A REPORT ON

IMPROVED SIGN BASED LEARNING ALGORITHM DERIVED BY COMPOSITE NONLINEAR JACOBI-PROCESS

BY

Name of the Student	ID.No.
SHRI LAKSHMI.A	2020H1400217H

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(January, 2021)

A REPORT ON

IMPROVED SIGN BASED LEARNING ALGORITHM DERIVED BY COMPOSITE NONLINEAR JACOBI-PROCESS

BY

Name of the Student	ID.No	Discipline
SHRI LAKSHMI.A	2020H1400217H	M.E,Embedded Systems

Prepared in partial fulfilment of the

Introduction to Artificial Neural Network

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(January, 2021)

ACKNOWLEDGEMENTS

I express my gratitude to the course instructor, **Dr. Harish Dixit** for enabling an opportunity to work on this project. A special mention of his lectures that based my understanding and motivation to completely explore the intricacies of this work.

A special thanks to the authors of the reference materials whose significance owes a major deal in the completion of this project.

ABSTRACT

It is extensively accepted that Rprop algorithm is one of the best training methods of neural network with arbitrary topology. In contrast to that, this project work proposes supervised learning algorithm of multilayer network that employs sign based information of the batch error cost function of Jacobi-Rprop (GJRprop) framework. It is effected by the application of Jacobi--Rprop method with global convergence property. This property promises convergence to local minimum from any initialization point, stability in locating local minimisers over other algorithm and consistency in behavior. The GJRprop descents to local minima by incorporating global convergent measure in Jacobi-Rprop method. The batch error cost function is taken as a measure of Sum of Squared differences between target and output of trained network (SSE).

This project compares the performance of Rprop method, Jacobi- Rprop method and Globally convergent JRprop method with convergence accuracy as the deciding parameter. The convergence accuracy of every method is then compared to prove the proficiency of GJRprop over other methods.

This strategy has been implemented on a benchmark instance and its convergence metric of three learning algorithms namely, Rprop, Jacobi-Rprop and GJRprop methods are compared to validate the proficiency of proposed algorithm.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1. Introduction:

A lot of intelligent applications incorporate neural networking such as digit signature verification, face recognition, biometrics and so on. The accuracy of neural network's output alludes to its learning-scheme. One of the challenges faced in learning phase is the time taken to train the neural network. It is otherwise interpreted as the convergence rate of the error function. The error function is said to converge when predicted and the expected value are nearly same. To meet this challenge, a number of learning algorithms has been developed to converge at the minimal time.

In this area of research, the gradient of error function has been taken to decide the direction of search and hence the weight and bias update. Gradient descent is a class of algorithm that implements the same idea for training the neural network. Backpropagation (BP), which is a popular and base of every other improved training algorithm, update weights and biases of nonlinear neural network using steepest decent method where at each iteration t the steepest direction is employed to minimize error [1].

$$x^{(t+1)} = x^t - \eta \nabla E(x^t); \quad t = 0,1,2 \ldots \ldots \tag{1}$$

Where, E is the batch error measure taken as Sum of Squared errors(SSE), x is the varying parameter , η is a heuristic that is known as learning rate and is defined as $0 < \eta < 1$ to confirm convergence in certain cases and avoid oscillations in the steepest direction.

The correct choice of learning rate inhibits the confluence of network to saddle point or maximum. Different heuristics find its genesis on defining the step size by employing the principle of derivatives, as the step size has an impact on the training speed, quality of the learning process and the results produced by the network[1]. First order and second order learning rules employ dynamic modulation of step size to accelerate training process, mostly implemented in small or medium sized network. The immanent challenge with the first and second order derivative learning-scheme is the convergence to local minima [1]. Though certain local minima provide acceptable convergence however compromise the performance of the network. Hence an ultimate solution to this issue is the use of global optimization particularly for large networks for efficient performance [2].

This project work proposes a new algorithm called GJRprop that promises global convergence of a non-linear network. It is an efficient amelioration of resilient propagation algorithm and Jacobi- bisection algorithm which employs the sign of the error gradient to decide the direction of the weight by corresponding varying the step size. This project work presents comparative results on the convergence of error function of different optimal algorithm to validate the proficiency of the proposed novel GJRprop algorithm.

