Second-order optimization methods for deep neural network

Pengrui Quan

Department of Electrical and Computer Engineering
UCLA

December 2, 2019

Overview

Flaws of using SGD and its variants

- Require laborious parameters tuning such as learning rate, learning rate decay and etc.
- Keskar et al. (2017) [2] find the generalization gap between large and small batch.

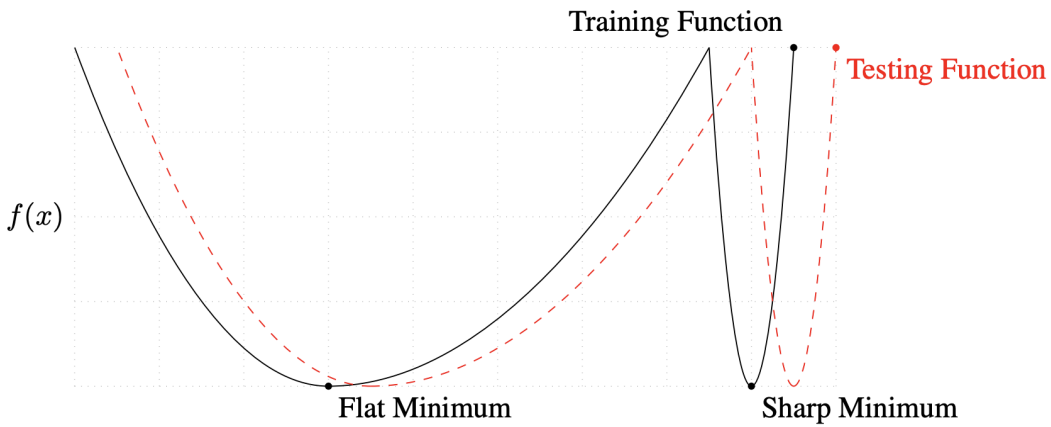

Figure 1: A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)

Figure: Sharpe local minimum

Flaws of using SGD and its variants

- Goyal et al. (2017) [3] use linear scaling rule for adjusting learning rates and develop a new warmup scheme.

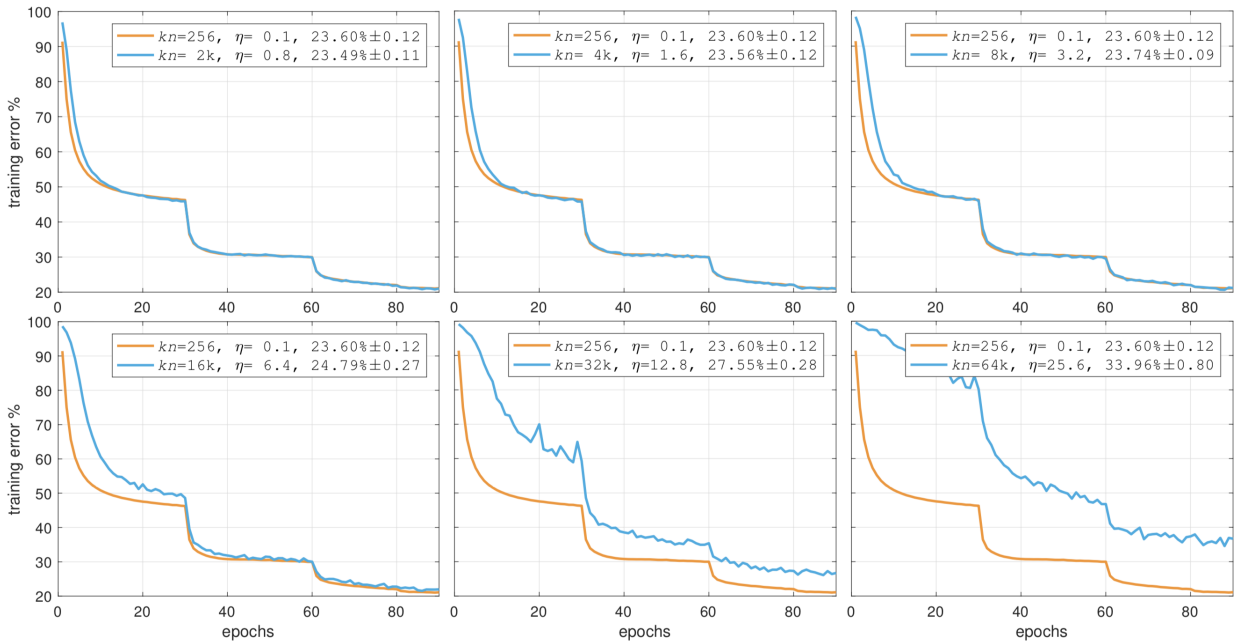

Figure 3. **Training error *vs*. minibatch size.** Training error curves for the 256 minibatch baseline and larger minibatches using gradual warmup and the linear scaling rule. Note how the training curves closely match the baseline (aside from the warmup period) up through 8k minibatches. *Validation* error (mean±std of 5 runs) is shown in the legend, along with minibatch size kn and reference learning rate η.

