REPRODUCIBILITY CHALLENGE ON $\mathcal{G}$-SGD: OPTIMIZING RELU NEURAL NETWORKS IN ITS POSITIVELY SCALE-INVARIANT SPACE

Yijian Lei, Yixuan Zhou & Shan Ouyang
School of Electronics and Computer Science
University of Southampton
{yl12u19,yz27u19,so1u19}@soton.ac.uk

ABSTRACT

This report reproduce the work proposed in a conference paper published at 2019 The International Conference on Learning Representations (ICLR) of which the title is $\mathcal{G}$-SGD: Optimizing ReLU Neural Networks in its Positively Scale-Invariant Space. The structure of this report is as follows: Section 1 introduces the idea of the original paper, Section 2 illustrates the reproduction process including methodology and some implementation details. Section 3 contains the setting of experiments and analysis of the results comparing those of reproducing paper, Section 4 summarises our reproduce work and indicates the potential improvements.

1 INTRODUCTION

$\mathcal{G}$-SGD is a method of optimizing ReLU neural network model in a defined $\mathcal{G}$-space where the positively scale-invariant property of ReLU activation function is satisfied Meng et al. (2018). ReLU is proved to possess positively scale-invariant property LeCun et al. (2015) which means multiplying the input with a constant c and dividing the output by c after ReLU is activated will obtain the same result as doing nothing. This property can not be embodied in a traditional weight vector space but the limitation is addressed in Meng et al. (2018).

Firstly, defining the value of the path which is denoted as V_{p^i}. For an L layer neural network model it defines a path denoted as p^i, that starts from one of the input nodes to one of the output nodes and every path is unique. The length of each path (how many weights it contained) is L. And the value of the path V_{p^i} is the cumulative multiplication of the weights it contained. It also mentions that the set of the path values can represent a ReLU neural network completely.

Secondly, finding the basis paths. The $\mathcal{G}$-space is formed by the maximal linearly independent group denoted as $A^{'}$ of the structure matrix A. The The size of A is $m * n$ where m is the number of all the weights $(w_1, w_2, ..., w_m)$ in the neural network model and n is the number of paths $(p^1, p^2, ..., p^n)$. If the weight w_m is in the path p^n, the element a_{mn} is 1 otherwise 0. So, for each column in matrix A there are L elements equal to 1. These basis paths can be find by a method called *Skeleton Method*.

Thirdly, the gradients w.r.t. basis path values can be inferred from the chain rule of weight vector, which is called the *Inverse-Chain-Rule Method*.

The last procedure, *Weight-Allocation Method*, is to convert the updates of path values to updates of weights using the *path-ratio* and *weight-ratio*.

Figure 1 demonstrates the entire procedure of the proposed $\mathcal{G}$-SGD method. Next we will discuss the reproduction in detail.

2 IMPLEMENTATION

In our reproduce work, we practiced the $\mathcal{G}$-SGD algorithm in a 3 layer MLP of which size of all hidden layers is the same. Since the source code of this reproducing paper is not available, **all the codes are written by ourselves.** The reproduction codes are available at `https://github.com/COMP6248-Reproducability-Challenge/g-sgd`.

2.1 SKELETON METHOD

There are two steps for *skeleton method*. First we need to find the *skeleton weights*, then we use the *skeleton weights* to construct the *basis paths*.

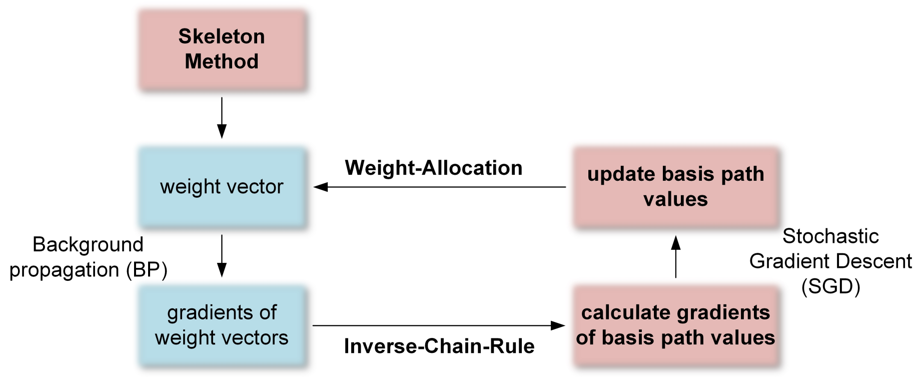
Skeleton Method
weight vector
Weight-Allocation
update basis path values
Background propagation (BP)
Stochastic Gradient Descent (SGD)
gradients of weight vectors
Inverse-Chain-Rule
calculate gradients of basis path values