1.1 Literature survey:

In understanding the algorithms that promise optimizations, mathematical perspective of the training algorithms need to be studied. The simpler case of neural networks is assumed to be linearly separable problem set. The learning scheme for the case of linearity proceeds with an approximation on x^0 to the solution $x = Mx + v$ for some bias v and matrix M [3]. The iterative approximation solutions are generated from a simple expression $x^{t+1} = Mx^t + v\,;\quad t = 0,1,2\ldots$ without much invest in decision process. This simplicity of iterative methods on linear problems set marked its priority over direct methods when the overhead of computations increase in large networks. In cases where accuracy is not demanding, the iteration overhead is reduced by a large amount without comprising the performance [3]. The other benefits of iterative learning are in its less memory storage imposition, especially in cases where partial derivative decision making is employed in the neural network, data causality is not mandatory. This method also eases the issues of instability due to immanent stifling of errors [4].

These advantages in iterative system based its existence in network learning, however, the approach in dealing with nonlinear system embraces numerical methodologies in tracing the roots such as bisection method, newton method and so on which is a one-dimensional search. The delimiting factor of methodologies composed for nonlinear methods is its level of intensive procedure as the network gets larger. This cumbersome aspect initiates the extension of linear iterative methods to nonlinear set through unconstrained optimization of existing nonlinear function. The optimization strategy thus mentioned forms the base for novel algorithms, for this project objective, it is global convergence of nonlinear Jacobi- network [3].

1.1.1 Unconstrained minimization:

The unconstrained minimization of nonlinear function defines the problem statement for which the solutions are generated through computation of different numerical methods. The objective statement for any neural network is to find the minimizer point x^* of a continuous differentiable $f(x_1, x_2, \dots \hat{x_i} \dots, x_n)$ function agreeing the condition

$$\partial_i f(x_1, x_2, \dots \hat{x_i} \dots, x_n) = 0; \;\; i = 1,2, \dots n \tag{2}$$

Where, $\partial_i f$ represent the partial derivative at i^{th} parameter [3].

The most preferred solution to the problem condition mentioned above is composite nonlinear Jacobi- method as the algorithm incorporates parallel computation and is expected for better results in terms of convergence. By parallel computation, it means parallel adjustment of variables towards qualifying equation (2). The metric improvement then follows the following expression:

$$x^{(t+1)} = \; x^t + \tau^t(\hat{x_i} - \; x_i^t); \quad t = 0,1,2 \dots \tag{3}$$

Where, t is the number of epochs steps towards subminimisation, τ^t is the relaxation factor as mentioned in literature [4].

The Jacobi- nonlinear method relaxes the number of iterations as the search is in reference to minima points of the objective function f and the solicited accuracy for a given network. As the accuracy becomes a stringent parameter, more computation pressure is expected. This challenge witnesses the idea of employing adaptive step size or relaxation factor for better convergence output [5].

The ideology of unconstrained minimizations form the foundation of improved algorithm like Resilient Backpropagation (Rprop) and Jacobi- Bisection which is the heuristic tool that hybrids the proposed algorithm of interest.

2. Heuristic approach:

The novel algorithm that is said to promise global convergence in this project hybrids the characteristic feature of Rprop and JRprop. An understanding of learning schemes of these algorithms shall help appreciate the proficiency of GJRprop.

2.1 Rprop algorithm:

As backpropagation uses chain of partial derivatives to update the weight bias, it can get cumbersome for large network (Anastasiadis et al., 2003). To succumb the computation cost, the sign of the partial derivative is used to decide the weight adjustments which is the base idea of Rprop algorithm [7]. As in Rprop, the weight and bias updates are effected by ,

$$x_{ij}^{(t+1)} = \begin{cases} -\Delta_{ij}(t), & if\ \dfrac{\partial E(t)}{\partial x_{ij}} > 0 \\ +\Delta_{ij}(t), & if\ \dfrac{\partial E(t)}{\partial x_{ij}} < 0 \\ 0, & else \end{cases}; \quad t = 0,1,2 \ldots \tag{4}$$

Where, $x_{ij}^{(t+1)}$ is the varying parameter, weight and bias, at the i^{th} neuron and j^{th} input in $(t+1)^{th}$iteration , $\frac{\partial E(t)}{\partial x_{ij}}$ is the partial derivative of error function with respect to the parameter at the t^{th} epoch. Further $\Delta_{ij}(t)$ is the update-value and is defined by:

$$\Delta_{ij}^{(t+1)} = \begin{cases} \eta^{+}.\Delta_{ij}(t), & if\ \dfrac{\partial E(t-1)}{\partial x_{ij}} . \dfrac{\partial E(t)}{\partial x_{ij}} > 0 \\ \eta^{-}.\Delta_{ij}(t), & if\ \dfrac{\partial E(t-1)}{\partial x_{ij}} . \dfrac{\partial E(t)}{\partial x_{ij}} < 0 \\ \Delta_{ij}(t-1), & else \end{cases}; \quad t = 0,1,2 \ldots \tag{5}$$

Where, $\eta^{+}$is the learning rate in the desired convergence direction, η^{-} is the learning when the search direction is unfavorable. These adaptive learning rates are bounded by condition, $0 < \eta^{-} < 1 < \eta^{+}$ [8].

