Comparison of methods in accelerating convergence of back-propagation

Yuting Mei

April 2023

1 Background

In machine learning, backpropagation is a widely used algorithm for training feedforward neural networks by updating weight to minimize the error. However, since it used gradient descent as optimization algorithm to find the minimum, it often takes a long time for converging on an acceptable solution[1]. The gradient descent algorithm is especially vulnerable to slow convergence where the error surface consists of long valleys with steep sides and would be sensitive to the changes in certain directions, which is the case when 'ill-conditioned' case is met[6].
There're some methods designed which can construct searches to let solutions get rid of characteristic of graident scheme and outperformed the back-propagation method, such as conjugate gradient algorithm[3], the Modified Back-Propagation Method[7] and some techniques based on gradient method[5].
In this paper, we compared two optimization method: conjugate gradient method with classic gradient descent in fully connected multi neuron network. Newton's method, regula falsi and secant method would be covered. In section 2 we will introduce methodology of neuron network, optimization method. In section 3 we will go through a concrete example to show the weight update procedure.

2 Methodology

2.1 Neural Network(NN)

The neural network of a human is a part of their nervous system and comprises numerous interconnected neurons. Artificial Neural Networks borrow their central theme from the analogy of biological neural networks and are also called "Neural Nets," "parallel distributed processing systems," and "connectionist systems." For a computing system to be called by these names, it should have a labeled directed graph structure where nodes perform simple computations, and each connection conveys a signal from one node to another labeled by a "Connection Strength" or "Weight"[4]. It is a supervised learning algorithm that aims to

minimize the difference between the predicted output of the neural network and the actual output by adjusting the weights of the network.

2.1.1 Notation

In general case, suppose we have X to be an $n \times i$ matrix, for doing weight update procedure, suppose we are going through each row of X, then we have x $= [x_1, x_2, \cdots, x_i]$ with shape of 1 by i, $W = \begin{bmatrix} w_{11} & \cdots & w_{1j} \\ \vdots & & \vdots \\ w_{i1} & \cdots & w_{ij} \end{bmatrix}$, $B = [b_1, \cdots, b_j]$, $Y = [y_1, \cdots, y_j]$ with shape of 1 by j, suppose we have activation function σ, let $z = XW + B$ in each layer, we have $\hat{Y} = \sigma(XW + B) = \sigma(z)$ for multi neuron neural network. Let E to be our loss function, we choose mean squared error as our objective function, we can have the following equations hold by chain rule:

$$\begin{aligned} \frac{\partial E}{\partial Y} &= [\frac{\partial E}{\partial y_1} \cdots \frac{\partial E}{\partial y_j}] \\ &= \frac{2}{n}[y_1 - \hat{y}_1, \cdots y_j - \hat{y}_j] \\ &= \frac{2}{n}(Y - \hat{Y}) \end{aligned}$$

where $\hat{y}$ denoted as precited value.
When activation function is not considered:

$$\begin{aligned} \frac{\partial E}{\partial X} &= \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial X} \\ &= [(\frac{\partial E}{\partial y_1}w_{11} + \cdots + \frac{\partial E}{\partial y_j}w_{1j}) \cdots (\frac{\partial E}{\partial y_1}w_{i1} + \cdots + \frac{\partial E}{\partial y_j}w_{ij})] \\ &= [\frac{\partial E}{\partial y_1} \cdots \frac{\partial E}{\partial y_j}] \begin{bmatrix} w_{11} & \cdots & w_{i1} \\ \vdots & & \vdots \\ w_{1j} & \cdots & w_{ij} \end{bmatrix} \\ &= \frac{\partial E}{\partial Y}W^T \end{aligned}$$

$$\begin{aligned} \frac{\partial E}{\partial W} &= \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial W} \\ &= \begin{bmatrix} \frac{\partial E}{\partial y_1}x_1 & \cdots & \frac{\partial E}{\partial y_j}x_1 \\ \vdots & & \vdots \\ \frac{\partial E}{\partial y_1}x_i & \cdots & \frac{\partial E}{\partial y_j}x_i \end{bmatrix} \\ &= X^T\frac{\partial E}{\partial Y} \end{aligned}$$

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial b_j} + \cdots + \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial b_j}$$
$$= \frac{\partial E}{\partial y_j}$$
$$\Rightarrow \frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial B} = \frac{\partial E}{\partial Y}$$

