Project Report

On

Analysis of an Improved Adagrad Algorithm

Submitted by

PRAYAG JAIN (170001037)

SAKSHAM TANWAR (170001044)

Computer Science and Engineering

3rd year

Under the Guidance of

Dr. Kapil Ahuja

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Spring 2019

Index

1. Introduction

The paper [1] describes an approach to improve convergence for Adagrad convergence function. We compare the convergence of Gradient Descent, Adagrad and the Improved Adagrad provided by the paper [1].

2. Algorithm Analysis

2.1 Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost). And is given by the following formula

$$\theta=\theta_0-\eta\cdot\nabla_\theta J(\theta_0) \tag{1}$$

where θ_0 is the parameter to be update, $J(\theta)$ is the loss function in deep learning, $\nabla_\theta J(\theta_0)$ is the gradient of the parameter θ_0, η is the learning rate.

Equation (1) can be derived using first-order Taylor expansion. Equation (1) both shows that the fastest direction of the function decline is the negative direction along the gradient.

Although the negative direction along the gradient is the fastest direction of the function decline, the unchangeable learning rate is the weakness of many gradient descent optimization algorithms. Because the unchangeable learning rate is not suitable for the sparse data in deep learning.

2.2 Adagrad

To make the learning rate become adaptable in sparse data, Adagrad algorithm was born. Adagrad algorithm divides the learning rate by a gradient-related quantity to achieve this purpose.

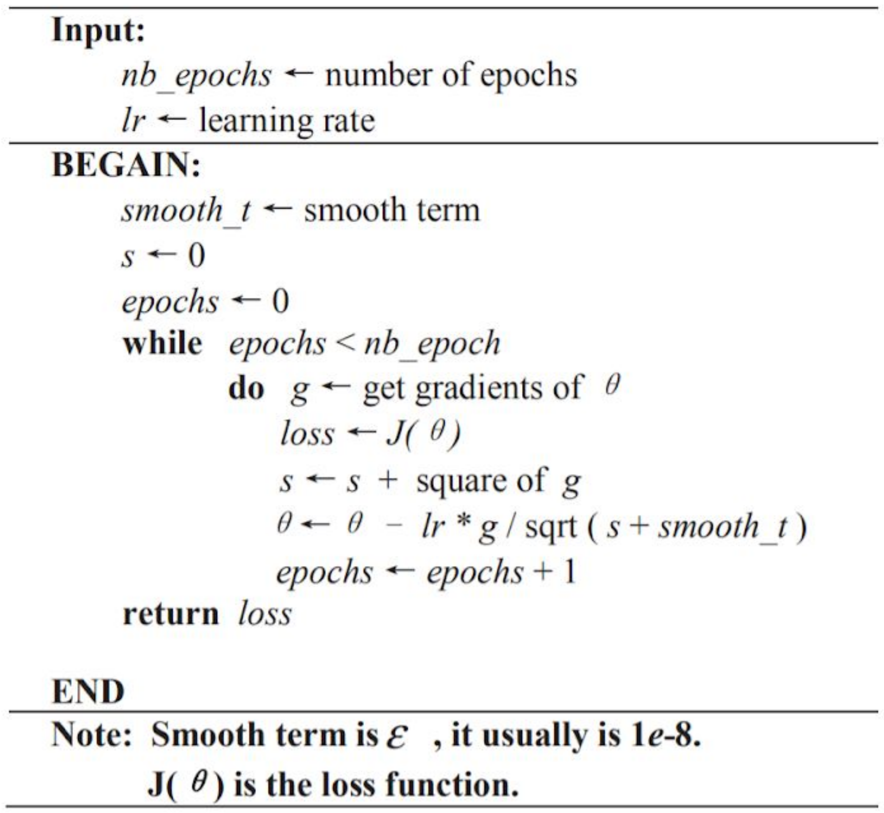
$$s \leftarrow s + \nabla_{\theta} J(\theta_0) \cdot \nabla_{\theta} J(\theta_0) \qquad (2)$$

$$\theta = \theta_0 - \frac{\eta}{\sqrt{s+\varepsilon}} \cdot \nabla_{\theta} J(\theta_0) \qquad (3)$$

where the element of s is the sum of the square of gradients and the initial value is 0, ε is a smoothing term that avoids division by zero (usually is $1e$-8). The other symbols are the same as before.

Adagrad algorithm divides the learning rate by the sum of the square of gradients to achieve the purpose of changing the learning rate during training. This approach increases the adaptability of the learning rate in sparse datasets and performs well in deep learning tasks. Although Adagrad algorithm has many advantages, it still has the weakness of the diminishing learning rate. This weakness makes the convergence process and the validation loss become unsatisfactory in multiple epochs.

