Second-order method for Deep Neural Network Optimization
ECE 236C Course Project Report

Pengrui Quan (805227042)
Department of Electrical and Computer Engineering
University of California, Los Angeles

June 11, 2019

1 Introduction

Deep Neural Network (DNN) has shown its prominent position in image classification [1], object detection [3][4], speech recognition [2] and etc. Despite numerous existing optimization methods, Stochastic Gradient Descent (SGD) and its variants are the most widely applied algorithms in searching effective network parameters. However, SGD has several deficiencies: Firstly, gradient descent, compared to second-order method, has slower convergence and zigzagging phenomenon in convex optimization as is shown in convergence analysis. Secondly, SGD commonly comes with time-consuming hyper-parameters tuning, such as choosing an appropriate step size. Furthermore, with quite a large batch size, SGD may render a sharp local minimum and poor generalization of the model [5].

In this project, we are going to try different second-order optimization methods in DNN in a stochastic manner, including Trust-region Method (TR) and (adaptive) Cubic Regularization (CR), and compared their convergence rate, generalization error and computation efficiency with SGD. Codes are made available at `https://github.com/quanpr/Second-order-method-for-Deep-Neural-Network`.

2 Related work

In the context of neural network, directly constructing a Hessian matrix is impractical due to the volume of network parameters. Therefore, there are basically two ways for compensation: one is to use the celebrated Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm with its limited memory version (L-BFGS)[9]. Bollapragada (2018) applied an aggressive batching strategy and stochastic line search to Quasi-Newton method in DNN [6].

Another, in a randomized approach, is to approximate the model locally with quadratic equation using Hessian-vector product or Gauss-Newton Hessian Matrix-vector product with a mini-batch of data:

$$H = \frac{1}{N}\sum_{i=1}^{N}\nabla^2 L(\theta; x_i) \tag{1}$$

$$g = \frac{1}{N}\sum_{i=1}^{N}\nabla L(\theta; x_i) \tag{2}$$

$$f(\theta + d; x) \approx \frac{1}{2}d^T H d + g^T d + f(\theta; x) \tag{3}$$

$$Hd = -g \tag{4}$$

and work out the minimization of the sub-problem (3) through solving a linear equation (4) using Conjugate Gradient method (CG). Martens (2010) applied the Gauss-Newton matrix G and a damping strategy [7] [8] for minimizing quadratic equation (3).

It is worth noticing that through CG method, we do not necessarily construct the full Hessian matrix but making use of the Hessian-vector product or Gauss-Newton matrix-vector product same as the ideas in [7] and [8], where Martens develop a "Hessian-free" approach and apply it to training deep auto-encoders. Hessian-vector products can be computed efficiently using a form of automatic differentiation supported by most popular deep learning library.

Alternatively, besides solving linear equation (4), we can also minimize the quadratic approximation of the model (3) using gradient descent directly as is suggested in [10] (Nocedal J, 2016) and [13] (Liu X, 2018). However, gradient descent empirically requires laborious parameter tuning, i.e. step size, from problem to problem.

3 Trust-region Method (TR)

Typically, TR method is to obtain a descent direction by solving the following quadratic problem (5) within a trust region, and iterate this process until a stopping criterion is met.

$$d_{t+1} = argmin\frac{1}{2}d^T H_t d + g_t^T d, s.t. \|d\| \leq r \tag{5}$$

3.1 Conjugate gradient for TR method

To apply the conjugate gradient for solving (5), we should take care of two requirements different from standard CG method: the Hessian matrix of a DNN is indefinite in general, and the optimal solution may violate the trust region boundary. To handle these conditions, we will adopt the famous CG-Steihaug method, as is mentioned in [21] and [11] (Nocedal J, 2006). That is, if the current searching direction d_j has a non-positive curvature with the Hessian H, then we will return a vector along this searching direction satisfying the trust region criterion. Or if the trust region bound is violated, then we will project the solution

onto the trust region boundary and terminate the CG iteration. We refer our reader to [11], [14], [15] and [21] for detailed algorithms of CG-Steihaug method.

For efficiency consideration, instead of using the full gradient and stochastic Hessian in each Newton update as is done in [7], [8] and [12], we use stochastic gradient and stochastic Hessian to solve linear equation (4).

