A Comparison of First-Order and Second-Order Optimization Methods

Chenkai Wang
323452
chenkai.wang@epfl.ch

Qinyue Zheng
325450
qinyue.zheng@epfl.ch

Shijian Xu
327448
shijian.xu@epfl.ch

***Abstract*—It is well-known that the second-order optimization methods have many advantages over the first-order optimization methods. However, in practice, especially for deep learning, it is dominated by the simple first-order optimization methods, like SGD and Adam. In this project, we would like to investigate the performance of the second-order methods. We conduct comprehensive experiments on Logistic Regression, Multi-Layer Perceptron (MLP) and Convolutional Neural Networks (CNN) using Adam, SGD-momentum and L-BFGS. Empirical results demonstrate that the second-order optimization methods have some advantages but also have many limitations.**

I. Introduction

A. First-Order Methods

The most common first-order optimization method is gradient descent. The algorithm is extremely simple and surprisingly robust in the sense that it also works well for many loss functions that are not convex, like in deep neural networks. In deep learning, stochastic gradient descent (SGD) is much more popular, where gradient is computed from one single data point or a batch of data points, instead of the whole dataset.

Gradient descent is a very simple iterative algorithm. The step of gradient descent is defined by:

$$x_{t+1} := x_t - \gamma \nabla f(x_t) \tag{1}$$

Here, the γ is stepsize/learning rate, which can be dependent on t. Many variants of GD have been proposed, like proximal gradient descent, accelerated gradient descent, etc.

Another very popular first-order optimization method is Adam [1], which is based on adaptive estimates of lower-order moments. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. Given a loss function $L(w)$, Adam's parameter update is given by:

$$\begin{aligned} g_t &\leftarrow \nabla_\theta f_t(\theta_{t-1}) \\ m_t &\leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &\leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t &\leftarrow m_t/(1-\beta_1^t) \\ \hat{v}_t &\leftarrow v_t/(1-\beta_2^t) \\ \theta_t &\leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon) \end{aligned} \tag{2}$$

B. Second-Order Methods

For second-order optimization methods, we only consider the L-BFGS algorithm. L-BFGS is the **L**imited-memory variant of the **B**royden–**F**letcher–**G**oldfarb–**S**hanno algorithm, which is a Quasi-Newton method.

Newton's method can be used to find a global minimum x^* of a differentiable convex function (i.e., search for a zero of f'. The downside of Newton's method is computing the inverse Hessian matrix and solving the final linear system cost $O(d^3)$ time, which is prohibitive in practice.

Quasi-Newton methods can alleviate this problem. They are second-derivative-free version of Newton's method. In Quasi-Newton methods, the Hessian matrix is approximated by a symmetric matrix H_t, which satisfies the $d-$dimension secant condition:

$$\nabla f(x_t) - \nabla f(x_{t-1}) = H_t(x_t - x_{t-1}) \quad t \geq 1 \tag{3}$$

The update step is:

$$x_{t+1} = x_t - H_t^{-1} \nabla f(x_t) \tag{4}$$

Quasi-Newton methods is a family of related algorithms and any scheme of choosing a symmetric H_t in each step of the secant method that satisfying the secant condition, defines a Quasi-Newton method. Greenstadt's method is a kind of Quasi-Newton method. In this method, it is assumed that $H_t \approx H_{t-1}$ or $H_t^{-1} \approx H_{t-1}^{-1}$. Greenstadt's approach updates H_{t-1}^{-1} by an "error matrix" E_t to obtain:

$$H_t^{-1} = H_{t-1}^{-1} + E_t \tag{5}$$

where E_t should be minimized subject to H_t satisfying the secant conditon.

