), which was usually used for variable selection and feature extraction. The L_0 regularization method can obtain the sparsest solution by constraining the number of coefficients, but it also involves solving an NP-hard problem, and these sparse solutions are not easy to calculate. When $p = 1$, it is called the L_1 regularization term (also called LASSO method; see [25, 28]). Compared with L_0, L_1 is easier to solve and only needs to solve the quadratic programming problem, but its sparsity is weaker than the L_0 regularization term [7]. When $p = 2$, it is called L_2 regularization term; see [12, 29]. It can only optimize the network, but cannot make the network sparser. That is, L_2 regularization cannot make redundant weights to zero or remove them. Subsequently, in 2010 and 2012, Xu, Zhang, and Wang [37] and Xu, Chang and Zhang [38] proposed the $p = 1/2$ case (also called the $L_{1/2}$ regularization term), which is a non-convex penalty term with unbiased, sparsity, and oracle properties [8]. At the same time, it can be regarded as a representation of L_p $(0 < p < 1)$. It was proved [38] that the $L_{1/2}$ regularization generates sparser solutions than the L_1 regularization. The $L_{1/2}$ regularization term is defined as follow:

$$Cost = Error + \lambda \|w\|_{\frac{1}{2}}^{\frac{1}{2}},$$

where $\|w\|_{\frac{1}{2}}^{\frac{1}{2}} = \sum_i |w_i|^{1/2}$, and $L_{1/2}$ is one of the most useful tools for solving sparse problems. It has been widely studied by reducing the estimated value of some parameters to zero. At the same time, the $L_{1/2}$ regularization has an absolute value function, which is not differentiable at the origin. In other words, during the training process, this will cause the gradient of the error function to oscillate in numerical experiments, and its convergence analysis is difficult to prove.

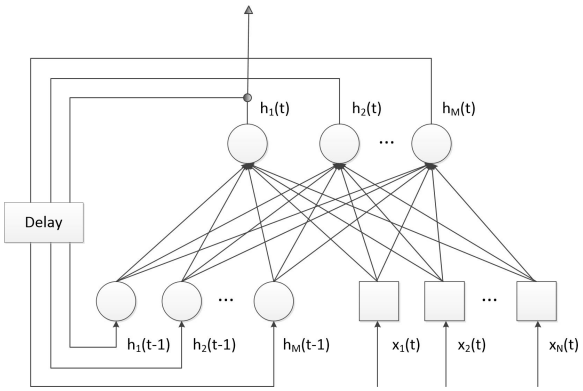


FIGURE 1. Recurrent neural networks

Therefore, the smoothing $L_{1/2}$ regularization was proposed in [9, 34] by replacing the absolute value function with a smooth function to reduce the oscillation phenomenon generated in the numerical experiment.

Inspired by the existing results, this paper proposes an RNN based on $L_{1/2}$ regularization to optimize the network structure to improve its generalization ability. That is, a smoothing technique is introduced to solve the problem that the absolute value function is not differentiable at the origin. More clearly, we summarize the main contributions of this article as follows:

(1) A pruning algorithm based on smoothing $L_{1/2}$ regularization forthe RNN (RNN$L_{1/2}$) is proposed, which can punish the weight matrix of the hidden layer and identify redundant weights, which is achieved by reducing the output redundant weights to zero optimize the effect of network structure.

(2) The $L_{1/2}$ regularization is not differentiable at the origin, which may cause numerical oscillations in the training process. Therefore, we consider using a smoothing function to approximate the absolute value function, which is called the RNN$SL_{1/2}$ and those drawbacks can be well addressed.

(3) Perform numerical experiments to verify the proposed new algorithm. The XOR problem and the classification experiment are given, respectively. The experimental results verify the algorithm can sparse the weight matrix better, and the classification accuracy is also improved.

The rest of this article is organized as follows: In Section 2, the main model structure of the RNN is described. Section 3 describes in detail the RNN gradient algorithm with smooth regularization and gives the algorithm pseudo-code. Section 4 discusses some numerical experiments. Section 5, which is also the last section, provides some conclusions for this paper.

