Second-order Optimization Methods in Deep Neural Networks Training

Anastasia Filippova Sofia Blinova Tikhon Parshikov
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Abstract

Optimization in machine learning, both theoretical and applied, is dominated by first-order gradient methods such as stochastic gradient descent (SGD). This article explores the applicability of second-order methods to the image classification problem. We analyze the performance of the Convolutional Neural Network trained with AdaHesian, LBFGS, and BB methods in comparison to the first-order optimization methods such as SGD, SGD-momentum, and Adam.

1. Introduction

It is well known that second-order optimization methods such as Newton's method outperform first-order optimization methods in convex problems. However, they are far less prevalent in the non-convex optimization in the deep learning problems due to their prohibitive computation, memory, and communication costs. In this article, we explore the applicability of second-order methods to the image classification problem. We are considering three stochastic quasi-Newtonian methods: AdaHesian (Yao et al., 2020), LBFGS (Fletcher, 1987) and BB (BARZILAI & BORWEIN, 1988). The stochastic quasi-Newtonian methods are compared with the first-order stochastic methods — SGD, SGD-momentum (Ruder, 2016) and Adam (Kingma & Ba, 2014).

The methods are compared on CIFAR-10 classification problem (Krizhevsky & Hinton, 2009) using a Convolutional Neural Network (CNN). To compare the methods, we use the final value of the accuracy on the test set, the rate of convergence of the method on the train and validation sets, and the time spent on one epoch.

2. Optimization methods

2.1. First order methods

Deep neural networks have not only complex architectures but also a great number of parameters to optimize which makes their training a large-scale task. In this work, we will try several first-order optimization methods. Due to their computational low complexity and efficiency, they are perfectly fit for large-scale problems. To optimize loss function f all these methods on each step update its argument using first derivative of the function (its gradient ∇). All of them in general have an update step defined as (1):

$$x_{t+1} = x_t - \gamma_t * \frac{m_t}{v_t}, \tag{1}$$

where γ_t is a learning rate at step t, m_t and v_t are the first and the second moment terms, respectively.

Stochastic gradient descent. Stochastic gradient descent (SGD) is a basic and the most famous method with m_t equal to function's gradient $\nabla_t f(x)$ on each iteration and $v_t = 1$.

As soon as it is a stochastic algorithm, the gradient is taken not on the full data, but on some subset (batch) of it that significantly decreases computational complexity. This method is simple and therefore has issues such as noise and slow convergence due to frequent updates. Some of these issues are addressed in more complex algorithms.

Stochastic gradient descent with momentum. This is an upgraded version of SGD that significantly boost the speed of learning by using such concept as *velocity*. First moment is calculated as $m_t = \mu_t m_{t-1} + (1 - \mu_t)\nabla f(x_t)$, where μ_t is a momentum at step t, and the second one v_t is equal to 1 again.

Adaptive moment estimation (Adam). Adam is a combination of two optimized versions of the SGD. In practice, in many tasks it is the most efficient first-order optimization method with quite a complex formulas (2, 3) for the first and second order moments estimation:

$$m_t = \frac{(1 - \beta_1)\sum_{i=1}^{t} \beta_1^{t-i} \nabla f(x_i)}{1 - \beta_1^t} \tag{2}$$

$$v_t = \sqrt{\frac{(1 - \beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \nabla f(x_i)^2}{1 - \beta_2^t}}, \tag{3}$$

where β^t- decay rate at step t.

2.2. Second order methods

Second-order optimization methods are a more complex and more effective way of solving Machine learning tasks. Such

methods use not only the first but also the second derivative of the loss function on each step and therefore obtain some information about the function's curvature. All of them in general have an update step defined as (4):

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

where H is a Hessian on step t.

At the same time, this group of methods has a high computationally cost due to Hessian computations. This cost significantly increases with the number of parameters.

AdaHessian. AdaHessian is an extension of first-order adaptive methods that allows to reach much better results. This method contains three main components (Hessian diagonal approximation, spatial averaging, and Hessian momentum) and is defined by the Algorithm 2.

Next two methods relate to Quasi-Newton's methods that are usually used when other Newton's methods are not appropriate, i.e. time consuming or cannot be applied. Quasi-Newton's methods are computationally cheaper and faster in comparison to Newton's and do not need to calculate the second derivative. At the same time, they have some drawbacks such as a higher number of convergence steps and a less precise convergence path than Newton's methods.