3.2 Gradient Descent for TR method

An alternative to Newton-CG is to solve the problem (3) using gradient descent. If we apply the standard gradient method, we can obtain the following update:

$$d_{t+1} = d_t - \eta(H_i d_t + g_i) \tag{6}$$

Therefore, we can have the following algorithm for solving problem (3):

Algorithm 1 Trust region method

Initial model parameters θ_0
for $i = 1 : N$ **do**
  estimate stochastic gradient g_i
  Initial d_1 with $d_1 \leftarrow \alpha \nabla g_i$ ▷ s.t. d_t satisfies trust region criterion
  for $t = 1 : N_i$ **do**
    if $\|d_t\| \leq r$ **then**
      estimate Hessian-vector product $H_i d_t$
      $d_{t+1} = d_t - \eta(H_i d_t + g_i)$ ▷ apply gradient descent step to the sub-problem
    else
      project d_{t+1} onto the unit ball r
      break
  update model parameters $\theta_{i+1} \leftarrow \theta_i + d_{t+1}$
return θ_{N+1}

4 Cubic Regularization Method (CR)

The idea of Cubic Regularization Method (CR) is to introduce a cubic penalty term:

$$d = argmin \frac{1}{2} d^T H_t d + g_t^T d + \frac{\rho}{3} \|d\|_2^3 \tag{7}$$

The regularization coefficient ρ describes how trustworthy our second-order approximation is over the parameter space.

4.1 Conjugate Gradient for CR Method

Besides the standard CG approach, we also make three modifications to adopt CG method. Firstly, in each CG iteration, we utilize the stochastic Hessian and approximate full gradient in computation. Since using the whole dataset demands too large GPU memory in practice,

we only adopt 10% of the dataset for full gradient estimation $\nabla L(\theta; x_i)$ and function value evaluation $L(\theta; x_i)$. The reason for using full gradient is that we are more likely to have a descent direction, and gradient evaluation is cheaper than the one of Hessian. As is mentioned, we compute stochastic Hessian with a mini-batch same as the procedures in TR method.

Secondly, since Hessian matrix is indefinite in general, we also adopt an early stopping strategy same as CG-Steihaug method [22]. The iteration is terminated whenever a negative curvature occurs.

Furthermore, to make CR problem solvable by CG method, we use quadratic penalty term instead of the cubic one, which can be further referred as damping[7], like with standard Newton approach. And we also use the famous Levenberg-Marquardt method (LM) to adaptively choose the penalty parameter ρ. Theoretically speaking, since the optimal solution to the quadratic regularized problem is:

$$d = -(H + \rho I)^{-1} g \tag{8}$$

The parameter ρ is to describe how conservative a step we want to make: a large ρ will render a gradient step and a small ρ will give a Newton step. The reduction ratio is a scalar quantity which attempts to measure the accuracy of $f(\theta + d; x)$ and is given by:

$$\alpha = \frac{L(\theta + d; x) - L(\theta; x)}{f(\theta + d; x) - f(\theta; x)} \tag{9}$$

where $L(\theta; x)$ is the actual loss function and $f(\theta; x)$ is the quadratic approximation given DNN parameters θ with x the input training data.

$$\rho = \begin{cases} \rho/2, & \alpha \geq 0.8 \\ \rho, & otherwise \\ 2\rho, & 10^{-4} \leq \alpha \leq 0.8 \end{cases}$$

Therefore, the full CG algorithm can be summarized in Algorithm 2.

It is worth mentioning that only when the estimation of actual decrease is large enough will LM method accepts the network parameter update.

4.2 Gradient descent for CR Method

The gradient descent method for CR problem is similar to the one in TR method, except that gradient is changed to:

$$\nabla f(\theta; x) = Hd + g + \rho \|d\|_2 d \tag{10}$$

and there is no trust region constraint in the descent direction.

Algorithm 2 CG method of adaptive quadratic regularization

Initial model parameters θ_0, step size η
for $i = 1 : N$ **do**
    estimate full gradient g_i
    Initial $d \leftarrow -g_i$
    estimate Hessian-vector product $H_i d$
    solve $(H_i + \rho I)d = -g_i$ using CG method
    set reduction ratio $\alpha = \frac{L(\theta+d;x)-L(\theta;x)}{f(\theta+d;x)-f(\theta;x)}$
    if $\alpha \geq 0.8$ **then**
        $\theta_{i+1} \leftarrow \theta_i + \eta d$
        $\rho \leftarrow \rho/2$
    else if $\alpha \geq 10^{-4}$ **then**
        $\theta_{i+1} \leftarrow \theta_i + \eta d$
    else
        $\theta_{i+1} \leftarrow \theta_i$
        $\rho \leftarrow 2\rho$
return θ_N

5 Experiments

For comparison, we design a Convolutional Neural Network (CNN) with the following configuration as is shown in Table 1.