2. THE MODEL ARCHITECTURE OF THE RNN

We consider the structure of the RNN in Figure 1. The expanded view is shown in Figure 2. The input layer has N input nodes. The input of this layer is a vector sequence of elapsed time t $(t = 1,2,\cdots)$, such as $\{x(t),x(t+1),...\}$, where $x(t) = (x_1,x_2,...,x_N)$. The input unit in the fully connected RNN is connected to the hidden unit in the hidden layer, where the connection is defined by the weight matrix $W_I \in \mathbb{R}^N$. The hidden layer has M hidden units $\{h(t-1),h(t),h(t+1),...\}$, where $h(t) = (h_1,h_2,...,h_M)$, which are connected to each other through time and cyclic connections. Using zero elements to initialize hidden units can improve the overall performance and stability of the network. The connection weight is represented by

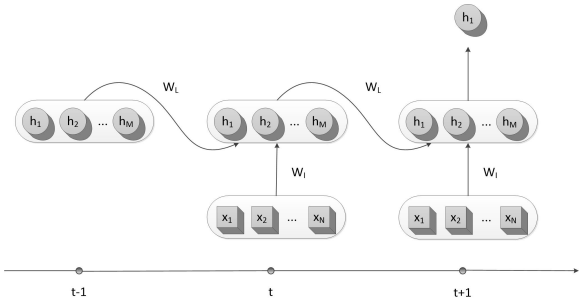


FIGURE 2. The expanded view recurrent neural networks

$W_L \in \mathbb{R}^M$. And the combination weight matrix is

$$W = (W_I, W_L) \in \mathbb{R}^{M\times(N+M)}.$$

It is assumed that the activation function of neurons in the hidden layer and the output neurons is $g : R \to R$. Then, for any vector $a = (a_m) \in \mathbb{R}^M$, define victor function $G : \mathbb{R}^M \to \mathbb{R}^M$:

$$G(a) = (g(a_1), g(a_2), \cdots, g(a_M))^T.$$

Then combine x_t and h_{t-1} as dimensional vector $u_t \in \mathbb{R}^{N+M}$ is

$$u(t) = \begin{pmatrix} x(t) \\ h(t-1) \end{pmatrix}.$$

The input vector s of recurrent layer is $s(t) = (s_1(t), s_2(t), \cdots, s_M(t))^T \in \mathbb{R}^M$. The relationship between the input vector and the output vector at the time of recurrent layer $(t-1)$ is as follows:

$$s(t) = Wu(t) = W_I x(t) + W_L h(t-1), \quad (t = 1, 2, \cdots).$$

The output of the recurrent layer is

$$h(t) = G(s(t)),$$

and the output of the whole network is the first component of $h(t)$

$$h_1(t) = g(s_1(t)), \quad (t = 1, 2, \cdots).$$

3. GRADIENT LEARNING METHOD IN THE RNN WITH SMOOTHING $L_{1/2}$ REGULARIZATION

Suppose that the training sample set is $\{x(t), O(t)\}_{t=1}^{Q}$. Given a matrix $A = (a_{ij}) \in \mathbb{R}^{m\times n}$, *vecA* is defined as *mn* dimension vector as follows:

$$vecA = (a_{11}, a_{21}, \cdots, a_{m1}, a_{12}, a_{22}, \cdots, a_{m2}, \cdots, a_{1n}, a_{2n}, \cdots, a_{mn})^T.$$

Then, we define

$$w = vecW.$$

Define the square error function by

$$\widetilde{E}(w) = \frac{1}{2}\sum_{t=1}^{Q}(O(t) - h_1(t))^2. \tag{3.1}$$

Algorithm 1: RNN

Input: Training set $D = \{x(t), O(t)\}_{t=1}^{Q}$, learning rate η, maximum iterative number $MaxEpochs$, minimum error Err, activation function g, vector e_1;

Output: Determined connection weights $\boldsymbol{w}$;

```
Initialize w randomly in the range of (-1,1), the recurrent layer h(0) = 0;
Epochs = 0;
while Ẽ(w) ⩽ Err or Epochs > MaxEpochs do
    for t = 1 : Q do
        h(t) = g(x(t), h(t − 1), B);
        h_1(t) = h(t)e_1
    end
    for t = 1 : Q do
        Calculate Ẽ(w) based on Eq. (3.1);
        Calculate ΔẼ(w) based on Eq. (3.2);
    end
    Update w based on Eq. (3.3) and Eq. (3.4);
    Epochs = Epochs+1;
end
return w;
```