This algorithm owes its validity from the exert of gradient based condition check; assuming the sign change of the gradient hints at traversing the local minima [9]. The gradient of the error function gives information of the search direction through its sign. Since error is obversely preferred, negative sign of the gradient is taken as a favorable output. When the previous iteration and current iteration move in decreasing the error, the step-size is increment by η^+ and vice versa (Anastasiadis et al., 2003) . Whenever the jump effected by learning rate is greater that it missed its local minima that results in sign change, the step size is decreased in the next iteration [10]. To limit the extensive leap of learning rate, the maximum bound is defined at 50. Conventionally, the algorithm parameters are set to the following default values, η^-=0.5 ; $\eta^+ =$ 1.2; $\Delta_{ij}(0)$ = 0.1; $\Delta_{max}= 50$.

2.2 Jacobi- Bisection algorithm:

Though Rprop algorithm works fine, there is a lacunae in backing the significance of how the error is modified upon parametric updation. It does not qualify the procedure it adopts when the sign change is positive in the consecutive iterations, which eventually results in adverse trajectory tracing away from the minima points. To meet the fails of Rprop, another algorithm orients the results by taking into account the magnitude of the derivative function to individually decide the increment or decrement of weight and bias update. The gradient function of the performance index gives information about the end points of the interval. In order to reduce the interval, an interior information of the function is required to proceed through the searching algorithm. This is facilitated from the last two iterations of function value and its gradient at end point itself. The inference that contributes the sway of intervals is obtained from the following conditions:

$$\frac{\partial E(S_1)}{\partial x_{ij}} < 0 \; and \; \frac{\partial E(S_2)}{\partial x_{ij}} > 0, \tag{6}$$

$$\frac{\partial E(S_1)}{\partial x_{ij}} < 0 \; and \; E(S_1) < \; E(S_2), \tag{7}$$

$$\frac{\partial E(S_1)}{\partial x_{ij}} > 0 \; and \; \frac{\partial E(S_2)}{\partial x_{ij}} > 0 \; and \; E(S_1) > E(S_2) \tag{8}$$

Where, S_1 and S_2 are the parameters whose corresponding coordinate is given by a = $\min\{x_{ij}(t-1), x_{ij}(t)\}$ and b = $\max\{x_{ij}(t-1), x_{ij}(t)\}$ [11].

The satisfaction of the above condition qualifies the existence of local minima in the interval [a,b]. Further narrowing of the point in this interval is enabled by employing bisection theorem where the interval is split as [a,m] and [m,b]; $m = \frac{1}{2}(a+b)$. This mode of interval reduction has been modified as follows [11]:

$$x_k^{(t+1)} = x_k^t - h_k\left(\frac{sign\left(\partial_k E(x_k^t)\right)}{2^{(t+1)}}\right); \quad t = 0,1,2 \ldots \tag{9}$$

where, t denotes the number of interval reductions , $\partial_k E(x_k^t)$ denotes the partial derivative function at k^{th} coordinate, the metric at first iteration is $x_k^0 = a_k$; $h_k = sign\left(\partial E(x^0)\right)(b_k - a_k)$ and sign denotes the triple value sign function , which is a segregation of negatives, zeros and positives in a matrix [5]. This modification hints that there is a sure convergence of the metric in the interval, given the above conditions are satisfied. The mathematical background of the bisection methods gives details on the number of epochs f it may take for convergence which is given by the following expression:

$$f = \left[log_2[(b_k - a_k)\varepsilon^{-1}]\right] \tag{10}$$

The definitive number of subminimisation steps predict the feasibility on the efficiency of the network in terms of convergence rate. A theoretical validation of this method is presented in [12].

2.3 Jacobi- Rprop algorithm:

Basing the above mentioned algorithm, Jacobi- Rprop has been proposed that employs bisection interval reduction methodology and the gradient sign influenced update technique of Rprop [4]. In addition, this algorithm further enhances its proficiency by adopting metric updates as in Rprop when the current error is less than the previous iteration's error. When the magnitude of error increases at subsequent iterations, bisection theorem handles the network in deciding the interval with subminimization locations [11]. The intricate significance of this method is in reducing the computational cost it takes in locating the local minima through one-step minima

approximation. A heuristic approach of JRprop is formulated by satisfying the following conditions in update technique which is presented in a form of pseudocode [11].