With consideration of activation function:

$$\frac{\partial E}{\partial X} = [\frac{\partial E}{\partial x_1} \cdots \frac{\partial E}{\partial x_i}]$$
$$= [\frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_1} \cdots \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i}]$$
$$= [\frac{\partial E}{\partial y_1}\sigma'(x_1) \cdots \frac{\partial E}{\partial y_i}\sigma'(x_i)]$$
$$= [\frac{\partial E}{\partial y_1} \cdots \frac{\partial E}{\partial y_i}] \odot [\sigma'(x_1) \cdots \sigma'(x_i)]$$
$$= \frac{\partial E}{\partial Y} \odot \sigma'(X)$$

$$\frac{\partial E}{\partial W^l} = \frac{\partial E}{\partial Y^l} * \frac{\partial Y^l}{\partial W^l} = \frac{\partial E}{\partial Y^l} \odot \sigma^{l-1}(z)^T$$

$$\frac{\partial E}{\partial B^l} = \frac{\partial E}{\partial Y^l} * \frac{\partial Y^l}{\partial B^l} = \frac{\partial E}{\partial Y^l}$$

where l denoted the l^{th} layer.

For newton's method we also need the following:

$$\frac{\partial^2 E}{\partial W^2} = X^T(\frac{\partial^2 E}{\partial Y^2} \odot (\sigma'(z) \odot \sigma'(z))^T)X$$

$$\frac{\partial^2 E}{\partial B^2} = \sum_{i=1}^{j} \frac{\partial^2 E}{\partial y_i^2}\sigma'(z_i)^2$$

2.1.2 Feedforward Propagation

A feedforward neural network can be regarded as a nonlinear mathematical function which transforms a set of input variables into a set of output variables[2].

2.1.3 Backforward Propagation

The backforward propagation algorithm involves computing the derivative of the error with respect to each weight in the network. This is done using the chain rule as mentioned in 2.1.1, which enables us to calculate the derivative of a function with respect to one of its inputs by multiplying the derivatives of the function with respect to the intermediate variables in the calculation chain.

2.2 Optimization Method

2.2.1 Line search method

Typically all line search method used in this paper can also be used as optimization method in one dimension. Line search method helps finding appropriate moving step size in each iteration. Newton's Method The stopping criteria of Newton's Method can be set as any or more from the follows:

1. Maximum number of iterations
2. Convergence of the function: if the difference between the value of the function at two consecutive iterations is less than a predefined tolerance, then the algorithm is considered to have converged. Noted that in Newton's method we also need to pay attention to diverge case. If code can be implemented successfully, then we can observe the frequency of divergence by looking at error at iteration if this criteria is not set.
3. Norm of the gradient.

Regula Falsi In Regula Falsi, the algorithm works by bracketing the root with two initial guesses, and then using linear interpolation to find the root. The stopping criteria of Regula Falsi is as follows:

1. The absolute error between two consecutive iterates is less than a specified tolerance level
2. The relative error between two consecutive iterates is less than a specified tolerance level
3. The maximum number of iterations has been reached.

2.2.2 Secant Method

In Secant Method, the algorithm uses an iterative process to estimate the root of a function. The algorithm begins with two initial guesses and then approximates the function by a line that passes through these two points. Like Newton's method, it would be unstable sometimes. The stopping criteria of Secant Method is as follows:

1. The absolute error between two consecutive iterates is less than a specified tolerance level
2. The relative error between two consecutive iterates is less than a specified tolerance level
3. The maximum number of iterations has been reached
4. The function value is zero or close to zero.

Pseduo code of line search method in NN In the following we have the pseudo code of doing line search, as l denotes layer index, p is the moving direction, t is moving step size, w denotes weight and bias.

Algorithm 1 Line search method with newton's method

Require: Input data X; Weight; Bias; Layers
  Initialize starting step size t
  while not satisfied stopping criteria **do**
    for i in range(number of observation) **do**
      for j in range(number of layers) **do**
        Forward Propagation $(w_{ij} + t_{ij}p_{ij}) \rightarrow$ weight, $\frac{\partial E}{\partial X^l}$ updated
        Backward Propagation $(w_{ij} + t_{ij}p_{ij}) \rightarrow$ gradient updated
        $t_{ij} = t_{ij} - (H(w_{ij} + t_{ij}p_{ij}))^{-1} J(w_{ij} + t_{ij}p_{ij})$
      end for
    end for
  end while
  return t

2.2.3 Gradient Descent

The moving direction in each iteration of gradient descent is orthogonal to each other in order to get the graident to move towards to minimizing objective function. But this method is slow when ill-condition is met, it's also sensitive to the starting point which sometimes easy to get explode.

