REPRODUCIBILITY CHALLENGE REPORT OF THE ARTICLE: BACKPACK: PACKING MORE INTO BACKPROP

Wenxuan Huang & **Yuanpeng Zhang** & **Linhao Song**
School of Electronies and Computer Science
University of Southampton
Southampton,UK
{wh1g19, yz20u19, ls1g19}@soton.ac.uk

ABSTRACT

This report is a summary report on the reproducibility of the article BACKPACK: PACKING MORE INTO BACKPROP. This report first compares the Backpack framework with the existing Pytorch framework and other automatic differential frameworks, and then briefly explores the first-order expansion and second-order expansion The realization principle of the method is finally explained with the same examples in the article as the rich and higher-order automatic differential information brought by the framework, including second-order information and variance information in batch.

1 INTRODUCTION OF THE RESEARCH OBJECTIVES

An important purpose of reproducible research is to trace the source of the original research goal, which is the basis for reproducing and evaluating the performance of the original research. In this study, the initial problem was raised because the current automatic differentiation frameworks, such as Tensorflow, Chainer, and MXNst, etc., automatically calculate the average differential of small batch samples. For example, two-step expansion, or the variance of the gradient value under the current batch, etc., there is no research framework with low calculation cost.

Therefore, the author is committed to implementing this function on the basis of Pytorch. In the Backpack framework, the main functions implemented are first-order extension and second-order extension, which are implemented through the Jacobian structure and the method of spreading information along the graph. In addition, the Backpack framework also supports functions such as traditional feed-forward networks.

In terms of research structure, the project author first analyzed the current standard stochastic gradient descent method (SGD) used for deep learning, and combed the relevant literature to clarify the current application and limitations of the complex second-order extension method. It is complicated and the accuracy offset caused by the high computational cost is a problem; and the current approximate second-order methods all rely on the application of the underlying automatic differentiation, which is also the reason for the limitation of the second-order methods.

In this project, the author works based on the concept of Dangel & Hennig (2019)Dangel & Hennig (2019), that is, modular back propagation establishes a set of implementable Backpack framework. The framework can be built on the existing graph-based back propagation module. In terms of algorithm principles and implementation, the author first explains the principles of first-order and second-order extensions; in order to illustrate the function of the library, the author uses Backpack to implement pre-processing gradient descent optimizer for GGN diagonal approximation and recent Kronecker decomposition KFAC(Martens & Grosse, 2015)Martens & Grosse. (2015), KFLR and KFRA.

2 RELATED ALGORITHM

In this framework, the author gives a clear definition of first-order extension and second-order extension. The first-order extension is defined as extracting more information from the standard backward path; and the second-order extension is calculated from the perspective of computational graph and obtains more knowledge of communication. In this part, the author defines the sequential model f and writes the objective function L according to the loss function l. It is clear that the goal of the framework is to provide more differential information of the loss function l. The following equation shows the objective function.

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} l(f(\theta, x_n), y_n) \tag{1}$$

2.1 METHODS ON FIRST ORDER EXTENSIONS

In the first-order extended framework, an important effort is to extract the discrete gradient information in the standard maximum gradient calculation and use it as the gradient that the framework can output. In section 2.1 of the original article, the author briefly introduced the process of obtaining standard gradient information using propagation, that is, in a sequence transformation, the gradient is repeatedly obtained from the output value of each layer according to gradient descent, so the final parameter theta obtained by the gradient descent method It can be obtained by repeatedly applying the chain rule to the loss function l of each layer.

In the first-order extended framework, an important effort is to extract the discrete gradient information in the standard maximum gradient calculation and use it as the gradient that the framework can output. In section 2.1, the author briefly introduced the process of obtaining standard gradient information, that is, in a sequence transformation, the gradient is repeatedly obtained from the output value of each layer according to gradient descent, so the final parameter theta obtained by the gradient descent method It can be obtained by repeatedly applying the chain rule to the loss function l of each layer.

A common idea for obtaining independent gradients is to perform N backpropagations on a path with N independent gradients, but doing so greatly increases the computational cost, because matrix and matrix multiplication are replaced by matrix sum Vector multiply. Therefore, in the first-order extension framework, the author adds another Jacobian multiplication to the standard backward propagation rule, so as to achieve the calculation of the individual gradient before the gradient sum, as shown in the following formula, where Jthetai represents Jacobian multiplication in a single backward pass. This is also more efficient and cost-saving when compared with for-loop approach.

$$\bigtriangledown_{\theta^i} l_n(\theta)_{n=1}^{N} = [J_{\theta^i} z_n^i]^T \bigtriangledown_{z_n^i} \, l_n(\theta)_{n=1}^{N} \tag{2}$$

2.2 METHODS ON SECOND ORDER EXTENSIONS

The Second order extensions framework performs optimization calculations along the computational graph, thereby outputting more high-order information. Therefore, in this framework, the calculation cost must be considered, because no matter what kind of differential and optimization algorithm is selected, the dimension of the gradient matrix will be over Large, leading to difficulties in overall calculations.