More concretely, with a generalized error measure term $||AEA^T||_F^2$, where $A \in \mathbb{R}^{d\times d}$ is some fixed invertible transformation matrix, the Greenstadt's approach can be distilled into the following convex constrained minimization problem:

$$\begin{aligned} &\min \frac{1}{2}||AEA^T||_F^2 \\ &s.t. \quad Ey = r, E^T - E = 0 \end{aligned} \tag{6}$$

where $y = \nabla f(x_t) - \nabla f(x_{t-1}), r = \sigma - H_{t-1}^{-1} y, \sigma = x_t - x_{t-1}$.

Let $M \in \mathbb{R}^{d\times d}$ be a symmetric and invertible matrix. Considering the Quasi-Newton method $x_{t+1} = x_t - H_t^{-1}\nabla f(x_t), \quad t \geq 1$, wehre $H_t^{-1} = H_{t-1}^{-1} + E_t$ is chosen such that H_t^{-1} is symmetric and satisfies the secant condition. Define

$$E^* = \frac{1}{y^T M y}(\sigma y^T M + M y \sigma^T - H y y^T M - M y y^T H - \frac{1}{y^T M y}(y^T \sigma - y^T H y) M y y^T M) \tag{7}$$

where $H = H_{t-1}^{-1}, H' = H_t^{-1}, \sigma = x_t - x_{t-1}, y = \nabla f(x_t) - \nabla f(x_{t-1})$. If we choose $E_t = E^*$, the resulting method is called the Greenstadt method with parameter M.

The BFGS method is the Greenstadt method with paramter $M = H' = H_t^{-1}$, in which case the update matrix E^* is:

$$E^* = \frac{1}{y^T \sigma}(-H y \sigma^T - \sigma y^T H + (1 + \frac{y^T H y}{y^T \sigma}) \sigma \sigma^T) \tag{8}$$

Any method in Greenstadt's family avoids the computation of Hessian matrix, and reduce the cost per iteration from $O(d^3)$ to $O(d^2)$. However, in high dimension, even an iteration cost $O(d^2)$ may be prohibitive. L-BFGS method alleviate this problem by using only information from the previous m iterations. The idea is not to compute the actual $H' = H_t^{-1}$ to perform the Quasi-Newton step $x_{t=1} = x_t - \alpha_t H_t^{-1} \nabla f(x_t)$, but only vector $H' \nabla f(x_t)$.

II. Experiments

To compare the performances of the first-order and second-order methods, we conduct experiments on different popular machine learning models, including Logistic Regression (LR), multi-layer fully connected networks (MLP), and deep Convolutional Neural Networks (CNN). Empirical results show that the second-order optimization methods have some advantages over the first-order methods in some ways, but also have very severe limitations.

We use the standard SGD and Adam optimizer provided by PyTorch. For L-BFGS, the orginal implementation in PyTorch is not very efficient. Instead, we use an existing implementation[1].

A. Experiment: Logistic Regression

We apply different optimization methods on multi-class logistic regression using both the MNIST and CIFAR10 dataset. The experiment results are shown in Fig. 1 and Fig. 2. As we all know, logistic regression has a well-studied convex objective, making it suitable for the comparison of different optimizers without worrying about local minimum issues.

We use the same hyperparameters when comparing different optimization algorithms. In all the following experiments, we set the batch size as 128, the number of epochs as 100, and the learning rate as 0.001. The output labels for both MNIST and CIFAR10 are 10. While the logistic regression classifies directly on the 784 dimension image vectors for MNIST, the input dimension is 3072 for CIFAR10. We compare L-BFGS with Adam and SGD-momentum.

According to the result of MNIST in the left of Fig. 1, we found that the L-BFGS nearly yields a similar convergence

[1]https://github.com/nlesc-dirac/pytorch/blob/master/lbfgsnew.py

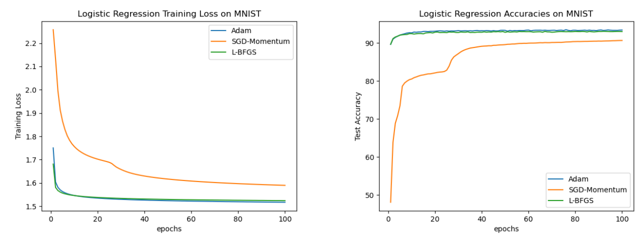

Fig. 1: Results of Logistic Regression on MNIST.