2.2.4 Conjugate Gradient

In conjugate gradient, the method finds a search direction that is conjugate to the previous search direction, and that also minimizes the function along that direction as negative gradient. The search direction is a linear combination of the negative gradient and the previous search direction, and the coefficients are chosen to satisfy the conjugacy condition. In our context, two vectors are said to be conjugate if their dot product with Hessian matrix is zero. Noted that after n iterations of the conjugate gradient method, where n is the dimension of the weight or bias we want to update in our case, the algorithm needs to restart.

2.2.5 Pseduo Code for Conjugate Gradient

Notice that in the above algorithm, if we use fixed number rather than line search method to find moving step, it would be the gradient descent method.

2.3 Concrete Example

Suppose we have training dataset $X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$, and target variable in training set $Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$.
The following figure shows the workflow of how feedforward propagation of NN doing

Algorithm 2 Line search method with Regula Falsi and Secant Method

Require: Input data X; Weight; Bias; Layers

Initialize starting step size t_0, t_1
if Regula Falsi **then**
while gradient $g_{w0} * g_{w1} < 0$ **do**
regenerate t_1
end while
end if
while not satisfied stopping criteria **do**
for i in range(number of observation) **do**
for j in range(number of layers) **do**
Forward Propagation for t_0, t_1 in $(w_{ij} + t_{ij}p_{ij}) \rightarrow$ weight, $\frac{\partial E}{\partial X^l}$ updated
Backward Propagation for t_0, t_1 in $(w_{ij} + t_{ij}p_{ij}) \rightarrow$ gradient updated
$t_{temp} = \frac{t_0 g_{w1} - t_1 g_{w0}}{g_{w1} - g_{w0}}$
if Regula Falsi **then**
if $g_{w0} * g_{t_{temp}} < 1$ **then**
$g_{w1} = g_{t_{temp}}$
$t_1 = t_{temp}$
end if
else
$g_{w0} = g_{t_{temp}}$
$t_0 = t_{temp}$
end if
else
$t_0 = t_1$
$t_1 = t_temp$
end for
end for
end while
return t

Algorithm 3 Steepest Descent with line search

Require: Input data X; Weight; Bias; Layers
  while not satisfied stopping criteria **do**
    for i in range(number of observation) **do**
      for j in range(number of layers) **do**
        LineSearch $\rightarrow t_{ij}$
        Set $p_{ij} = -g_{ij}$
        Forward Propagation $(w_{ij} + t_{ij}p_{ij}) \rightarrow$ weight, $\frac{\partial E}{\partial X^l}$ updated
        Backward Propagation $(w_{ij} + t_{ij}p_{ij}) \rightarrow$ gradient updated
      end for
    end for
  end while

Algorithm 4 Conjugate gradient with line search

Require: Input data X; Weight; Bias; Layers
  Initialize $k = 0$
  Do steepest descent
  $k+ = 1$
  while not satisfied stopping criteria **do**
    if not meet restarting criteria **then**
      if last$_s tep = steepest$ **then**
        $p_k = -g_k$
        $g_k = g_k$
        $g_{k1} = g_k$
    else
      $p_k = pk$
      $g_k = g_k$
      $g_{k1} = g_{k1}$
    else
      Do steepest Descent
    end if
    $p_k 1 = -g_k + \frac{(g_{k1})^T (g_{k1} - g_k)}{(g_k)^T g_{k-1}}$
    $k+ = 1$
  end while
  return pk1

prediction and weight update of backforward propagation. Suppose two layers is added with $Layer_1(2 \times 3)$ and $Layer_2(3 \times 1)$.
For feedforward propagation, we start from traversing layer l by layer of row i of X as section 2.1.1. With given weight and bias matrix, after doing linear combination with activation function $\sigma(x_i W_l + B_l)$, we stop until layer ends, which give us predicted value as same shape of y_i. We repeat this procedure until all rows of X has been traversed.
We then moving step to backforward propagation by reversing layers and x_i from the update of forward propagation. In general, the element of weight update is consisted from moving direction and moving step size. In gradient descent and steepest descent, they both directly use negative gradient $-\frac{\partial E}{\partial W}, -\frac{\partial E}{\partial B}$ as moving direction except conjugate gradient. In the meantime, different line search method is used to help quantify the exact moving step of weight update in each iteration which would minimize the objective function, in gradient descent it would be the learning rate customized by user.