Layer	**Kernel structure**	**Feature map**
Conv1 + SReLU + BatchNorm	$32 \times 3 \times 3 \times 3$	$32 \times 32 \times 32$
Conv2 + SReLU + BatchNorm	$64 \times 32 \times 3 \times 3$	$64 \times 32 \times 32$
Conv3 + SReLU + BatchNorm	$64 \times 64 \times 3 \times 3$	$64 \times 32 \times 32$
MaxPool	2×2	$64 \times 16 \times 16$
Conv4 + SReLU + BatchNorm	$64 \times 64 \times 3 \times 3$	$64 \times 16 \times 16$
MaxPool	2×2	$64 \times 8 \times 8$
FC layer + SReLU	4096×1024	1024
FC layer + SReLU	1024×256	256
FC layer + SReLU	256×10	10

Table 1: CNN configuration

For terminology clarification, BatchNorm is the same as the normalization method in [18]; FC layer refers to the fully connected layer; SReLU is the continuous activation function with $\epsilon = 0.1$ approximating its counterpart ReLU [17].

$$ReLU(x) = max(x, 0) \tag{11}$$

$$SReLU(x) = \frac{x + \sqrt{x^2 + \epsilon}}{2}, \epsilon > 0 \tag{12}$$

The reasons for changing ReLU to SReLU is that we want to compare the effectiveness of second-order method on ReLU activated CNN and SReLU activated CNN, where more non-linearity is introduced to the network with derivative existing everywhere. We trained and tested CNN on CIFAR-10, a tiny images classification dataset [16].

Experiments are conducted on 4 NVIDIA GeForce GTX 1080 Ti graph cards with 2.40GHz Intel(R) Xeon CPU. Neural network model are either CNN in Table 1 or ResNet-18 [19], a widely used deep neural network structure in computer vision. For gradient descent based TR method, size of trust region γ equals to 0.1, while penalty coefficients ρ of CR method is set to 10. For all gradient descent experiments, batch size is fixed to 128. We also disable momentum and regularization term, set step size to 0.1 and decrease it every 80 epochs (an epoch refers to iterate through the whole training dataset for one time). For conjugate gradient based solver, step size and size of trust region are set to 1 and 0.1 respectively, and penalty coefficients ρ of CR method is initialized with 1. Since each iteration is quite expensive, we only run our algorithms for 80 epochs with batch size 512. Codes are made public at github `https://github.com/quanpr/Second-order-method-for-Deep-Neural-Network`.

5.1 Gradient Descent for TR and CR method

We show the convergence of training loss and test accuracy when the CNN is trained using SGD directly, gradient descent based TR method and gradient descent based CR method. For comparison, we conducted experiments on both SReLU activated CNN and ReLU activated CNN, and their results are presented from Figure 1 to 4. For simplicity, we denote GD as the gradient descent based inner solver and CG as the conjugate gradient based inner solver in the following sections.