In the Backpack second-order expansion framework, the gradient optimization method is very different from the standard method, because the Jacobian determinant calculation causes a technical bottleneck; therefore, the author chooses the Hessian back propagation method(Mizutani & Dreyfus (2008))Mizutani & Dreyfus (2008) to calculate the loss function (Dangel & Hennig (2019))Dangel & Hennig (2019) For the optimization method of gradient descent, Gaussian-Newten matrix is also used for optimization, and orthogonal factorization is performed on the matrix to reduce its calculation dimension.The following schematic diagram shows the decomposition process of the second-order extension for the additional GGN. It can be seen that the decomposition value of the same-order extension is obtained iteratively in the gradient direction.

Figure 1: The Schematic Diagram of Second Order Extensions

The Hessian matrix is a simulation consisting of the second-order partial derivatives of multivariate functions and can be used to describe the local curvature of the function; the Gauss-Newton method (GGN) (Schraudolph, 2002)Schraudolph. (2002) is one of the optimization methods for nonlinear equations, which is composed of multi-dimensional unconstrained extreme Newton's method evolved. The corresponding iteration formula of Gauss-Newton method is as follows, where r represents for residual while J is the element in Hessian matrix.

$$x^{(i+1)} = x^{(i)} + \triangle, \triangle = -(J_r^T J_r)^{-1} J_r^T r \tag{3}$$

Compared with the Newton method, this method discards the second-order partial derivative implementation in the Hessian matrix, and the information obtained by the first-order degree approximates the second-order information. In the GGN method adopted in this paper, the way to solve the quadratic programming problem and extract the second-order information is not to calculate the complete N-samples matrix. The optimization formula formed by the GGN method is as follows.

$$G(\theta) = \frac{1}{N} \sum_n [J_\theta f(\theta, x_n)]^T \bigtriangledown_f^2 l(f(\theta, x_n), y_n)[J_\theta f(\theta, x_n)] \tag{4}$$

3 Reproducibility Implementations

exact DiagGGN Optimizer

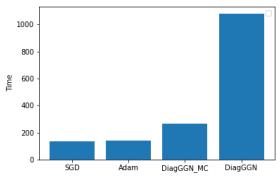
```
class DiagGGNEOptimizer(torch.optim.Optimizer):
  def __init__(self, parameters, step_size, damping):
    super().__init__(parameters, dict(step_size=step_size, damping=
    damping))

  def step(self):
    for group in self.param_groups:
      for p in group["params"]:
        step_direction = p.grad / (p.diag_ggn_exact + group["damping"])
        p.data.add_(-group["step_size"], step_direction)
```

optimization step

```
net_DiagGGNE= model
extend(net_DiagGGNE)
extend(loss_function)
opt_DiagGGNE= DiagGGNEOptimizer(model.parameters(), step_size=STEP_SIZE,
    damping=DAMPING)

for step, (b_x, b_y) in enumerate(loader):
    b_x, b_y = b_x.to(DEVICE), b_y.to(DEVICE)
    output = net_DiagGGNE(b_x)
    loss = loss_function(output, b_y)

    with backpack(DiagGGNExact()):
      loss.backward()

    opt_DiagGGNE.step()
```

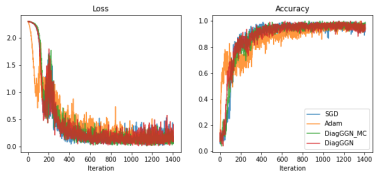
To illustrate the utility of BACKPACK, we implement gradient descent optimizers using the exact diagonal of the generalized Gauss-Newton (DiagGGN) and an curvature approximations based on Monte-Carlo (MC) estimate of the GGN (DiagGGN-MC), and compare with traditional gradient

descent optimizers like SGD and Adam. We create a convolutional Neural Network for MNIST dataset to test these optimizers. Fig.1 and Fig.2 respectively shows run time and performance of each optimizer.



(1).png

Figure 2: Overhead benchmark for computing the gradient on real networks with batch=128



(2).png

Figure 3: Curves of Loss and Accuracy

The figures show that in regard of overhead, DiagGNN takes much more time than others while DiagGNN-MC only adds little overhead and in terms of performance, all of optimizers have similar curves of loss and accuracy. The results indicate that although the computation of the exact diagonal of the GGN is far more expensive than traditional methods, with use of the MC approximation which can be computed at minimal overhead—much less than two backward passes, DiagGNN-MC is much cheaper to compute and has little difference in performance.

4 Conclusions and Limitations

This reproducibility study first analyzes the principle of the framework method provided in this article, and then combs the related method implementation and framework use process provided in the original article, and reproduces the information extraction and gradient of the second-order extensions framework through the MNIST dataset Optimize the process, and discuss the calculation cost, availability, and limitations of the framework.

In general, as a framework different from the standard gradient calculation method, the BackPACK framework is based on the PyTorch framework, in order to extract more gradient-related information, optimized settings and algorithm selection, compared with the previous The method has higher feasibility and efficiency. At the same time, it should also be seen that the second-order extensions framework can not be applied to more cross-entrophy methods, so the framework has more room for expansion.

References

Felix Dangel and Philipp Hennig. A modular approach to block-diagonal hessian approximations for secondorder optimization methods. `http://arxiv.org/abs/1902.01813.`, 2019.

James Martens and Roger B. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. `http://proceedings.mlr.press/v37/martens15.html.`, 2015.

Eiji Mizutani and Stuart E. Dreyfus. Second-order stagewise backpropagation for hessian-matrix analyses and investigation of negative curvature. `https://doi.org/10.1016/j.neunet.2007.12.038.`, 2008.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.