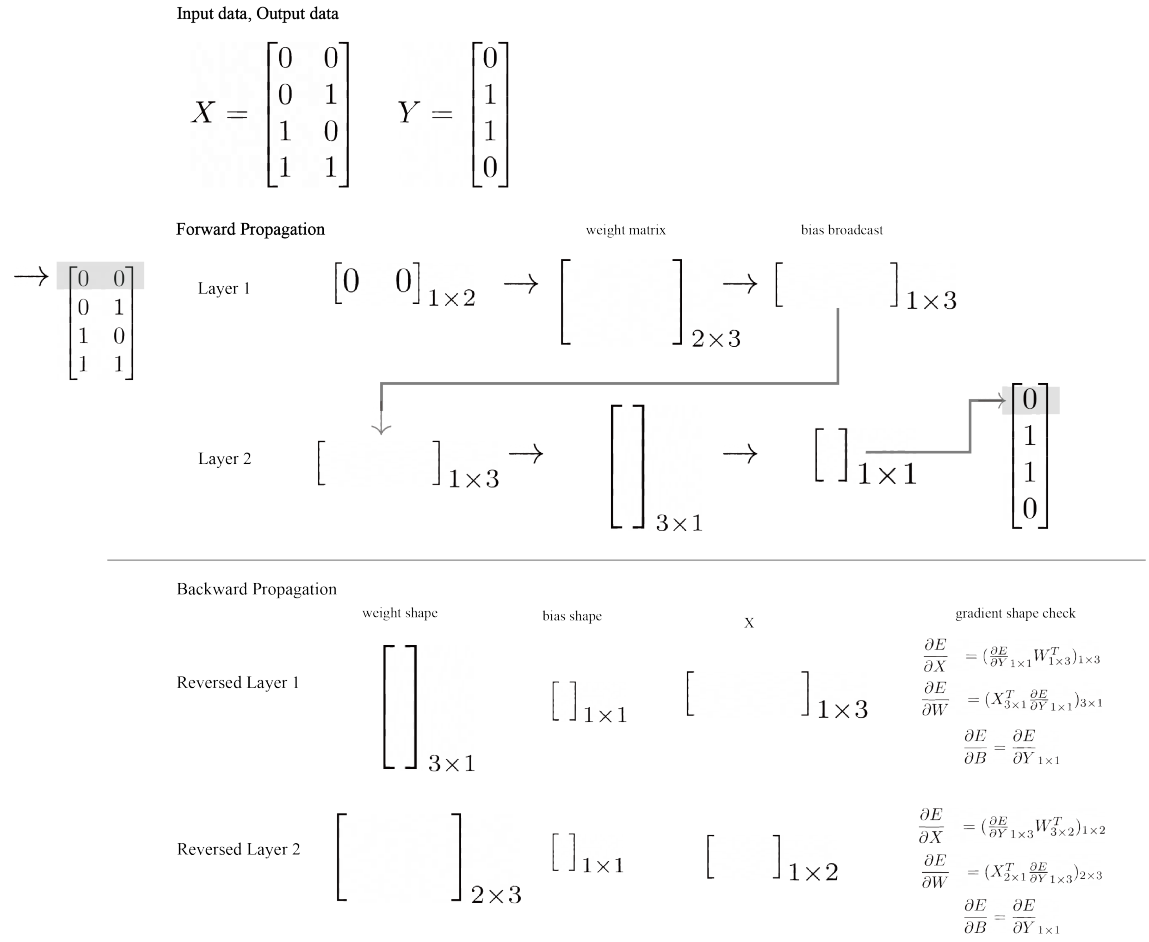

Fig1: procedure of neural network weight update

3 Online Resources

The sources for the work are available via(if code doesn't work, can be treated as pseudo code)

- GitHub Link

References

[1] Hirotugu Akaike. On a successive transformation of probability distribution and its application to the analysis of the optimum gradient method. *Annals of the Institute of Statistical Mathematics*, 11(1):1–16, 1959.

[2] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[3] Christakis Charalambous. Conjugate gradient algorithm for efficient training of artificial neural networks. *IEE Proceedings G (Circuits, Devices and Systems)*, 139(3):301–310, 1992.

[4] AD Dongare, RR Kharde, Amit D Kachare, et al. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194, 2012.

[5] Saad Hikmat Haji and Adnan Mohsin Abdulazeez. Comparison of optimization techniques based on gradient descent algorithm: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology*, 18(4):2715–2743, 2021.

[6] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[7] Thomas P Vogl, JK Mangis, AK Rigler, WT Zink, and DL Alkon. Accelerating the convergence of the back-propagation method. *Biological cybernetics*, 59:257–263, 1988.