The gradient of the error function is defined as

$$\widetilde{E}_w(w) = -\sum_{t=1}^{Q}(O(t) - h_1(t))\left(\frac{\partial h(t)}{\partial w}\right)^T e_1, \tag{3.2}$$

where e_1 is the first M-dimensional column vector with 1 and the remaining 0. Then, it can be calculated

$$\frac{\partial h(t)}{\partial w} = G'(s(t))\left((u(t))^T \otimes I_M + W_L \frac{\partial h(t-1)}{\partial w}\right),$$

where I_M is the $M \times M$ unit matrix, and $\otimes$ means the Kronecker product. Because the initial condition $h(0) \equiv 0$, we have

$$\frac{\partial h(0)}{\partial w} = 0.$$

The training of the network is: Starting with an initial value w^0, the weights w^k are updated iteratively by

$$w^{k+1} = w^k + \Delta w^k, \quad (k = 0, 1, 2, \cdots), \tag{3.3}$$

where

$$\Delta w^k = -\eta \widetilde{E}_w(w), \tag{3.4}$$

where η is a positive learning parameter.

The framework of the RNN is shown in Algorithm 1.

On this basis, the $L_{1/2}$ regularization is added to the network weight pruning to optimize the network. So we can obtain the new error function:

$$\begin{aligned} E(w) &= \widetilde{E}(w) + \lambda \|w\|^{\frac{1}{2}}, \\ &= \frac{1}{2}\sum_{t=1}^{Q}(O(t) - h_1(t))^2 + \lambda \|w\|^{\frac{1}{2}}, \end{aligned}$$

where $\lambda \in (0,1)$ is the coefficient of penalty term, and the gradient function of error $E(w)$ is

$$\begin{aligned} E_w(w) &= \widetilde{E}_w(w) + \lambda \frac{sgn(w)}{2\|w\|^{\frac{1}{2}}} \\ &= -\sum_{t=1}^{Q}(O(t) - h_1(t))\left(\frac{\partial h(t)}{\partial w}\right)^T e_1 + \lambda \frac{sgn(w)}{2\|w\|^{\frac{1}{2}}}. \end{aligned}$$

The gradient learning algorithm with penalty terms modifies the value as follows

$$w^{k+1} = w^k + \Delta w^k, \quad (k = 0, 1, 2, \cdots),$$

where

$$\Delta w^k = -\eta \left[\widetilde{E}_w(w) + \lambda \frac{sgn(w)}{2\|w\|^{\frac{1}{2}}}\right].$$

Because the absolute value function is not differentiable at the origin, it is easy to oscillate in the iterative process. To overcome this difficulty, we adopt smoothing technology. If a smooth function is used to approximate the absolute value of the weight in a small neighborhood near the origin, the error function of the algorithm with $L_{1/2}$ regular term is as follows:

$$E(w) = \widetilde{E}(w) + \lambda \|f(w)\|^{\frac{1}{2}}, \tag{3.5}$$

where $f(w)$ is a smooth function that approximates $|w|$.

Generally, we choose a smoothing function of $|x|$ as follow:

$$f(x) = \begin{cases} |x|, & |x| \geq a, \\ -\frac{|x|^4}{8a^3} + \frac{3|x|^2}{4a} + \frac{3a}{8}, & |x| < a. \end{cases}$$

It is not hard to see that

$$f(x) \in [\frac{3}{8}a, +\infty), \quad f'(x) \in [-1, 1], \quad f''(x) \in [0, \frac{3}{2a}],$$

where a is a small positive constant, which is close to zero. The new error equation after adding smoothing is

$$\begin{aligned} E_w(w) &= \widetilde{E}_w(w) + \lambda \frac{f'(w)}{2|f(w)|^{\frac{1}{2}}} \\ &= -\sum_{t=1}^{Q}(O(t) - h_1(t))\left(\frac{\partial h(t)}{\partial w}\right)^T e_1 + \lambda \frac{f'(w)}{2|f(w)|^{\frac{1}{2}}}. \end{aligned} \tag{3.6}$$

Then, we obtain

$$w^{k+1} = w^k + \Delta w^k, \quad (k = 0, 1, 2, \cdots), \tag{3.7}$$

where

$$\Delta w^k = -\eta \left[\widetilde{E}_w(w) + \lambda \frac{f'(w)}{2|f(w)|^{\frac{1}{2}}} \right]. \tag{3.8}$$

The framework of the smoothing $L_{1/2}$ regularizer with recurrent neural network (RNN$SL_{1/2}$) is shown in Algorithm 2.