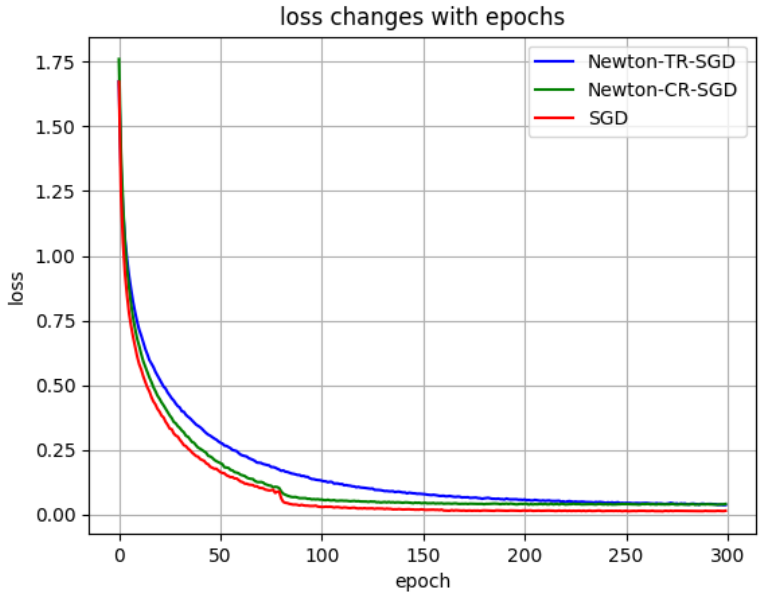

Figure 1: Training loss of SReLU-CNN

From Figure 1 and 2 we can observe that both TR-GD method and CR-GD method are comparable to SGD. Even though they have slightly slower convergence rate, testing accuracy is quite similar to SGD, and CR-GD method even outperforms SGD. From Figure 3 and 4, we can also notice that advantage of CR-GD method is less significant, and we consider this might be due to the less non-linearity in ReLU activated network. Therefore, we may test the performance of CR-GD method on those Recurrent Neural Network (RNN) [20] with *sigmoid* or *tanh* activation function to verify this in future investigation.

We also conduct experiments on ResNet-18 [19], with results summarized in Figure 5 and 6. In the ResNet experiments, we keep hyper-parameter setting the same as previous experiments.

However, differnt from the SReLU-CNN experiments, SGD has the best convergence and Newton-CR-GD method is less prominent in generalization accuracy. Another observation is that the loss of Newton-CR-GD further increase in later training process as is shown in Figrue 5. We conjecture the reason for this unusual phenomenon is that the constant penalty coefficient ρ makes the training unstable, and we think further improvement can be made to adaptively select the coefficient same as the procedures in LM method.

You may refer to Table 2 to 4 for detailed numerical results.