Pseudocode for Adagrad



```
Input:
    nb_epochs ← number of epochs
    lr ← learning rate
BEGAIN:
    smooth_t ← smooth term
    s ← 0
    epochs ← 0
    while  epochs < nb_epoch
           do  g ← get gradients of $\theta$
               loss ← J( $\theta$ )
               s ← s + square of g
               $\theta$ ← $\theta$ − lr * g / sqrt ( s + smooth_t )
               epochs ← epochs + 1
    return  loss

END
Note:  Smooth term is $\varepsilon$ , it usually is 1e-8.
       J( $\theta$ ) is the loss function.
```

2.3 Improved Adagrad (As suggested in the paper)

To improve the performance of Adagrad, both improve the stability of the convergence process. The authors propose an improved Adagrad algorithm. The specific algorithm is as follows.

$$s \leftarrow s + \left|\nabla_{\theta} J(\theta_0)\right| \tag{4}$$

$$\theta = \theta_0 - \frac{\eta}{s+\varepsilon} \cdot \nabla_{\theta} J(\theta_0) \tag{5}$$

where $|J(\theta)|$ is the length of the gradient. The other symbols are the same as before.

The improved Adagrad algorithm replaces the square of the gradient with the length of the gradient and also removes the square root operation in (3).

Pseudocode for Improved Adagrad

```
Input:
    $nb\_epochs \leftarrow$ number of epochs
    $lr \leftarrow$ learning rate
BEGIN:
    $smooth\_t \leftarrow$ smooth term
    $s \leftarrow 0$
    $epochs \leftarrow 0$
    while $epochs < nb\_epochs$
        do $g \leftarrow$ get gradients of $\theta$
           $loss \leftarrow J(\theta)$
           $s \leftarrow s +$ length of $g$
           $\theta \leftarrow \theta - lr * g / (s + smooth\_t)$
           $epochs \leftarrow epochs + 1$
    return $loss$

END
```

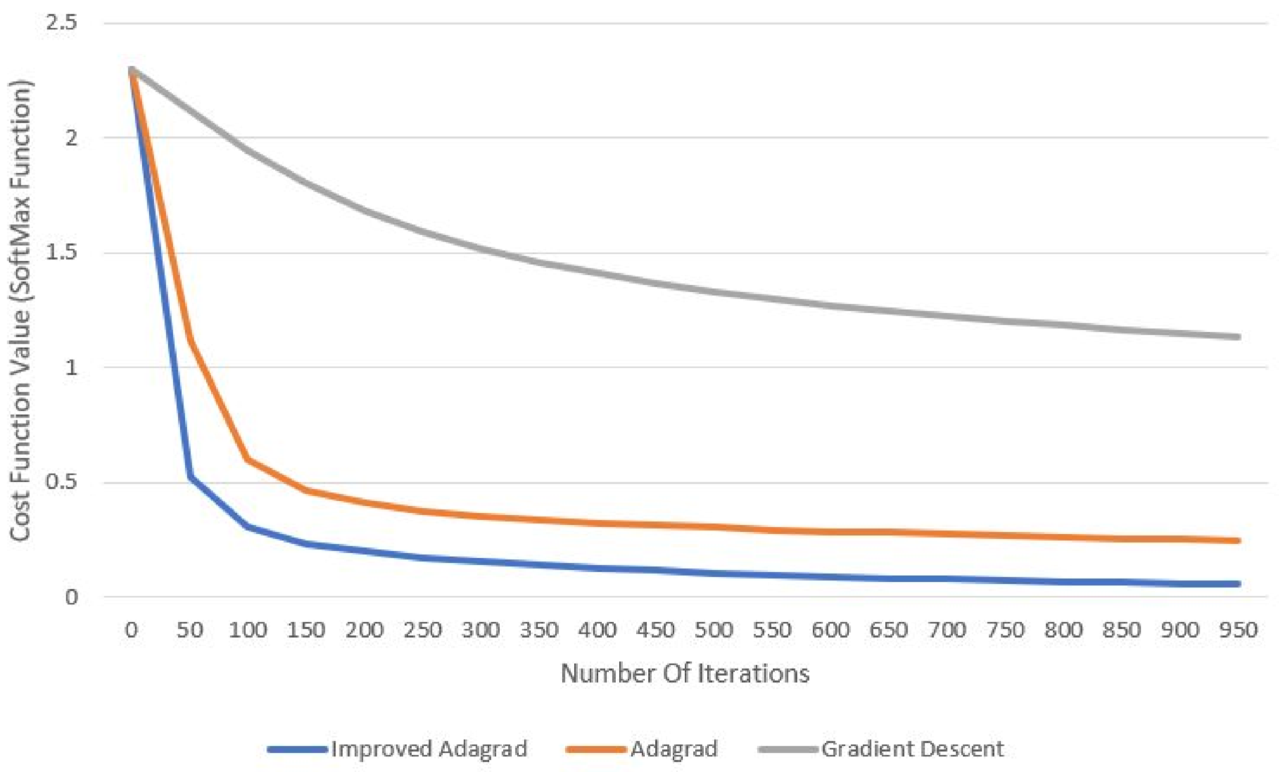
Note: Replace the square of *g* with the length of *g* and remove the square root operation in (3).

3. Problem Statement

To apply the and check convergence of the following algorithms we define a problem statement which requires us to minimize the loss function.

We compared the three approaches by training a 3 Layer - Neural Network to classify binary images of digits using MNIST Dataset.

The convergence results are shown below using the three different descent functions :-



Project Link: Github

References

[1] An improved Adagrad gradient descent optimization algorithm

Available: An improved Adagrad algorithm.

[2] https://www.wikipedia.org/

[3] https://cs.stackexchange.com/

[4] https://keras.io/optimizers/