Limited-memory Broyden-Fletcher-Goldfarb-Shanno. LBFGS is an upgraded version of the most popular Quasi-Newton's method BFGS (BROYDEN, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). Overall, it is quite effective and stable optimization method but it is not always appropriate for different Machine learning tasks due to it requires full batch gradients. The main idea of this method is based on estimation of the inverted Hessian and update step defined as $x_{t+1} = x_t - \gamma H_t^{-1}\nabla f(x_t)$. The estimation of the inverted Hessian is built in the way to meet quasi-newton's equation ($s_{t+1} = H_{t+1}y_{t+1}$, where $y_{t+1} = \nabla f(x_{t+1}) - \nabla f(x_t)$ and $s_{t+1} = x_{t+1} - x_t$). For the update step Algorithm 1 is proposed.

Barzilai-Borwein adaptive learning rate. BB is another Quasi-Newton's algorithm that uses information from the last iterations to define a step size. It estimates the Hessian by η_t^{-1}, where I is an identity matrix and η is a solution of $\min_\eta ||\eta_t^{-1}s_t - y_t||_2^2$, where y_t and s_t are defined in the same way as it was in LBFGS. BB-based adaptive learning rate uses an idea of mini-batches computations and is shown in the Algorithm 3.

3. Experiments

To solve the problem of image classification, we use a CNN which consists of two convolutional layers and two linear layers. We train our model on the CIFAR-10 dataset, which consists of the images of size 32×32 and includes 10 classes.

In each experiment, we train the network using a specific optimizer. LBFGS did not converge at all and returned NaN values. We suppose that the problem is in Hessian approximation and some technical issues with LBFGS PyTorch implementation. For other methods, we obtained the following results.

We selected the optimal learning rate (lr) using the grid search method for each optimizer. The obtained values of the learning rates are presented in the Table 1. Also, it can be seen that the difference in the time of one epoch between the first-order methods and the second-order methods is small. SGD turn out to be the fastest method, and AdaHessian is the slowest one. This could be due to several matrix multiplications that are done in AdaHessian algorithm.

METHOD	LR	ACCURACY	EPOCH TIME
SGD	10^{-1}	0.64	7.67± 0.09
SGD	10^{-2}	0.58	6.88± 0.12
SGD-MOMENTUM	10^{-2}	0.65	6.89± 0.30
ADAM	$5 * 10^{-4}$	0.62	7.00± 0.18
ADAHESSIAN	10^{-1}	0.51	8.28± 0.39
BB	10^{-2}	0.42	7.23 ± 0.13

Table 1. Methods comparison

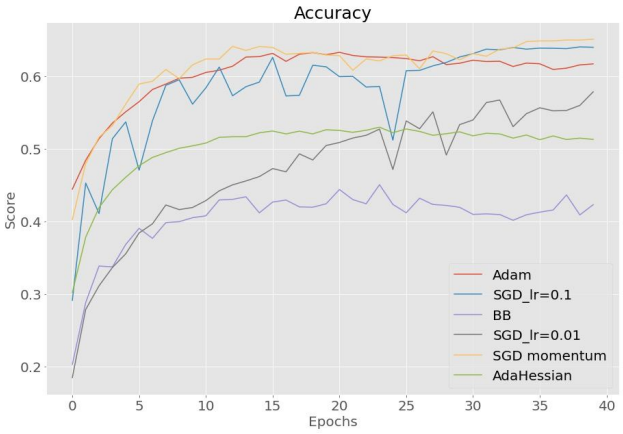

Figure 1. Test accuracy

The accuracy of the second-order methods turned out to be lower than the accuracy of the first-order methods. SGD-momentum showed the best accuracy, while the accuracy of predictions using the BB optimization method turned out to be the worst (see Table 1). It can be seen from the Figure 1 that all methods, except $SGD\_lr = 0.01$, have reached a plateau and their maximum accuracy. Therefore, we can assume that the final accuracy values reflect the accuracy of these methods on the considered problem.
The Figures 2,3 show the dependencies between train and