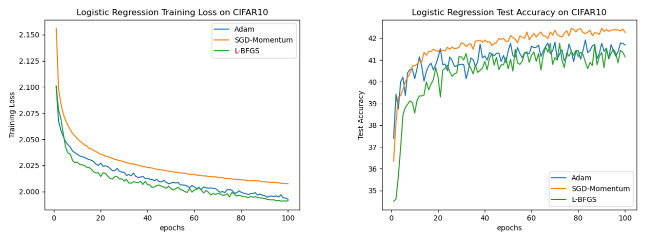

Fig. 2: Results of Logistic Regression on CIFAR10.

as Adam. Both of them not only obtain a much lower loss than SGD-momentum, but also converge faster than SGD. Meanwhile, we found that L-BFGS and Adam perform well on the MNIST dataset with an accuracy of about 95%. The SGD with momentum gets an accuracy of about 88%. We can conclude that L-BFGS obtains a great performance on the MNIST dataset, which is nearly the same as Adam does. Compared with SGD, the result shows the allover superiority of L-BFGS.

Additionally, we compare the performances on the CIFAR10 dataset. Like the performances in MNIST, L-BFGS and Adam converge around the same loss value. To be specific, Adam converges faster at the beginning, but L-BFGS catches up in about 5 epochs. They also achieve a better and faster convergence than SGD, but this time the gap is smaller. We infer that the complexity of the dataset decreases the superiority of L-BFGS and Adam. To evaluate the accuracy among these three optimizers, we notice that all of them get accuracies of around 40% to 42%. All three optimizers tend to oscillate a lot, where SGD has a smaller vibration amplitude than L-BFGS and Adam. Interestingly, SGD obtains better accuracy than both L-BFGS and Adam. And Adam gets a better accuracy with a little advantage of 0.5%, which is shown in the Fig. 2.

B. Experiment: Multi-Layer Neural Network

As one of the classic machine learning models, multi-layer neural networks are powerful models with non-convex objective functions. We choose our MLP model with two fully connected hidden layers with 1000 hidden units each. ReLU activations are used here. And for experiment, we set minibatch size as 128. Hyper-parameters are chosen according to the reported best performance.

On MNIST dataset, MLP does quite well and the results are consistent with the theories. As we can see in Fig. 3, the

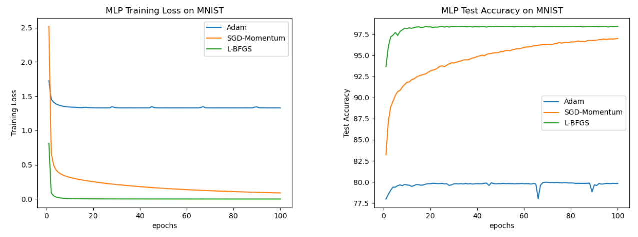

Fig. 3: Results of MLP on MNIST.

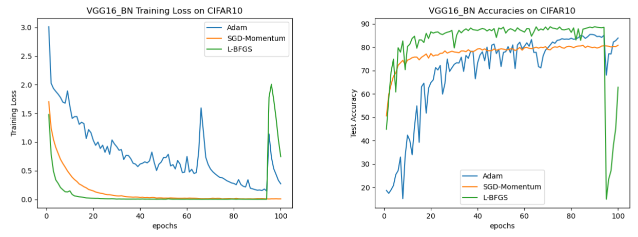

Fig. 5: Results of VGG_16 on CIFAR10.

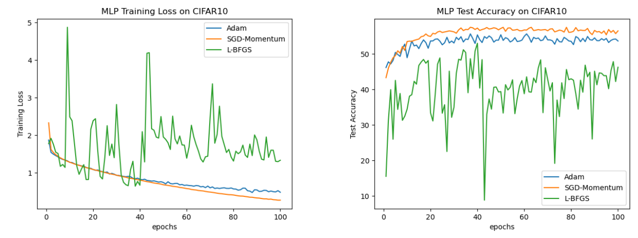

Fig. 4: Results of MLP on CIFAR10.