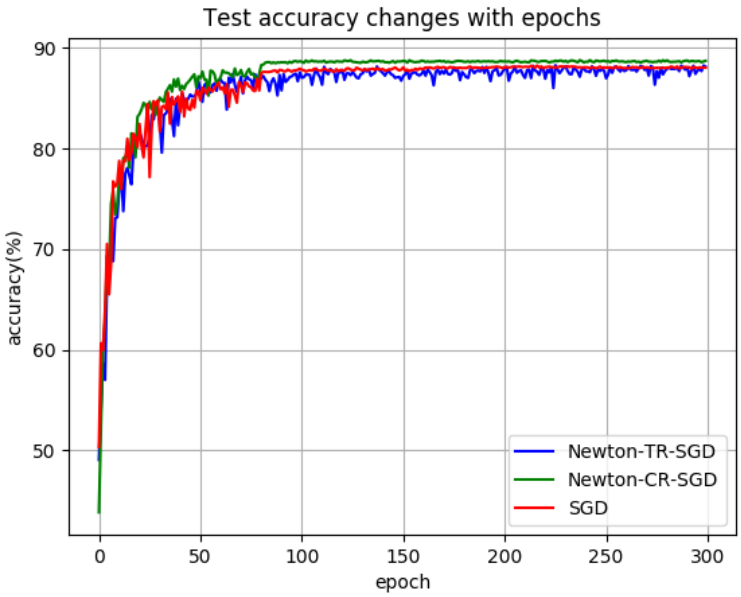

Figure 2: Test accuracy of SReLU-CNN

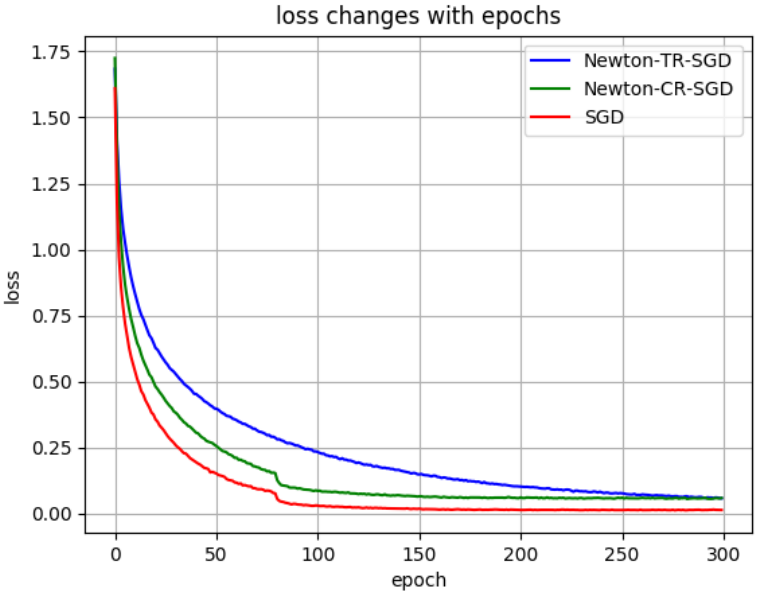

Figure 3: Training loss of ReLU-CNN

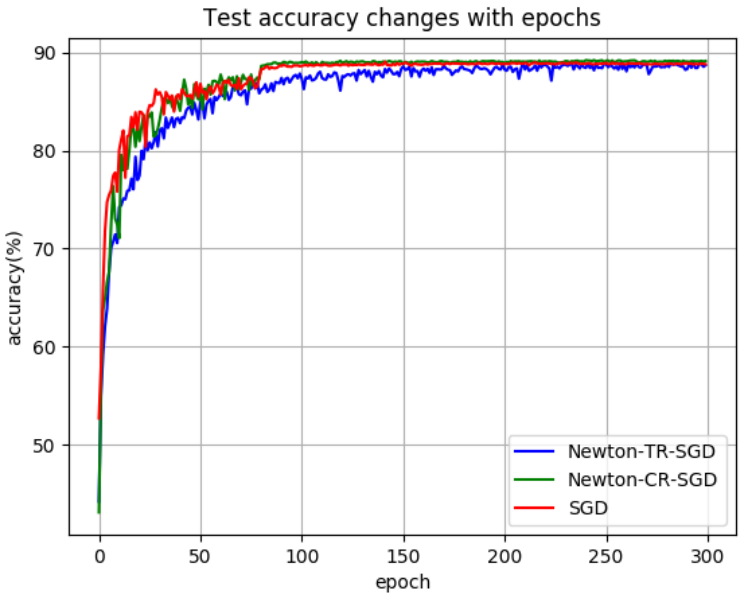

Figure 4: Test accuracy of ReLU-CNN

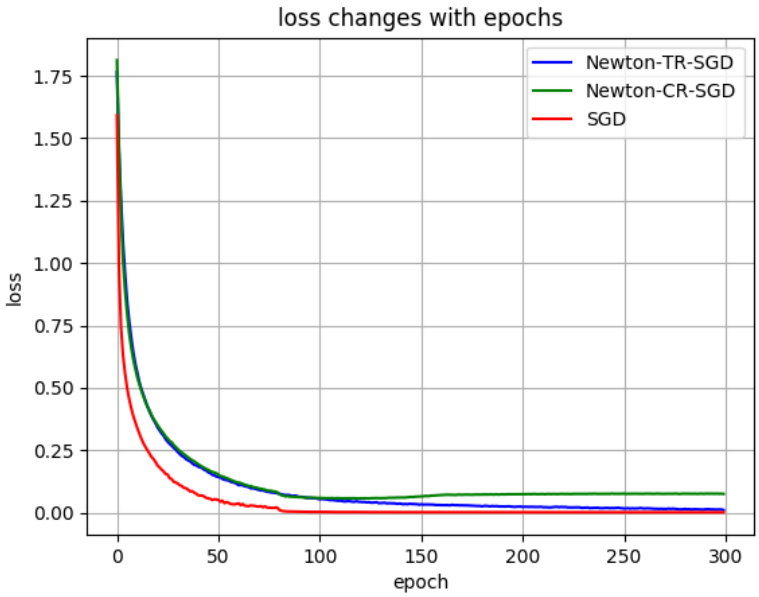

Figure 5: Training loss of SReLU-ResNet

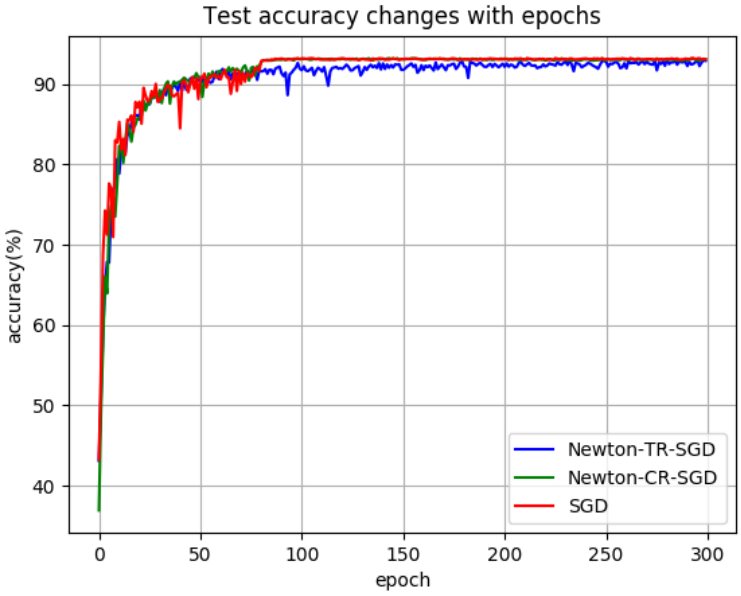

Figure 6: Test accuracy of SReLU-ResNet

5.2 Conjugate Gradient for TR and CR method

In this section, we present the performance of conjugate gradient based solver for TR and CR method. Besides, we also compare the famous LBFGS algorithm with ours.

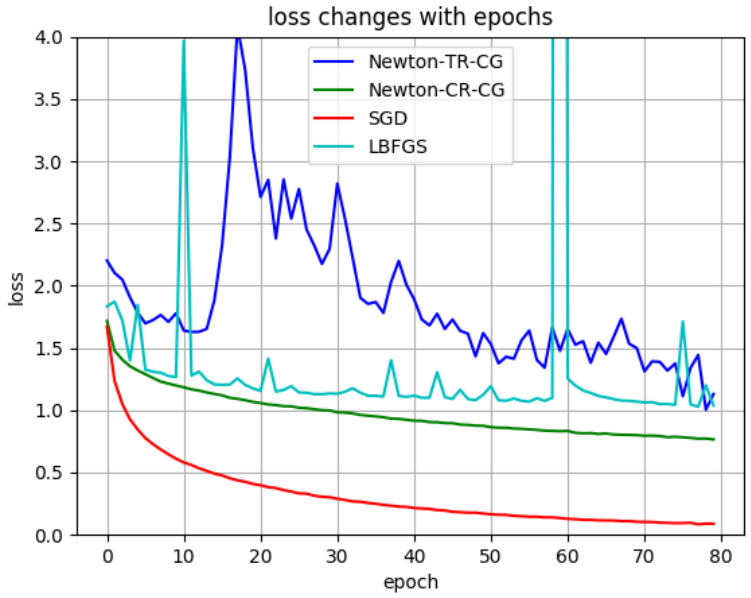

Figure 7: Training loss of SReLU-CNN-CG

From Figure 7, we may observe that all second-order methods conerveges much slower than SGD. And CR method is much smoother than TR method and LBFGS in training phase. We think that might be caused by using full gradient for training and it significantly reduce noise when solving linear equation (4).

However, CR method is notably less stable than the others in terms of generalization in Figure 8. We think this may be caused by LM method's constantly rejecting some update and render a much narrower local minimum. Therefore, despite the stable training loss, generalization performance varies significantly when evaluate the model on test dataset. One alternative is that we can try on Gauss-Newton method where the algorithm is guarantee to have descent step without rejection.

We can also observe from Figure 7 that LBFGS algorithm increases abruptly in training phase, and we think this might be caused by not enough batch size in estimating gradient and the instability of LBFGS when approximating Hessian crudely.

Figure 8: Test accuracy of SReLU-CNN-CG

5.3 Generalization error and time comparison

In this section, we report performance of different methods from Table 2 to 5. We mainly focus on their generalization errors and training efficiency. From Table 2 to 4, we know that gradient based cubic regularization (CR-GD) can outperform the others in generalization performance, and SGD converges to the smallest local minimum with decaying step size. Furthermore, we can observe from Table 5 that second-order method will require about 5× to 20× expensive in each step, depending on which inner solver we chose.