Here, two metric update equations are employed. When the consecutive errors translates characteristic of convergence by its decrement in magnitude, Rprop methodology of adaptive learning rate aids in the movement of weights and biases in the convergence direction. When the errors seem to diverge, Jacobi- bisection algorithm is employed where g is the parameter that is used to localize the minima in the interval of uncertainty. Since Rrop learning scheme is utilized, the upper and lower bounds of learning rates are taken to be same. Such a search algorithm provides better results than the inherent Rprop and Jacobi- bisection algorithm. From the observation of the pseudocode, the JRprop does not implement third necessary criterion of Jacobi- bisection algorithm in (8). This criterion needs special formulation as it can lead to divergence when not considered properly considered [6].

2.4 Globally convergent JRprop algorithm:

The assured global convergence of JRprop assumes every conditions that pertains to JRprop. The additional modification that contribute to global convergence is the employment of line search methodology satisfying wolfe's rule, Armijo's rule or Goldstein search rule given they are condition specific [4]. The metric adjustment x expression is roughly given by:

$$x^{(t+1)} = x^t + \tau^t d^t; \quad t = 0,1,2 \ldots \tag{11}$$

Where , $\tau^t > 0$ is the update parameter satisfying the conditions of wolfe's rule at t^{th} iteration;

d^t is the search direction at t^{th} iteration and η^t is the adaptive learning rate .

Conventional iterative scheme that qualifies convergence is achieved at the search direction such that $\nabla E^t . d^t < 0$. This expression directs $d^t < -\nabla E^t$ [8]. The equation inherently utilizes the property of learning rate that is tuned dynamically depending on the error function at consecutive iterations. τ^t is the dynamic learning rate that is determined by adopting Wolfe's theorem [13]. This theorem states that, when the learning rate is obtained from the following inequalities,

$$f(x^t + \tau^t d^t) - f(x^t) \leq \sigma_1 \tau^t \nabla f(x^t)^T d^t, \tag{12}$$

$$\nabla f(x^t + \tau^t d^t)^T d^t \geq \sigma_2 \tau^t \nabla f(x^t)^T d^t, \quad (13)$$

Where, $0 < \sigma_1 < \sigma_2 < 1$, results in convergence of the network [14].

The theorem also validates the convergence when cosine of the angle between search direction d^t and gradient function $-\nabla f(x^t)$ is positive such that

$$\lim_{t \to \infty} \|f(x^t)\| = 0 \quad (14)$$

Note that this condition facilitates iterative scheme of learning, as detailed in works of literature by [3], the solutions involving limits give better global convergence results. The global convergence solution should not be mistaken for global minima localization and ideally .

On the basis of the above ken of deduction, the global convergence realization in JRprop learning rule is elaborated from the network update technique given below:

$$x_{ij}^{(t+1)} = x_{ij}^t - \tau^t \, diag\left(\eta_1^t, \eta_{2,}^t \ldots \eta_{i,}^t, \ldots . \eta_{n,}^t\right) sign\left(\nabla E(x^t)\right) \quad (15)$$

Where, search direction $d^t = - diag(\eta_1^t, \eta_{2,}^t \ldots \eta_{i,}^t, \ldots . \eta_{n,}^t) sign(\nabla E(x^t))$ moves in the descent direction, $sign(\nabla E(x^t))$ is a column matrix consisting of elements $\left(\partial_1 E(x^t), \partial_2 E(x^t) \ldots \partial_n E(x^t)\right)$ and η_i^t is the learning rate at the iteration of i^{th} variable obtained by the adaptive learning rule as employed in Rprop [12]. The ideal global convergence is achieved in this methodology by assigning one of the learning to be

$$\eta_i^t = - \frac{\sum_{\substack{j=1 \\ i \neq j}}^{n} \eta_j^t \, \partial_j E(x^t) + \delta}{\partial_i E(x^t)} \quad (16)$$

Where, δ is bounded between 0 and ∞ provided $\partial_i E(x^t) \neq 0$. This theorem has been theoretically proven in literature by [3] that confirms convergence in any algorithm in which it is adopted.

3. Simulation results:

This section presents the simulation result of convergence rate in Rprop, JRprop and GJRprop. Algorithm implementationis achieved in Python. The problem statement taken here is a

nonlinear XOR logic that has been extensively studied by the researchers. Standard neural architectures has been used to deploy this problem. The neural architecture meets the benchmarks as proposed by PROBEN1 website concerned at reducing possible number of biases [15]. PROBEN1 provides systematic rules for designing a neural architecture and consists of problem set that can be used for neural network training of pattern recognition and function approximation. The network architecture is denoted by I-H-O. A sigmoid logistic function is used at the hidden and input layer.

The results provided in a table are convergence accuracy in terms of percentage of iterations (out of 50) and comparison of convergence results of three algorithms in question.