Algorithm 2: RNN$SL_{1/2}$

Input: Training set $D = \{x(t), O(t)\}_{t=1}^{Q}$, learning rate η, maximum iterative number $MaxEpochs$, minimum error Err, activation function g, penalty rate λ, smoothing parameter a, vector e_1;

Output: Determined connection weights $\boldsymbol{w}$;

```
Initialize w randomly in the range of (-1,1), the recurrent layer h(0) = 0;
Epochs = 0;
while E(w) ⩽ Err or Epochs > MaxEpochs do
    for t = 1 : Q do
        h(t) = g(x(t), h(t−1), B);
        h_1(t) = h(t)e_1
    end
    for t = 1 : Q do
        Calculate Ẽ(w) based on Eq. (3.1);
        for i=1:size(w) do
            if w(i) > a then
                w(i) = abs(w(i));
            else
                w(i) = −|w|^4/(8a^3) + 3|w|^2/(4a) + 3a/8;
            end
        end
        Calculate E(w) based on Eq. (3.5);
        Calculate ΔE(w) based on Eq. (3.6);
    end
    Update w based on Eq. (3.7) and Eq. (3.8);
    Epochs = Epochs+1;
end
return w;
```

4. Numerical Simulations

In this section, there are 9 available data sets in the UCI database [3, 24] to evaluate the performance of the L_2 regularizer with recurrent neural network (RNNL_2), the $L_{1/2}$ with recurrent neural network (RNN$L_{1/2}$), the smoothing $L_{1/2}$ regularizer with recurrent neural network (RNN$SL_{1/2}$), and compare with the original RNN. In the first subsection, the results and error graphs of the XOR problem are discussed. In the second subsection, a comparison of the results

of the approximation problem is given. the last subsection provides the comparison results of the classification problems.

4.1. **XOR problems.** In this subsection, the algorithm proposed in the paper is applied to the XOR problem and numerical experiment simulations are given. We choose the network structure as 3 input nodes (including a bias) and 4 recurrent nodes. The sigmoid function $g(t) = \frac{1}{1+e^{-x}}$ is used as the activation function. Given a random initial weight matrix w, the training stops when the number of iteration steps reaches 5000. The experimental results are given in Table 1.

From Figure 3(a), Figure 3(b), and Figure 3(c), we presented the error function curves, the norm of gradient curves, and the norm of weight curves for the RNN, the RNN$L2$, the RNN$L_{1/2}$, and the RNN$SL_{1/2}$ algorithms, respectively. It can be clearly seen from the Figure 3 that the error function is monotonically decreasing, and the gradient norm decreases to zero. It can be seen from the weight norm curve that the size of the weight is effectively controlled.

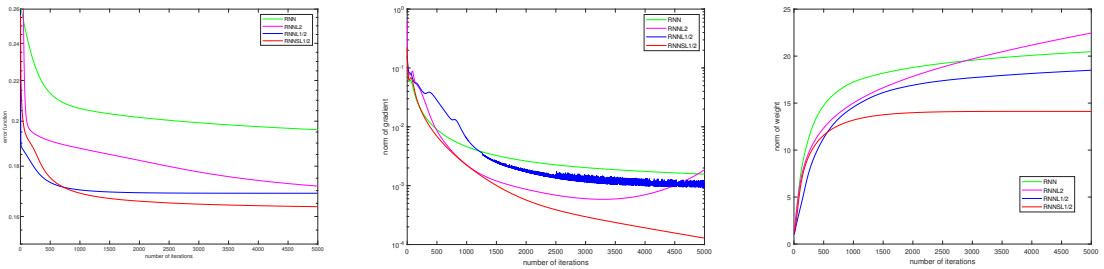
TABLE 1. The training samples and experiment result