Method	**Final Loss**	**Test accuracy (%)**
SGD	**0.0157**	88.04
TR-GD	0.0381	88.13
CR-GD	0.0223	**88.72**
TR-CG	1.13	72.06
CR-CG	0.773	40.83
LBFGS	1.038	66.29

Table 2: Test accuracy of SReLU activated CNN (TR denotes Trust region; CR denotes cubic regularization; GD denotes gradient descent; CG denotes conjugate gradient; LBFGS denotes Limited-memory BFGS, or so-called Quasi-Newton method)

Method	**Final Loss**	**Test accuracy (%)**
SGD	**0.0130**	88.82
TR-GD	0.0573	88.73
CR-GD	0.0583	**89.15**

Table 3: Test accuracy of ReLU activated CNN (TR denotes Trust region; CR denotes cubic regularization; GD denotes gradient descent)

Method	**Final Loss**	**Test accuracy (%)**
SGD	**0.001**	93.11
TR-GD	0.010	92.93
CR-GD	0.074	**93.13**

Table 4: Test accuracy of SReLU activated ResNet-18 (TR denotes Trust region; CR denotes cubic regularization; GD denotes gradient descent)

6 Summary

In this project, we compare SGD with different second-order optimization method on CIFAR-10 dataset. The gradient descent based Trust-Region (TR) and Cubic Regularization (CR)

Time per step (s)	**SGD**	**TR-GD**	**CR-GD**	**CR-CG**	**CR-CG**	**LBFGS**
SReLU-CNN	0.137	0.598	0.57	6.07	0.209	2.25
ReLU-CNN	0.120	0.604	0.568	-	-	-
SReLU-ResNet	0.123	2.38	1.52	-	-	-

Table 5: Time comparison of different solvers (GD denotes gradient descent is applied to subproblems; CG denotes conjugate gradient is used in solving linear equation approximately)

method can effectively converge to a local minimum that has comparable or even better generalization performance as SGD, with approximately $5\times$ to $20\times$ expensive in each step. On the other hand, both conjugate gradient based TR and conjugate gradient based CR methods have much slower convergence and unstable test accuracy in test phase. Further investigation will be made.

References

[1] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[2] Hinton G, Deng L, Yu D, et al. Deep neural networks for acoustic modeling in speech recognition[J]. IEEE Signal processing magazine, 2012, 29.

[3] Ren S, He K, Girshick R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[C]//Advances in neural information processing systems. 2015: 91-99.

[4] Everingham M, Van Gool L, Williams C K I, et al. The pascal visual object classes (voc) challenge[J]. International journal of computer vision, 2010, 88(2): 303-338.

[5] Keskar N S, Mudigere D, Nocedal J, et al. On large-batch training for deep learning: Generalization gap and sharp minima[J]. arXiv preprint arXiv:1609.04836, 2016.

[6] Bollapragada R, Mudigere D, Nocedal J, et al. A progressive batching L-BFGS method for machine learning[J]. arXiv preprint arXiv:1802.05374, 2018.

[7] Martens J. Deep learning via Hessian-free optimization[C]//ICML. 2010, 27: 735-742.

[8] Martens J, Sutskever I. Learning recurrent neural networks with hessian-free optimization[C]//Proceedings of the 28th International Conference on Machine Learning (ICML-11). 2011: 1033-1040.

[9] Liu D C, Nocedal J. On the limited memory BFGS method for large scale optimization[J]. Mathematical programming, 1989, 45(1-3): 503-528.

[10] Nocedal J. Sub-Sampled Newton Methods for Machine Learning. `https://engineering.jhu.edu/ams/wp-content/uploads/sites/44/2014/08/GOLDMAN-LECTURE-SLIDES.pdf`

[11] Nocedal J, Wright S. Numerical optimization[M]. Springer Science & Business Media, 2006.

[12] Chen P H, Hsieh C. A comparison of second-order methods for deep convolutional neural networks[J]. 2018.

[13] Liu X, Lee J D, Hsieh C J. Better Generalization by Efficient Trust Region Method[J]. 2018.

[14] Lin C J, Weng R C, Keerthi S S. Trust region newton methods for large-scale logistic regression[C]//Proceedings of the 24th international conference on Machine learning. ACM, 2007: 561-568.

[15] Conn A R, Gould N I M, Toint P L. Trust region methods[M]. Siam, 2000.

[16] Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images[R]. Technical report, University of Toronto, 2009.

[17] Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines[C]//Proceedings of the 27th international conference on machine learning (ICML-10). 2010: 807-814.

[18] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[19] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[20] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[21] Steihaug T. The conjugate gradient method and trust regions in large scale optimization[J]. SIAM Journal on Numerical Analysis, 1983, 20(3): 626-637.

[22] Marquardt D W. An algorithm for least-squares estimation of nonlinear parameters[J]. Journal of the society for Industrial and Applied Mathematics, 1963, 11(2): 431-441.