The supporting hyperparameters employed in the algorithm are listed as follows: η^+ = 1.2; η^- = 0.5, $\Delta_{ij}^0 = \eta^0$ = 0.1; Δ^{max} = 50 are taken from the literature [9]. Finally for optimal effect of GJRprop, δ is taken as 10^{-6} and τ^t which is a metric learning rate that is computed from Wolfe's line search algorithm is taken unity for all t.

3.1 XOR logic:

The neural framework employed for this problem is 1-2-1 layers [4]. The graphical results of GJRprop, JRprop and Rprop are presented below. Out of 50 iterations, the algorithm is run with different starting points and achieved an convergence accuracy of 80% unlike JRprop which was 72% and 70% with Rprop algorithm.

Algorithm	Rprop	JRprop	GJRprop
Convergence range	35-40	25-35	1.5-2.5
Convergence accuracy	70%	72%	80%

Table1. *Comparison of algorithm performance in xor problem set*

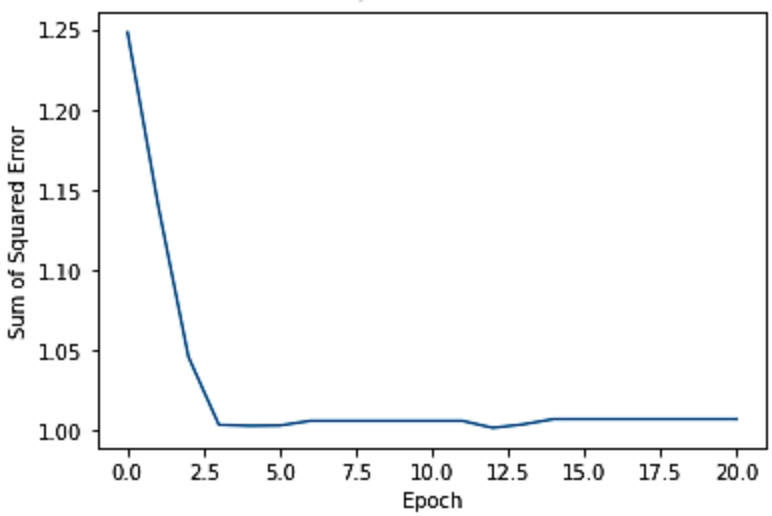

Fig.1 *Performance plot of GJRprop*

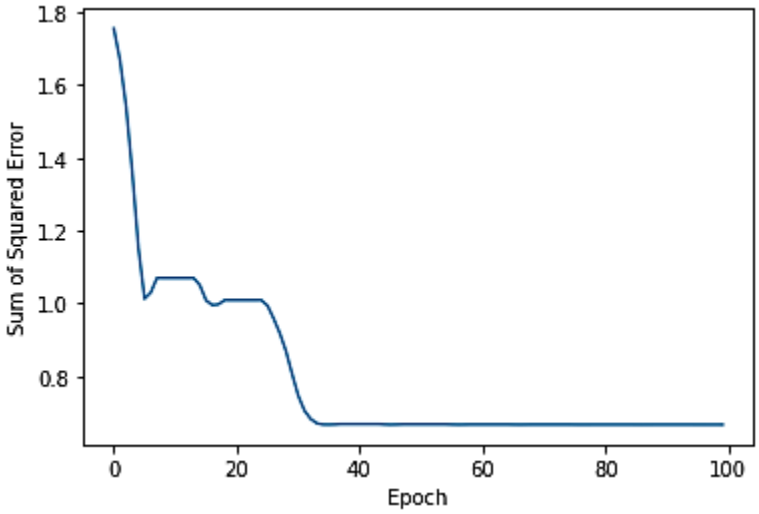

Fig.2 *Performance plot of JRprop*

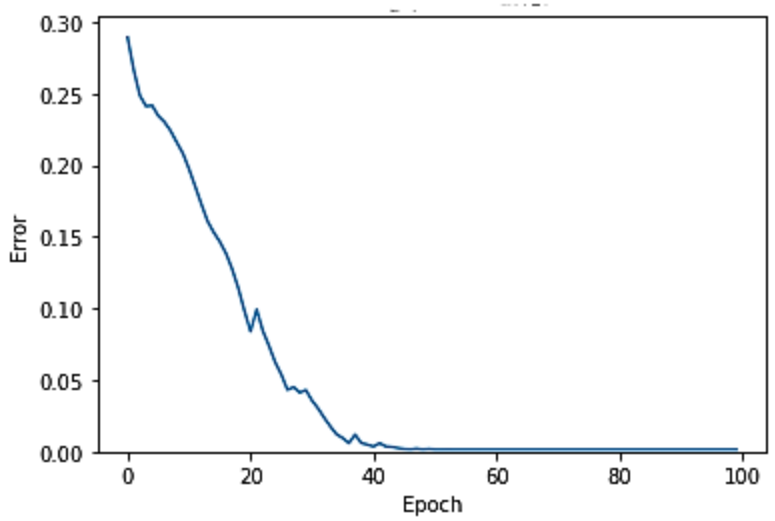

Fig.3 *Performance plot of Rprop*

As it is observed from the performance plot, the convergence of GPRrop converges much better than Rprop and JRprop learning scheme. The converge range in terms of epoch for the three algorithms is given below:

4 Conclusion:

Though Rprop learning scheme has been widely used and listed under one of the optimized algorithms, global convergence is not assured from any starting point. Simulation results show drastic improvement in convergence has been observed for the proposed novel algorithm Globally convergent JRprop. It is also clear that global convergence does not mean global local minima as the accuracy was less than 100%. This algorithm finds its application in medical fields for diagnostic tests, in mathematical streams for determining functions with strong minima and saddle point, and in data communication for N-parity check. Further research could be made to estimate from the containing contributing to global convergence can be applied to any algorithm to ensure optimization. This futuristic tool could then set high benchmarks in researches focused at optimizations. However, further research is required into the behavior of the algorithm to completely define its scope in pattern recognition networks.

5 Appendix:

This section contains the python code for JRprop and GJRprop algorithm.

5.1 GJRprop algorithm

```
import numpy as np
import math as m
import matplotlib.pyplot as plt
import scipy.optimize as s

def transfer(x):
    return 1/(1+np.exp(-x))

def deriv(o):
    f = np.matrix(np.zeros((len(o),len(o))))
    for i in range(len(o)):
        f[i,i] = o[i,0]*(1-o[i,0])
    return f

def sgn(k):
    if k<0:
        return -1.0
    elif k>0:
        return 1.0
    elif k==0:
        return 0
i = np.matrix([[1.,1.],[1.,0.],[0.,1.],[0.,0.]])
o = np.matrix([[0.],[1.],[1.],[0.]])
wt1 = np.matrix(np.random.rand(2,2))
wt2 = np.matrix(np.random.rand(1,2))
b1 = np.matrix(np.random.rand(2,1))
b2 = np.matrix(np.random.rand(1,1))
```

```
y1 = [[],[],[],[]]
y2 = [[],[],[],[]]
e = [[],[],[],[]]

dw2 = [[],[],[],[]]
db2 = [[],[],[],[]]
db1 = [[],[],[],[]]
dw1 = [[],[],[],[]]
q = [[],[],[],[]]
E = []
lr = [np.ones((1,1))*0.1,np.ones((1,2))*0.1,np.ones((2,1))*0.1,np.empty((2,2))*0.1]
q_prev = [np.ones((1,1)),np.ones((1,2)),np.ones((2,1)),np.ones((2,2))]
dell = [np.zeros((1,1)),np.zeros((1,2)),np.zeros((2,1)),np.zeros((2,2))]
dell_prev = [np.zeros((1,1)),np.zeros((1,2)),np.zeros((2,1)),np.zeros((2,2))]

for z in range(50):
    d2_1 = []

    for j in range(len(i)):
        y1[j] = transfer(wt1*np.transpose(i[j]) + b1)
        y2[j] = transfer(wt2*y1[j] + b2)
        e[j] =  o[j] - y2[j]
        db2[j] = -2.0*deriv(y2[j])*e[j]
        dw2[j] = db2[j]*np.transpose(y1[j])
        db1[j] = deriv(y1[j])*np.transpose(wt2)*db2[j]
        dw1[j] = db1[j]*i[j]

    for d in range(4):
        p = [db2,dw2,db1,dw1]
        q[d] = np.sum(p[d],axis=0)/4
```

```
    param = [b2,wt2,b1,wt1]
    E.append(np.sum([e[d1]**2 for d1 in range(4)]))
    if E[z] <= E[z-1]:
        for d2 in range(4):
            for x in range(np.shape(q[d2])[0]):
                for y in range(np.shape(q[d2])[1]):
                    if q_prev[d2][x,y]*q[d2][x,y] > 0:
                        lr[d2][x,y] = min(lr[d2][x,y]*1.2,50)
                        dell[d2][x,y] = sgn(q[d2][x,y])*lr[d2][x,y]
                        q_prev[d2][x,y] = q[d2][x,y]
                        dell_prev[d2][x,y] = dell[d2][x,y]
                    elif q_prev[d2][x,y]*q[d2][x,y] <0:
                        d2_1.append([d2,x,y])
                        lr[d2][x,y] = max(lr[d2][x,y]*0.5,0.000001)
                        q_prev[d2][x,y] = 0
                    elif q_prev[d2][x,y]*q[d2][x,y] == 0:
                        dell[d2][x,y] =  sgn(q[d2][x,y])*lr[d2][x,y]
                        q_prev[d2][x,y] = q[d2][x,y]
                        dell_prev[d2][x,y] = dell[d2][x,y]
        c = 1
        if (q[1][0,0] !=0 or q[2][0,0] !=0 or q[3][0,1] !=0) :
            lr[1][0,0] = ((lr[1][0,1]*q[1][0,1]) + m.pow(10,-6))/q[1][0,0]
            lr[2][0,0] = ((lr[2][1,0]*q[2][1,0]) + m.pow(10,-6))/q[2][0,0]
            lr[3][0,1]  =  ((lr[3][0,0]*q[3][0,0]+  lr[3][1,1]*q[3][1,1]+  lr[3][1,0]*q[3][1,0])  +
m.pow(10,-6))/q[3][0,1]
        for f in range(4):
            for ff in range(np.shape(q[f])[0]):
                for fff in range (np.shape(q[f])[1]):
                    if not [f,ff,fff] in d2_1:
                        param[f][ff,fff] -= sgn(q[f][ff,fff])*lr[f][ff,fff]
    elif E[z] > E[z-1]:
```

```
        for d2 in range(4):
            for x in range(np.shape(q[d2])[0]):
                for y in range(np.shape(q[d2])[1]):
                    param[d2][x,y] += m.pow(0.5,c)*dell_prev[d2][x,y]
                    c += 1

    h = np.matrix([0,1])
    n1 = transfer(wt1*np.transpose(h) + b1)
    n2 = transfer(wt2*n1 + b2)
    print (n2)

    plt.plot(E )
    plt.ylabel('Sum of Squared Error')
    plt.xlabel('Epoch')
    plt.title('Performance Plot of GJRprop')
```