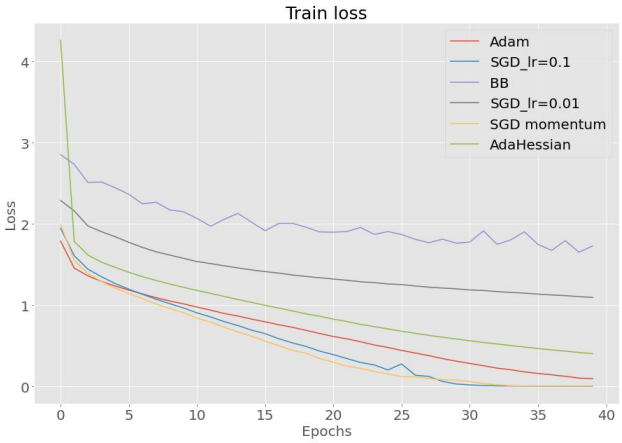

Figure 2. Train loss

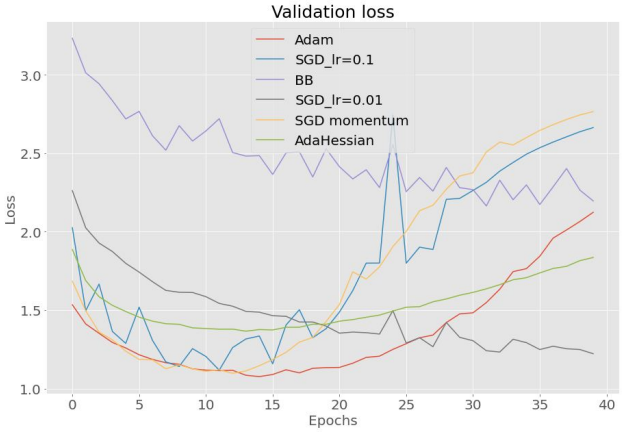

Figure 3. Validation loss

validation losses and the epoch number. It can be seen that the BB has the lowest convergence rate on both validation and train sets. Moreover, $AdaHessian$ converges faster that $SGD\_lr = 0.01$ on training set but it is caused by the non-optimal learning rate of the SGD.

Also, it can be seen from the validation plot (Figure 3) that the AdaHessian loss starts to increase after the $13th$ epoch. It means that the model starts to overfit after this point. However, this reached minimum is much larger than the minimums reached by first-order methods. The most likely reason for this is that the second-order method converged to a local minimum and could not overcome it.

4. Discussion

During our experiments, we noticed several insights. First, despite the lowest learning rate, the model with Adam optimizer reaches optimum and overfits significantly faster than other models. This is especially remarkable because Adam is a first-order method, and they tend to have a lower convergence rate than second-order methods in convex problems.

Also, it is interesting to compare the behavior of models with SGD optimizers with 0.1 and 0.01 learning rates. According to the plots 2,3, we can see that the model trained with $lr = 0.01$ does not converge in 40 epochs because the loss on validation does not "go into plateau". However, a model trained with $lr = 0.1$ overfits just in 11 epochs.

In the case of second-order methods in our experiments, the BB and AdaHessian methods showed the worse results even for a simple convolutional network consisting of only four layers. Moreover, for the LBFGS method, we did not even manage to obtain any results. Thus, second-order methods are more suitable for non-stochastic problems with a small number and a convex optimization function.

5. Conclusion

In this article, we explore the applicability of second-order methods to the image classification problem. We figured out that the performance of the Convolutional Neural Network trained with AdaHesian, LBFGS and BB methods is worse in comparison to the first-order optimization methods such as SGD, SGD-momentum and Adam in terms of accuracy and convergence rate. However, the time needed to train one epoch is almost the same for all considered methods. The reason for the worse model performance trained with second-order optimization methods could be the nonconvexity of the problem. Our experiments show that even for network with low number of parameters second-order methods performed not as good as first-order methods like Adam and SGD. In conclusion, for CIFAR-10 classification problems with deep neural networks first-order optimization methods give better performance than second-order optimization methods.

References

BARZILAI, J. and BORWEIN, J. M. Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 01 1988. ISSN 0272-4979. doi: 10.1093/imanum/8.1.141. URL https://doi.org/10.1093/imanum/8.1.141.

BROYDEN, C. G. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 03 1970. ISSN 0272-4960. doi: 10.1093/imamat/6.1.76. URL https://doi.org/10.1093/imamat/6.1.76.

Fletcher, R. A new approach to variable metric algorithms.