Figure 1: Illustration of $\mathcal{G}$-SGD method

2.1.1 *skeleton weights*

Assuming that the L MLP model whose width of input layer, hidden layer and output layer is d, h and K respectively. For all hidden-layer weight matrix whose size is $h * h$, the diagonal elements are selected to be the *skeleton weights*. For the first weight matrix which is between input layer and the first hidden layer, the elements ${w^1}_{i\%d,i}, i = 1, 2, ..., h$ are selected to be the *skeleton weights*. For the last weight matrix which is between last hidden layer and the output layer, the elements ${w^L}_{i,i\%K}, i = 1, 2, ..., h$ are selected to be *skeleton weights*. The notation ${w^l}_{j,k}, l = 1, 2, ..., L + 1$ refers the weight that connecting the j_{th} node in l_{th} layer to the k_{th} node in $(l + 1)_{th}$ layer. The rest of the weights are called non-skeleton weights.

2.1.2 *basis paths*

A path which contains at most one non-skeleton weights is a basis path. To get the basis paths, we composed L flag matrix to identity the *skeleton weights* (done by getSkeletonWeightsFlag() function in the code). If it is a skeleton weight the flag is 1 otherwise 0. We search every path from the the input nodes. The position of the weight is recorded by the index of the two nodes connected. For an example, the path from the $1st$ node in input layer to the $2nd$ node in hidden layer then to the $3rd$ node in output layer is recorded as a vector by the index $[0, 1, 2]$. The weights that it contains are ${w^1}_{1,2}$ and ${w^2}_{2,3}$. If the sum of the flag variables of the weights contained in a path is no less than $L - 1$ which means the path contains at most one non-skeleton weight. Then we compute the V_{p^i} of the basis path (done by getBasisPathValue() function in code). The number of basis paths is $m - H$ where H is the number of all hidden nodes.

2.2 INVERSE-CHAIN-RULE (ICR) METHOD

To do the ICR we need to compute the gradient vector $(\frac{dLoss}{dw_1}, \frac{dLoss}{dw_2}, ..., \frac{dLoss}{dw_m})$ w.r.t. the weights vector $(w_1, w_2, ..., w_m)$ at first. Then because we have the equation:

$$(\frac{dLoss}{dw_1}, \frac{dLoss}{dw_2}, ..., \frac{dLoss}{dw_m}) = (\frac{dLoss}{dV_{p^1}}, \frac{dLoss}{dV_{p^2}}, ..., \frac{dLoss}{dV_{p^{m-H}}}) * \begin{bmatrix} \frac{dV_{p1}}{dw_1} & \cdots & \frac{dV_{p1}}{dw_m} \\ \cdots & \cdots & \cdots \\ \frac{dV_{pm-H}}{dw_1} & \cdots & \frac{dV_{pm-H}}{dw_m} \end{bmatrix} \tag{1}$$

So the gradient $(\frac{dLoss}{dV_{p^1}}, \frac{dLoss}{dV_{p^2}}, ..., \frac{dLoss}{dV_{pm-H}})$ w.r.t. the basis path value vector $(V_{p^1}, V_{p^2}, ..., V_{p^{m-H}})$ can be computed.

2.2.1 *back propagation (BP)*

We first compute the gradients w.r.t. weights, $\frac{dLoss}{dw_i}$ for $i = 1, ..., m$ using standard back propagation.