5.2 JRprop algorithm:

```
import numpy as np
import math as m
import matplotlib.pyplot as plt
#from neupy.optimizations import wolfe
def transfer(x):
    return 1/(1+np.exp(-x))

def deriv(o):
    f = np.matrix(np.zeros((len(o),len(o))))
    for i in range(len(o)):
        f[i,i] = o[i,0]*(1-o[i,0])
    return f
```

```
def sgn(k):
    if k<0:
        return -1
    elif k>0:
        return 1
    elif k==0:
        return 0

i = np.matrix([[1.,1.],[1.,0.],[0.,1.],[0.,0.]])
o = np.matrix([[0.],[1.],[1.],[0.]])
wt1 = np.matrix(np.ones((2,2)))
wt2 = np.matrix(np.ones((1,2)))
b1 = np.matrix(np.ones((2,1)))
b2 = np.matrix(np.ones((1,1)))
y1 = [[],[],[],[]]
y2 = [[],[],[],[]]
e = [[],[],[],[]]

dw2 = [[],[],[],[]]
db2 = [[],[],[],[]]
db1 = [[],[],[],[]]
dw1 = [[],[],[],[]]
lr = [[],[],[],[]]
q = [[],[],[],[]]
E = []
lr = [np.ones((1,1))*0.1,np.ones((1,2))*0.1,np.ones((2,1))*0.1,np.empty((2,2))*0.1]
q_prev = [np.ones((1,1)),np.ones((1,2)),np.ones((2,1)),np.ones((2,2))]
dell = [np.ones((1,1)),np.ones((1,2)),np.ones((2,1)),np.ones((2,2))]
dell_prev = [np.ones((1,1)),np.ones((1,2)),np.ones((2,1)),np.ones((2,2))]

for z in range(100):
```

```
for j in range(len(i)):
    y1[j] = transfer(wt1*np.transpose(i[j]) + b1)
    y2[j] = transfer(wt2*y1[j] + b2)
    e[j] =  o[j] - y2[j]
    db2[j] = -2*deriv(y2[j])*e[j]
    dw2[j] = db2[j]*np.transpose(y1[j])
    db1[j] = deriv(y1[j])*np.transpose(wt2)*db2[j]
    dw1[j] = db1[j]*i[j]

for d in range(4):
    p = [db2,dw2,db1,dw1]
    q[d] = np.sum(p[d],axis=0)/4

param = [b2,wt2,b1,wt1]
E.append(np.sum([e[d1]**2 for d1 in range(4)]))
if E[z] <= E[z-1]:
    for d2 in range(4):
        for x in range(np.shape(q[d2])[0]):
            for y in range(np.shape(q[d2])[1]):
                if q_prev[d2][x,y]*q[d2][x,y] > 0:
                    lr[d2][x,y] = min(lr[d2][x,y]*1.2,50)
                    dell[d2][x,y] = -1* sgn(q[d2][x,y])*lr[d2][x,y]
                    param[d2][x,y] += dell[d2][x,y]
                    q_prev[d2][x,y] = q[d2][x,y]
                    dell_prev[d2][x,y] = dell[d2][x,y]
                elif q_prev[d2][x,y]*q[d2][x,y] <0:
                    lr[d2][x,y] = max(lr[d2][x,y]*0.5,0.000001)
                    q_prev[d2][x,y] = 0
                elif q_prev[d2][x,y]*q[d2][x,y] == 0:
                    dell[d2][x,y] = -1* sgn(q[d2][x,y])*lr[d2][x,y]
                    param[d2][x,y] += dell[d2][x,y]
```

```
                q_prev[d2][x,y] = q[d2][x,y]
                dell_prev[d2][x,y] = dell[d2][x,y]
        c = 1
    elif E[z] > E[z-1]:
        for d2 in range(4):
            for x in range(np.shape(q[d2])[0]):
                for y in range(np.shape(q[d2])[1]):
                    param[d2][x,y] += m.pow(0.5,c)*dell_prev[d2][x,y]
                    c += 1

h = np.matrix([0,0])
n1 = transfer(wt1*np.transpose(h) + b1)
n2 = transfer(wt2*n1 + b2)
print (n2)

plt.plot(E)
plt.ylabel('Sum of Squared Error')
plt.xlabel('Epoch')
plt.title('Performance Plot of JRprop')
```