t	x_1	x_2	O	RNN	RNNL_2	RNN$L_{1/2}$	RNN$SL_{1/2}$
1	1	1	0	0.0245	0.1493	0.0307	0.0178
2	0	0	1	0.9591	0.9623	0.9899	0.9999
3	1	0	0	0.0267	0.0237	0.0143	0.0083
4	0	1	1	0.9692	0.8978	0.9999	1.0000
5	1	1	1	0.9587	0.9637	1.0000	0.9999
6	0	0	1	0.9589	0.9733	0.9995	0.9999
7	0	1	0	0.4284	0.3283	0.0189	0.0713
8	0	1	1	0.9588	0.9788	0.9975	0.9992
9	1	1	1	0.9877	0.9812	0.9999	0.9998
10	0	0	1	0.9589	0.9673	1.0000	1.0000
11	1	0	0	0.1983	0.0984	0.0165	0.0451
12	0	1	1	0.9745	0.9781	0.9806	0.9899
13	1	1	1	0.9899	0.9543	0.9989	1.0000
14	0	0	1	0.9739	0.9912	0.9999	1.0000
15	0	1	0	0.0472	0.0659	0.0672	0.0173
16	1	0	1	0.9741	0.9876	1.0000	1.0000

4.2. **Function approximation problems.** In this subsection, we consider using the RNN, the RNNL_2, the RNN$L_{1/2}$, and the RNN$SL_{1/2}$ to approximate the function. The expression of the objective function is defined as follows:

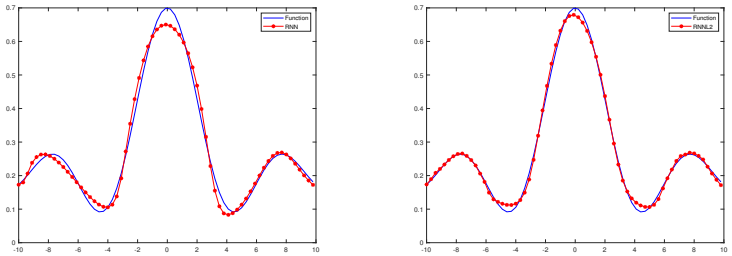
$$y(x) = \frac{\sin(x)}{x}.$$

The training set and testing set $\{x(i), y(i)\}_{i=1}^{T}$ with $T = 67$ data respectively, and $x(i)$ is uniformly distributed on the interval $(-10, 10)$. In order to make the regression problem more realistic, we add uniform noise with $[-0.2, 0.2]$ distribution to the training set, and the test set remains noise-free. For the authenticity of the experimental environment, we choose the learning rate $\eta = 0.06$, the regularization parameter $\lambda = 0.009$, and the smoothing coefficient

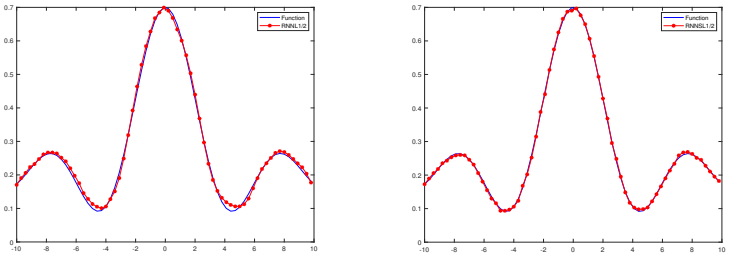


(a) The curve of error function. (b) The curve of norm of gradient. (c) The curve of norm of weight.

FIGURE 3. The performance results based on XOR problem with RNN, RNNL_2, RNN$L_{1/2}$ and RNN$SL_{1/2}$.



(a) Approximation result by the RNN. (b) Approximation result by the RNNL_2.

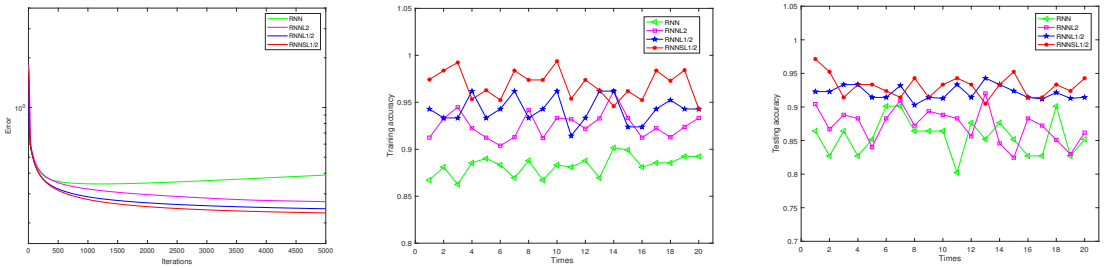


(c) Approximation result by the RNN$L_{1/2}$. (d) Approximation result by the RNN$SL_{1/2}$.