Figure: learning rate scaling and warm-up strategy

Newton methods in convex optimization literature

- Consider the optimization problem: $\underset{\theta}{\text{minimize}}\, f(\theta) : \mathbb{R}^n \to \mathbb{R}$ with $\nabla^2 f \geq 0$
- Newton method solve the following approximation problem to get a direction d:

$$\underset{\theta}{\text{minimize}} \frac{1}{2} d^T \nabla^2 f(\theta) d + \nabla f(\theta)^T d + f(\theta) \tag{1}$$

- If $\nabla^2 f(\theta)$ is positive semi-definite, then the optimal value is obtain on d where:

$$\nabla^2 f(\theta) d = -\nabla f(\theta) \tag{2}$$

Strengths of using Newton methods

- faster convergence when get close to the minimizer

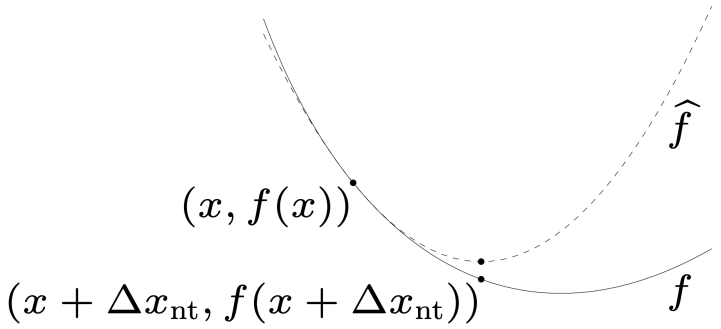

Figure: Good local approximation

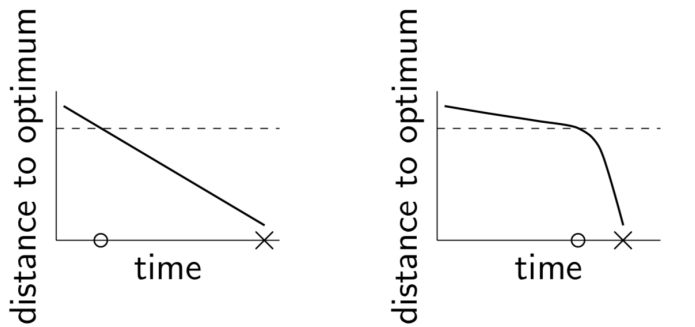

Figure: Compared to gradient descent

Strengths of using Newton methods

- Relieve the problem of ill-conditioning: if gradient changes fast, make a short step; if gradient changes slow, make a long step.
- Newton method does this by choosing $d = -\nabla^2 f(\theta)^{-1} \nabla f(\theta)$

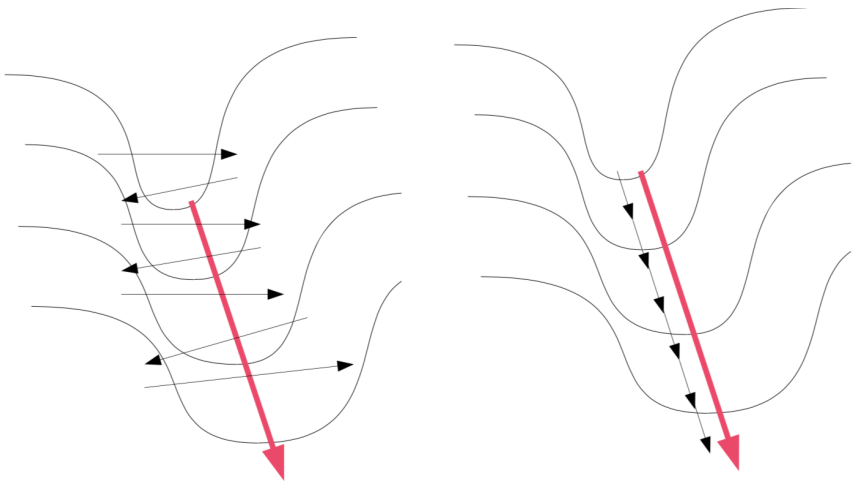

Figure 1. Optimization in a long narrow valley

Figure: ill-conditioning of gradient descent [1]