2.2.2 *construct the matrix G*

The matrix in Equation1 is denoted as G which is sparse matrix (done by composeMatrix_GandMatrix_Apr() function in code). If the weight w_i is contained in the basis path p^j the value of $\frac{dV_{pj}}{dw_i}$ is $\frac{V_{pj}}{w_i}$ otherwise 0. So, with the gradient vector $(\frac{dLoss}{dw_1}, \frac{dLoss}{dw_2}, ..., \frac{dLoss}{dw_m})$ and matrix G, it is easy to compute the basis path value gradient vector $(\frac{dLoss}{dV_{p^1}}, \frac{dLoss}{dV_{p^2}}, ..., \frac{dLoss}{dV_{pm-H}})$ by using the pseudo inverse matrix of G.

$$(\frac{dLoss}{dV_{p^1}}, \frac{dLoss}{dV_{p^2}}, ..., \frac{dLoss}{dV_{p^{m-H}}}) = (\frac{dLoss}{dw_1}, \frac{dLoss}{dw_2}, ..., \frac{dLoss}{dw_m}) * G^{-1} \tag{2}$$

2.3 STOCHASTIC GRADIENT DESCENT

Updating the path value vector $(V_{p^1}, V_{p^2}, ..., V_{p^{m-H}})$ by SGD with learning rate lr and the basis path value gradient vector $(\frac{dLoss}{dV_{p^1}}, \frac{dLoss}{dV_{p^2}}, ..., \frac{dLoss}{dV_{p^{m-H}}})$.

$$V_{p^j}{}_{new} = V_{p^j} - lr * \frac{dLoss}{dV_{p^j}}, j = 1, ..., m - H \tag{3}$$

2.4 WEIGHT ALLOCATION (WA) METHOD

Because we have the new basis path value now, we can get the ratio of the basis path value easily. The we can update the weights by the ratio.

2.4.1 *calculate path-ratio*

$$R_{p^j} = \frac{V_{p^j}{}_{new}}{V_{p^j}} \tag{4}$$

2.4.2 *calculate weight-ratio*

To calculate the weight-ratio $r_{w_i}, i = 1, 2, ..., m$, we need to define an operation denoted as $\bigodot$.

For matrix $W_{m*s} = [w_{ij}], i = 1, 2, ..., m, j = 1, 2, ..., s$ and $A_{s*n} = [a_{jk}], j = 1, 2, ..., s, k = 1, 2, ..., n,$

$$W \bigodot A = \begin{bmatrix} \prod_{j=1}^{s} sgn(w_{1j}) * |w_{1j}|^{a_{j1}} & ...... & \prod_{j=1}^{s} sgn(w_{1j}) * |w_{1j}|^{a_{jn}} \\ ...... & ...... & ...... \\ ...... & ...... & ...... \\ \prod_{j=1}^{s} sgn(w_{mj}) * |w_{mj}|^{a_{j1}} & ...... & \prod_{j=1}^{s} sgn(w_{mj}) * |w_{mj}|^{a_{jn}} \end{bmatrix} \tag{5}$$

This operation is defined in code by dotOperation() function.
Because we have the following relationship:

$$(R_{p^1}, R_{p^2}, ..., R_{p^{m-H}}) = (r_{w_1}, r_{w_2}, ..., r_{w_m}) \bigodot A' \tag{6}$$

where A' is the the maximal linearly independent group of A and the size of A' is $m*m-H$. So, the vector $(r_{w_1}, r_{w_2}, ..., r_{w_m})$ can be calculated by the pseudo inverse matrix of A'.

$$(r_{w_1}, r_{w_2}, ..., r_{w_m}) = (R_{p^1}, R_{p^2}, ..., R_{p^{m-H}}) \bigodot A'^{-1} \tag{7}$$

We implemented this step differently compared to the original step demonstrated in section 4.2 of the paper Meng et al. (2018). In the paper, Meng et al. constructed a matrix denoted as $\hat{A}$ which is applicable to the complex neural networks that contain the skip skeleton paths. But this skip structure is not contained in our simple 3-layer MLP. So, the matrix $\hat{A}$ is not applicable. Instead, we replaced $\hat{A}$ to A' for the weight-ratio calculation which is a change in the reproduce procedure.