FIGURE 4. Comparison of approximation results for the RNN, RNNL_2, RNN$L_{1/2}$ and RNN$SL_{1/2}$.

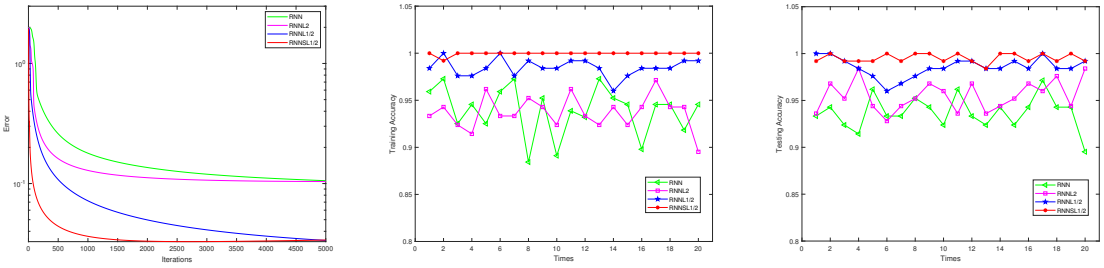
$a = 0.05$. Form Figure 4, we can observe that the approximate effect of the RNN$SL_{1/2}$ on the objective function is better than other algorithms.

4.3. **Classification problems.** To verify the effectiveness of the RNN$L_{1/2}$ and the RNN$SL_{1/2}$ algorithms. Our experiment uses 9 data sets from the UCI machine learning repository. The details of these data are given in Table 2. For each data set, we use 70% of the total data as the training set and 30% as the test set. For the authenticity of the experimental environment, we choose the learning rate $\eta = 0.06$, the regularization parameter $\lambda = 0.009$, and the smoothing coefficient $a = 0.05$. We use the same initial training parameters to build the same network



(a) The error of Iris. (b) The training accuracy of Iris. (c) The testing accuracy of Iris.

FIGURE 5. Comparison of Iris under RNN, RNNL_2, RNN$L_{1/2}$ and RNN$SL_{1/2}$ algorithms.



(a) The error of Wine. (b) The training accuracy of Wine. (c) The testing accuracy of Wine.

FIGURE 6. Comparison of Wine under the RNN, RNNL_2, RNN$L_{1/2}$ and RNN$SL_{1/2}$ algorithms.

structure, obtain the average result of each algorithm under 20 simulation experiments, and then perform a detailed analysis and comparison of its performance.

TABLE 2. Detail description of the classification data sets

Dataset	Instances	Training samples	Testing samples	Attributes	Class
Balance-Scale	625	438	187	4	3
Ecoli	336	235	101	7	8
Glass	214	150	64	10	7
Heart	270	189	81	13	2
Iris	150	105	45	4	3
Seeds	210	146	64	7	3
Sonar	208	146	61	60	2
Speaker	329	230	99	12	6
Wine	178	125	25	13	3

We can see from Table 3 that, compared with other algorithms, the RNN$SL_{1/2}$ can find redundant nodes and has the most significant impact on the sparseness of the weights. Therefore, the RNN$SL_{1/2}$ can better trim the network structure without affecting the test success rate, and can also improve the approximation or classification effect, and overcome the oscillation phenomenon in the learning process.