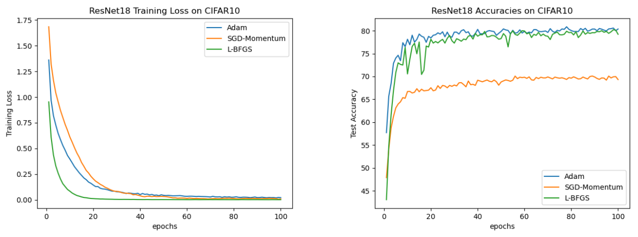

Fig. 6: Results of ResNet18 on CIFAR10.

second-order optimization method L-BFGS outperforms other two first-order methods significantly, achieving both highest accuracy and fastest convergence. However, for CIFAR10, which contains images with relatively higher dimension, the performance of MLP is not that satisfying. The best accuracy achieved by SGD and Adam is only around 58%. At the same time, as shown in Fig. 4, we do observe bizarre performance of MLP with L-BFGS optimizer. This might due to the much sever non-convexity of the objective function. The main goal of this section is not to delve into the non-convex regime but rather compare the efficiency of first and second-order optimization methods for MLP on two different datasets.

In general, the difference in performances between different methods is quite significant. Different optimization methods should be selected according to practical situations. In the sense of fast convergence, L-BFGS is worth choosing. Whereas if taking stability into account, it's reasonable to prefer the fist order method, whose performance is just as satisfying.

C. Experiment: Deep Convolutional Neural Network

Convolutional Neural Networks (CNN) have shown great success in various computer vision tasks. Unlike fully connected nueral netwroks, e.g., MLP, weight sharing in CNNs results in vastly different gradients in different layers.

To make a more comprehensive comparison between the first-order optimization methods and the second-optimization methods, we apply Adam, SGD with Momentum and L-BFGS to two different CNNs: `VGG16_bn` [2] and `ResNet18` [3]. Due to the input size constraints, we only do the experiments on CIFAR10. The input images are augmented by `RandomHorizontalFlip`. The minibatch size is 128, training epochs are 100 and learning rate is 0.001, similar to previous experiments. Experiment results are shown in Fig. 5 and Fig. 6.

We can see that in both networks, L-BFGS performs quite well. For VGG, L-BFGS not only achieves the fastest convergence rate but also achieves the highest test accuracy. For ResNet, L-BFGS also achieves the fastest convergence rate and outperforms SGD with momentum while on par with Adam. It is interesting to see that L-BFGS is not stable on VGG while it performs quite stable on ResNet. In Fig. 5, the training broke at the end and results in very large training loss and very poor test accuracy. We hypothesize that this is due to the much more severe non-convexity of VGG. As we all know, L-BFGS is a convex optimization method, and it might not work/converge on a non-convex problem. The different behaviors on VGG and ResNet might be because ResNet has skip connections. As [4] shows, skip connections promote flat minimizers and prevent the transition to chaotic behavior during training.

III. Conclusion

In this project, we investigate the performances of different first-order and second-order optimization methods. We conduct extensive experiments on MNIST and CIFAR10 datasets with 3 popular machine learning models: logistic regression, MLP and CNN. Empirical results show that in many situations, L-BFGS can achieve faster convergence rate and better accuracy than the first-order methods SGD and Adam. However, the second-order method also has very severe drawbacks. It is not stable when applied to highly non-convex problems. In summary, different optimization methods should be selected according to practical situations. For some models and objective functions with some nice properties, like logistic regression and ResNet, L-BFGS is definitely worth a try.

REFERENCES

[1] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, “Visualizing the loss landscape of neural nets,” *arXiv preprint arXiv:1712.09913*, 2017.