2.4.3 *update the weights vector*

Finally, with weight ratio we can update the weights vector by simply multiple the ratio and the current weights vector.

$$(w_{1new}, w_{2new}, ..., w_{mnew}) = (r_{w_1}, r_{w_2}, ..., r_{w_m}) * (w_1, w_2, ..., w_m) \tag{8}$$

2.5 $\mathcal{G}$-SGD PSEUDO CODE

The pseudo code of the $\mathcal{G}$-SGD algorithm is clearly presented in Algorithm 1. What is more, after the weight vector w_{inew} for $i = 1, ..., m$ is calculated, it needed to be reshaped into matrix form as the size of input matrix for MLP model before the next iteration.

3 EXPERIMENTS

In this section, We apply $\mathcal{G}$-SGD on Iris data containing 150 samples (train set : validation set = 2:1) to see how much it improves the process compared with ordinary SGD in 200 epochs. As is shown in Figure 2, compared to the performance of SGD, the performance of $\mathcal{G}$-SGD has a significant improvement with training loss reducing greatly to around 0.7 and validation accuracy increasing significantly to around 66% in 20 epochs. This is in line with the result of the reference paper Meng et al. (2018). Due to the less complexity of the neural network in our experiment, the improvement gained by $\mathcal{G}$-SGD is more

Algorithm 1 $\mathcal{G}$-SGD Algorithm

```
Require: initialization $w_{init}$, learning rate $lr$, training data D.
 1: Find skeleton weights and basis path using skeleton method
 2: for $e \in Num\_epochs$ do
 3:     Back propagation to get $\frac{dLoss}{dw_i}$ for $i = 1, ..., m$
 4:     Calculate path value $V_{p^j}$ for $j = 1, ..., m - H$
 5:     Construct matrix $A^{'}$, $G$ and Calculate $\frac{dLoss}{dV_{p^j}}$ for $j = 1, ..., m - H$
 6:     SGD on path value ${V_{p^j}}_{new} = V_{p^j} - lr * \frac{dLoss}{dV_{p^j}}$, $j = 1, ..., m - H$
 7:     Calculate path-ratio $R_{p^j} = \frac{{V_{p^j}}_{new}}{V_{p^j}}$ and weight-ratio $r_{w_i}$ using $A^{'-1}$
 8:     Update the weight $w_{inew} = r_{w_i} * w_i$ for $i = 1, ..., m$
 9: end for
10: return $w_{inew}$ for $i = 1, ..., m$
```

distinct. Figure 3 shows that with the increase of the hidden width H, the performance of both SGD methods on validation set are being better. And the gap between these two method is becoming narrow.

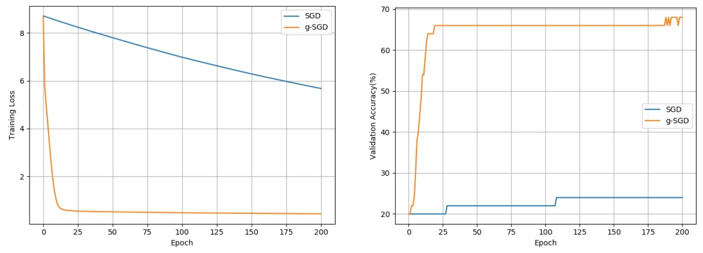

Figure 2: Training loss and validation accuracy of SGD and g-SGD method (Leaning rate:0.0001)

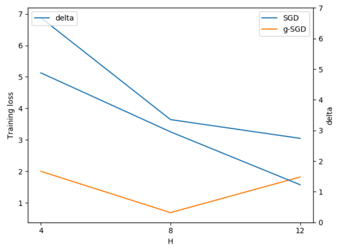

Figure 3: Validation accuracy of SGD and g-SGD method (Changing H)

4 Conclusioins

As the source code is not open to public and we are unable to get access to the details on how authors deal with the complicated network structures, our team tries to find the ”skeleton structure” based on our understanding and test whether SGD will perform better in a positively scale invariant parameter space. The results of our experiment support the idea that $\mathcal{G}$-SGD can bring remarkable improvements in our MLP. This implies that other gradient descents methods such as momentum SGD, Adam and Adagrad may have better performance in $\mathcal{G}$-space as well.

References

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Qi Meng, Shuxin Zheng, Huishuai Zhang, Wei Chen, Qiwei Ye, Zhi-Ming Ma, Nenghai Yu, and Tie-Yan Liu. G-sgd: Optimizing relu neural networks in its positively scale-invariant space. 2018.