TABLE 3. Performance comparison for classification problems

Data set	Algorithms	Training Accuracy(%)	Testing Accuracy(%)	Surviving Weights
Balance-Scale	RNN	84.64	82.12	21.00
	RNNL_2	86.17	85.11	21.00
	RNN$L_{1/2}$	88.19	85.23	13.81
	RNN$SL_{1/2}$	92.77	90.68	11.46
Ecoli	RNN	82.50	80.29	120.00
	RNNL_2	81.86	80.61	120.00
	RNN$L_{1/2}$	83.94	81.92	91.32
	RNN$SL_{1/2}$	87.28	84.50	88.92
Glass	RNN	87.50	84.29	90.00
	RNNL_2	86.17	85.11	90.00
	RNN$L_{1/2}$	87.94	85.92	68.30
	RNN$SL_{1/2}$	89.61	87.50	55.72
Heart	RNN	87.71	87.65	30.00
	RNNL_2	87.94	85.39	30.00
	RNN$L_{1/2}$	90.25	86.12	21.34
	RNN$SL_{1/2}$	91.49	91.01	21.03
Iris	RNN	86.40	84.49	21.00
	RNNL_2	90.92	89.76	21.00
	RNN$L_{1/2}$	93.28	90.70	11.93
	RNN$SL_{1/2}$	95.18	93.52	10.79
Seeds	RNN	90.11	89.70	30.00
	RNNL_2	90.49	89.95	30.00
	RNN$L_{1/2}$	93.76	90.65	18.82
	RNN$SL_{1/2}$	94.13	92.57	18.08
Sonar	RNN	93.24	90.80	124.00
	RNNL_2	86.17	85.11	124.00
	RNN$L_{1/2}$	94.14	92.82	60.91
	RNN$SL_{1/2}$	96.57	93.61	58.30
Speaker	RNN	84.96	79.91	108.00
	RNNL_2	85.06	78.39	108.00
	RNN$L_{1/2}$	92.94	85.73	70.32
	RNN$SL_{1/2}$	93.79	87.07	69.79
Wine	RNN	97.94	95.79	48.00
	RNNL_2	96.27	95.11	48.00
	RNN$L_{1/2}$	98.67	96.34	30.15
	RNN$SL_{1/2}$	1.00	99.69	29.08

5. Conclusions

In this paper, we proposed an improved algorithm based on the RNN, which is used to trim the redundant weights of the RNN in the training process. Through the use of smoothing technology, it was shown that the new algorithm can not only solve the oscillation phenomenon generated by the original regularizer but also effectively optimize the network structure and improve efficiency. More importantly, we used the XOR problem, the approximation problem, and the classification problem to prove that the new algorithm can indeed make the weights more sparse so that the network structure is simpler, and the convergence efficiency is faster.

6. Acknowledgment

This work was supported by the Natural Science Basic Research Plan in Shaanxi Province of China (No.2021JM-446) and the 65th China Postdoctoral Science Foundation (No.2019M652837), and the Natural Science Foundation Guidance Project of Liaoning Province (No.2019-ZD-0128).

References

[1] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (1994) 157-166.
[2] R. Bon, H. Carsdot, Advanced methods for time series prediction using recurrent neural networks, Recurrent Neural Networks for Temporal Data Processing, pp. 230-327, 2011. doi: 10.5772/16015.
[3] D. Dheeru, E.K. Taniskidou, UCI machine learning repository, Available online: http://archive.ics.uci.edu/ml, 2017.
[4] J.P. Evans, Classifying artificial neural network architecture, International Neural Network Conference, pp.927-927, 1990.
[5] Q. Fan, J. Peng, H. Li, S. Lin, Convergence of a gradient-based learning algorithm with penalty for ridge polynomial neural networks, IEEE Access 9 (2021) 28742-28752.
[6] Q. Fan, Q. Kang, J.M. Zurada, Convergence analysis for Sigma-Pi-Sigma neural network based on some relaxed conditions, Info. Sci. 585 (2022) 70-88.
[7] Q. Fan, T. Liu, Smoothing L0 regularization for extreme learning machine, Math. Probl. Eng. 2020 (2020) 1-10.
[8] Q. Fan, L. Niu, Q. Kang, Regression and multiclass classification using sparse extreme learning machine via smoothing group L1/2 regularizer, IEEE Access 8 (2020) 191482-191494.
[9] Q. Fan, J.M. Zurada, W. Wu, Convergence of online gradient method for Feedforward Neural Networks with smoothing L1/2 regularization penalty, Neurocomput. 131 (2014) 208-216.
[10] D. Graupe, Principles of Artificial Neural Networks, World Scientific, pp. 320-384, 2013.
[11] A. Graves, S. Fernandez, F. Gomez, J. Schmidhuber, Connectionist temporal classi cation: Labelling unsegmented sequence data with recurrent neural networks, In: Proceedings of the International Conference on Machine Learning, pp. 369-376, 2006.
[12] A. Gupta, S.M. Lam, Weight decay backpropagation for noisy data, Neural Networks 11 (1998) 1127-1138.
[13] S.S. Ho, M. Schofield, N. Wang, Learning incentivization strategy for resource rebalancing in shared services with a budget constraint, J. Appl. Numer. Optim. 3 (2021) 105-114.
[14] Q. Kang, Q. Fan, J.M. Zurada, Deterministic convergence analysis via smoothing group Lasso regularization and adaptive momentum for sigma-pi-sigma neural network, Info. Sci. 553 (2021) 66-82.
[15] C.M. Kuan, K. Hornik, H. White, A convergence result for learning in recurrent neural networks, Neural Comput. 6 (1994) 420-440.
[16] T. Liu, Q. Fan, Q. Kang, L. Niu, Extreme learning machine based on Firefly adaptive flower pollination algorithm optimization, Processes 8 (2020) 1583.