The Computer Journal, 13(3):317–322, 01 1970. ISSN 0010-4620. doi: 10.1093/comjnl/13.3.317. URL https://doi.org/10.1093/comjnl/13.3.317.

Fletcher, R. *Practical Methods of Optimization*. Number . 2 in A Wiley-Interscience publication. Wiley, 1987. ISBN 9780471915478. URL https://books.google.ru/books?id=3EzvAAAAMAAJ.

Goldfarb, D. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24: 23–26, 1970.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

Ruder, S. An overview of gradient descent optimization algorithms, 2016. URL https://arxiv.org/abs/1609.04747.

Shanno, D. F. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24 (111):647–656, 1970.

Yao, Z., Gholami, A., Shen, S., Mustafa, M., Keutzer, K., and Mahoney, M. W. Adahessian: An adaptive second order optimizer for machine learning. 2020. doi: 10.48550/ARXIV.2006.00719. URL https://arxiv.org/abs/2006.00719.

Appendix

Algorithm 1 Search direction in *LBFGS*

Input: gradient of the loss function on step t g_t, last p pairs $\{(y_i, s_i)\}_{i=t-p}^{t-1}$
Output: $H_t^{-1}\nabla f(x_t)$
for $i = t-1, t-2, ..., t-p$ **do**
  $\alpha_i \leftarrow s_i^T g_t / s_i^T y_i$
  $g_t \leftarrow g_t - \alpha_i y_i$
end for
$r \leftarrow \frac{s_{t-1}^T y_{t-1}}{y_{t-1}^T y_{t-1}} g_t$
for $i = t-p, t-p+1, ..., t-1$ **do**
  $\beta \leftarrow y_i^T r / s_i^T y_i$
  $r \leftarrow r + s_i(\alpha_i - \beta)$
end for

Algorithm 2 AdaHessian

Input: initial parameter x_0, learning rate γ, exponential decay rates β_1, β_2, block size b and Hessian power k
Set: $m_0 = 0, v_0 = 0$.
for $t = 1, 2, ...$ **do**
  $g_t \leftarrow$ current step gradient
  $D_t \leftarrow$ current step estimated diagonal Hessian
  Compute: $D_t^{(s)}$ using $D^{(s)}[ib + j] = \frac{\sum_{k=1}^{b} D[ib+k]}{b}$, where D is the Hessian diagonal, $D^{(s)}$ is the spatially averaged Hessian diagonal and $D[i]$ is the i-th element of D
  Update Hessian diagonal with momentum: $\overline{D}_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} D_i^{(s)} D_i^{(s)}}{1-\beta_2^t}}$
  Update $m_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} \nabla f(x_i)}{1-\beta_1^t}$
  Update $v_t = (\overline{D}_t)^k$
  Optimization step: $x_t = x_{t-1} - \gamma_t * m_t / v_t$,
end for

Algorithm 3 BB-based adaptive learning rate

Input: number of epochs K and steps per epoch T, batch size $|B|$, weighting parameter $\beta \in (0; 1]$, initial point $x_{0,0}$ (first index refers to epochs and the second one to steps), $\tau_0, \tau_{min}, \tau_{max}$
for $k = 0, 1, ..., K-1$ **do**
  if $k > 1$ **then**
    $y_{k-1} \leftarrow g_{k-1,T} - g_{k-2,T}$
    $s_{k-1} \leftarrow \frac{1}{T} x_{k-1,T} - x_{k-2,T}$
    $\eta_k \leftarrow \frac{||s_{k-1}||^2}{|s_{k-1}^T y_{k-1}|}$
    $\hat{\eta}_k = \begin{cases} \eta_k, if\ \eta_k \in [\frac{\tau_{min}}{k+1}, \frac{\tau_{max}}{k+1}], \\ \frac{\tau_0}{k+1}, otherwise \end{cases}$
  end if
  $g_{k,0} \leftarrow 0$
  for $t = 0, 1, ..., T-1$ **do**
    Randomly choose batch $B_{k,t}$
    $x_{k,t+1} \leftarrow x_{k,t} - \hat{\eta}_k \nabla f_{B_{k,t}}(x_{k,t})$
    $g_{k,t+1} \leftarrow (1-\beta) g_{k,t} + \beta \nabla f_{B_{k,t}}(x_{k,t})$
  end for
  $x_{k+1,0} \leftarrow x_{k,T}$
end for