Issues of applying Newton methods to DNN

- Hessian matrix: $H = \nabla^2 f(\theta)$ is no longer positive semi-definite
- Impossible to construct Hessian matrix explicitly, which will take up to $O(\theta^2)$ complexity.
- Impractical to solve:

$$\nabla^2 f(\theta) d = -\nabla f(\theta) \tag{3}$$

which requires $O(\theta^3)$ steps.

Strategies to mitigate the above issues

Martens (2011) [1] has tried the following strategies to mitigate them:

- Approximate the Hessian with a positive semi-definite matrix:
  - Pre-softmax layer $f(\theta) : \mathbb{R}^n \to \mathbb{R}^m$, where θ is the network parameters
  - Loss function: $L(f(\theta), y)$ where L is a convex function and y is the groundtruth labels
  - The Jacobian matrix of f with respect to θ is:

$$J_\theta = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \frac{\partial f_1}{\partial \theta_3} & \cdots & \frac{\partial f_1}{\partial \theta_n} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \frac{\partial f_2}{\partial \theta_3} & \cdots & \frac{\partial f_2}{\partial \theta_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \theta_1} & \frac{\partial f_m}{\partial \theta_2} & \frac{\partial f_m}{\partial \theta_3} & \cdots & \frac{\partial f_m}{\partial \theta_n} \end{bmatrix} \tag{4}$$

  - The pre-softmax function can be linearized as:

$$\tilde{f}(\theta) = f(\theta_0) + J_\theta(\theta - \theta_0) \tag{5}$$

Approximate Hessian with Gauss-Newton matrix

- Loss function can be approximated as $L(\tilde{f}(\theta), y)$, gradient w.r.t. θ can be calculated as:

$$\nabla \tilde{f}(\theta) = \frac{\partial L}{\partial f}^T J_\theta \tag{6}$$

- The second-order derivative of loss w.r.t. θ then becomes:

$$\nabla^2 \tilde{f}(\theta) = J_\theta{}^T B J_\theta \tag{7}$$

where B is the second-order derivative of loss w.r.t. f:

$$B = \begin{bmatrix} \frac{\partial^2 L}{\partial {f_1}^2} & \cdots & \frac{\partial^2 L}{\partial f_1 f_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial f_1 f_n} & \cdots & \frac{\partial^2 L}{\partial f_n f_n} \end{bmatrix} \tag{8}$$

In this case, the $\nabla^2 \tilde{f}(\theta)$ is the Gauss-Newton matrix, and positive semi-definite.

Linear conjugate gradient method (part 1)

- In stochastic cases, Gauss-Newton matrix is evaluated as:

$$G = \frac{1}{N} \sum_{i=1}^{N} {J_\theta^{(i)}}^T B^{(i)} J_\theta^{(i)} \tag{9}$$

  The gradient is:

$$g = \frac{1}{N} \sum_{i=1}^{N} \nabla f(\theta) \tag{10}$$

- To solve $Gd = -g$, we can rely on linear conjugate gradient method [4].

Linear conjugate gradient method (part 2)

Minimizing the function $f(x) = \frac{1}{2}x^T Ax - b^T x$ equals solving $Ax = b$:

Algorithm 5.1 (CG–Preliminary Version).

Given x_0;

Set $r_0 \leftarrow Ax_0 - b$, $p_0 \leftarrow -r_0$, $k \leftarrow 0$;

while $r_k \neq 0$

$$\alpha_k \leftarrow -\frac{r_k^T p_k}{p_k^T A p_k}; \tag{5.14a}$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k; \tag{5.14b}$$

$$r_{k+1} \leftarrow Ax_{k+1} - b; \tag{5.14c}$$

$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T A p_k}{p_k^T A p_k}; \tag{5.14d}$$

$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k; \tag{5.14e}$$

$$k \leftarrow k + 1; \tag{5.14f}$$

end (while)

Figure: Linear conjugate gradient [6]

A few more strategies

There are also a few strategies that are important but I didn't mention:

- Calculate Gauss-Newton matrix-vector product in deep learning library, i.e. Tensorflow:

$$Gv = {J_\theta}^T B J_\theta v \tag{11}$$

- Use line search to adaptively choose a step size
- Use full batch for gradient and function value evaluation

You may refer to [1] [5] for details.