6. **Reference**:

[1] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis, "Effective backpropagation training with variable stepsize," *Neural Networks*, vol. 10, no. 1. pp. 69–82, 1997,

[2] S. Leyffer and A. Mahajan, "Nonlinear constrained optimization: methods and software," *Wiley Encycl. Oper. Resaerch Manag. Sci.*, no. February, 2011,.

[3] M. N. Vrahatis, G. D. Magoulas, and V. P. Plagianakos, "From linear to nonlinear iterative methods," *Appl. Numer. Math.*, vol. 45, no. 1, pp. 59–77, 2003.

[4] A. D. Anastasiadis, G. D. Magoulas, and M. N. Vrahatis, "Sign-based learning schemes for pattern classification," *Pattern Recognit. Lett.*, vol. 26, no. 12, pp. 1926–1936, 2005.

[5] H. Zhang, P. Zhang, and C. J. Hsieh, "RecurJac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications," *arXiv*, no. Figure 1, 2018.

[6] A. D. Anastasiadis, G. D. Magoulas, and M. N. Vrahatis, "An efficient improvement of the Rprop algorithm," *Proc. First Int. Work. Artif. Neural Networks Pattern Recognit. (IAPR 2003), Univ. Florence, Italy*, no. June 2014, p. 197, 2003.

[7] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The Rprop algorithm. In {IEEE} International Conference On Neural Networks," *IEEE Int. Conf. Neural Networks*, vol. 16, no. 3, pp. 586--591, 1993, .

[8] C. Igel and M. Hüsken, "Empirical evaluation of the improved Rprop learning algorithms," *Neurocomputing*, vol. 50, no. September 2002, pp. 105–123, 2003,.

[9] B. Robitaille, B. Marcos, M. Veillette, and G. Payre, "Modified Quasi-Newton methods for training neural networks," *Comput. Chem. Eng.*, vol. 20, no. 9, pp. 1133–1140, 1996,

[10] I. E. Livieris, "Forecasting economy-related data utilizing weight-constrained recurrent neural networks," *Algorithms*, vol. 12, no. 4, 2019,.

[11] A. D. Anastasiadis, G. D. Magoulas, and M. N. Vrahatis, "A Globally Convergent Jacobi-

Bisection Method for Neural Network Training," *Int. Conf. Comput. Methods Sci. Eng. 2004 (ICCMSE 2004)*, no. April, pp. 843–848, 2019.

[12] S. Thrun, "France," *Econ. Outlook*, vol. 19, no. 4, pp. 46–47, 1995.

[13] P. Wolfe, "Convergence condition for ascent methods," vol. 11, no. 2, pp. 226–235, 1969.

[14] G. Piccinini and S. Bahar, "Neural Computation and the Computational Theory of Cognition," *Cogn. Sci.*, vol. 37, no. 3, pp. 453–488, 2013.

[15] L. Prechelt, "A Set of Neural Network Benchmark Problems and Benchmarking Rules," *Tech. rep., Univ. Karlsruhe, Karlsruhe, Ger.*, 2010.