[17] T. Lin, B.G. Horne, C.L. Giles, How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies, Neural Netw. 11 (1998) 861-868.
[18] E. Levin, A recurrent neural network: Limitations and training, Neural Netw. 3 (1990) 641-650.
[19] S. Miyoshi, H.F. Yanai, M. Okada, Associative memory by recurrent neural networks with delay elements, Neural Networks 17 (2004) 55-63.
[20] O. Obst, M. Riedmiller, Taming the reservoir: Feedforward training for recurrent neural networks, International Joint Conference on Neural Networks, pp. 1-7, 2012.
[21] M. Paletta, Artificial neural network for cooperative distributed environments, Artificial Neural Networks Application, 190-210, 2011. doi: 10.5772/15984.
[22] R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, International Conference on Machine Learning, pp. 1310-1318, 2013.
[23] V. Pham, T. Bluche, C. Kermorvant, J. Louradour, Dropout improves recurrent neural networks for handwriting recognition, International Conference on Frontiers in Handwriting Recognition, 2167-6445, 2014.
[24] H. Sakai, C. Liu, M. Nakata, Information dilution: Granule-based information hiding in table data a case of lenses data set in UCI machine learning repository, International Conference on Computing Measurement Control and Sensor Network, 1-7, 2016.
[25] M. Schmidt, Least squares optimization with L1-norm regularization, (2005) 1-5.
[26] R. Setiono, A penalty-function approach for pruning feedforward neural networks, Neural Comput. 9 (1997) 185-204.
[27] N. Srivastava, Improving neural networks with dropout. (2013) 3-79.
[28] R. Tibshirani, Regression shrinkage and selection via the Lasso: A retrospective, J. Royal Stat. Soc. 73 (2011) 273-282.
[29] J. Wang, W. Wu, J.M. Zurada, Computational properties and convergence analysis of BPNN for cyclic and almost cyclic learning with penalty, Neural Netw. 33 (2012) 127-135.
[30] R.J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks, Neural Compu. 1 (1989) 270-280.
[31] R.J. Williams, D. Zipser, Experimental analysis of the real-time recurrent learning algorithm, Connection Sci. 1 (1989) 87-111.
[32] W. Wu, J. Wang, M. Cheng, Z. Li, Convergence analysis of online gradient method for BP neural, Neural Networks 24 (2011) 91-98.
[33] W. Wu, H. Shao, D. Qu, Strong convergence of gradient methods for BP Networks training, International Conference on Neural Networks and Brain, pp. 332-334, 2005.
[34] W. Wu, Q. Fan, J.M. Zurada, J. Wang, D. Yang, Y. Liu, Batch gradient method with smoothing regularization for training of Feedforward Neural Networks, Neural Networks. 50 (2014) 72-78.
[35] D. Xu, Z. Li, W. Wu, X. Ding, D. Qu, Convergence of gradient descent algorithm for a recurrent neuron, Adv. Neural Netw. ISNN 2007 (2007) 117-122.
[36] D. Xu, Z. Li, W. Wu, Convergence of gradient method for a fully recurrent neural network, Soft Comput. 14 (2009) 245-250.
[37] Z.B. Xu, H. Zhang, Y. Wang, X.Y. Chang, L1/2 regularization, Sci. China Info. Sci. 6 (2010) 1159-1169.
[38] Z.B. Xu, X. Chang, F. Xu, H. Zhang, L1/2 regularization: A thresholding representation theory and a fast solver, Neural Netw. 7 (2012) 1013-1027.
[39] H. Zhang, Y. Tang, Online gradient method with smoothing L0 regularization for feedforward neural networks, Neurocomputing 224 (2017) 1-8.