Experiments on 3-layer CNN

Results on 3-layer CNN with MSE loss function, trained and tested on CIFAR10:

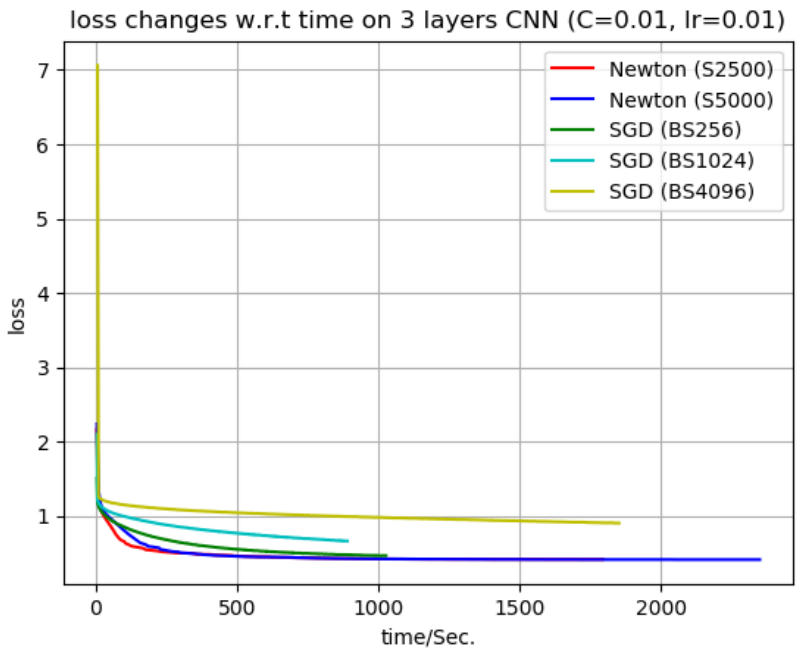

Figure: Training loss

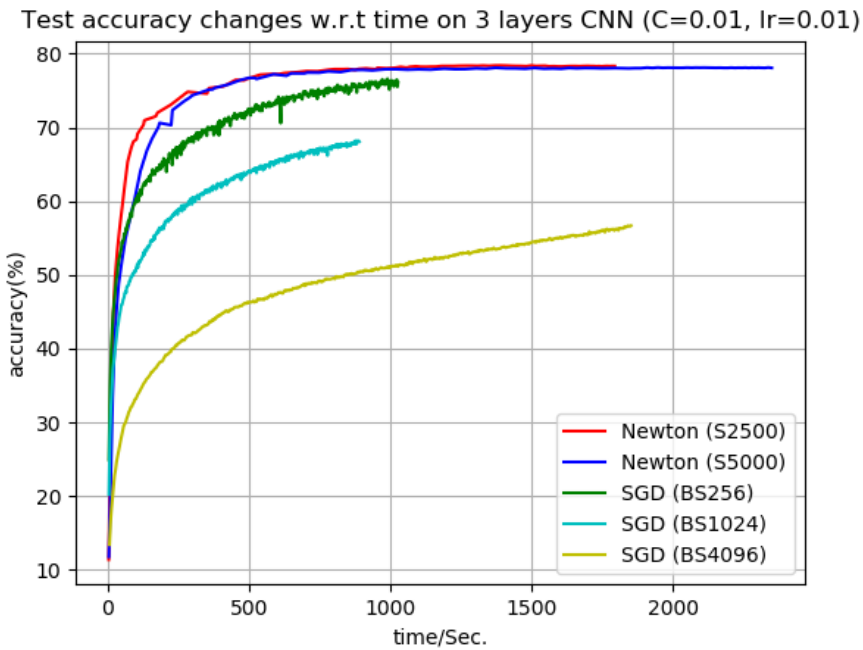

Figure: Testing accuracy

Experiments on 3-layer CNN

C	10% sub-sampled Gv	5% sub-sampled Gv	1% sub-sampled Gv
0.01 /	78.09	78.33	75.62
0.1 /	74.96	75.33	73.45
1 /	73.03	73.35	72.82

Table: Test accuracy of 3-layer CNN (%)

Memory	bsize 1024	bsize 512	bsize 258
10% sub-sampled Gv	3.1GB	1.8GB	1.1GB
5% sub-sampled Gv	3.1GB	1.8GB	1.1GB
1% sub-sampled Gv	3.1GB	1.8GB	1.1GB
SGD	3.1GB	1.8GB	1.1GB

Table: Memory consumption

Experiments on 6-layer CNN

Results on 6-layer CNN with MSE loss function, trained and tested on CIFAR10:

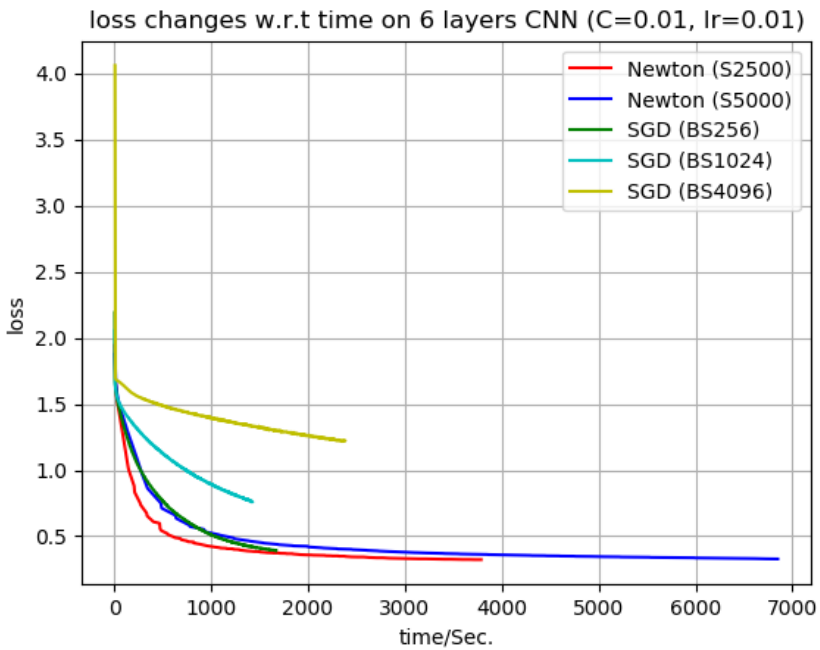

Figure: Training loss

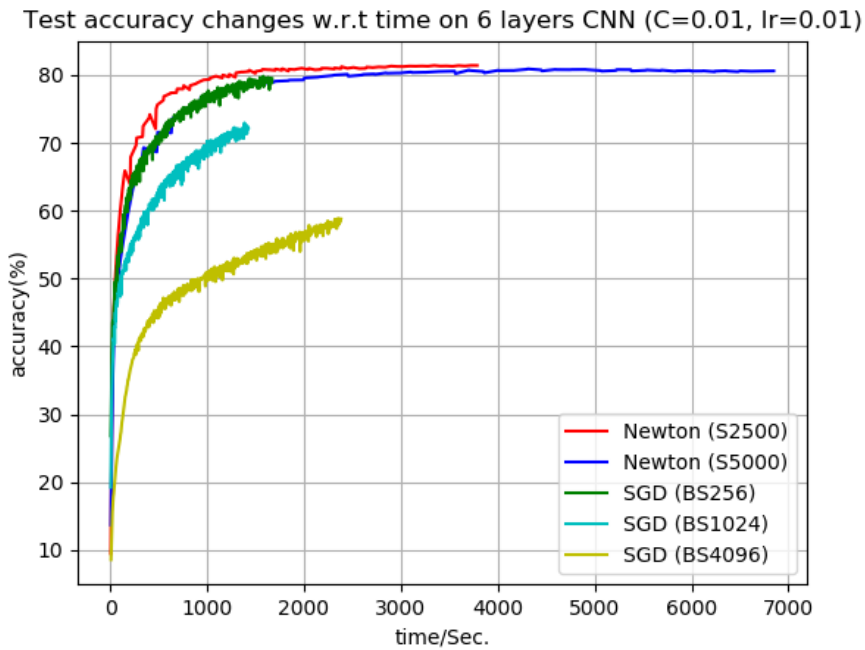

Figure: Testing accuracy

Experiments on 6-layer CNN

C	10% sub-sampled Gv	5% sub-sampled Gv	1% sub-sampled Gv
0.01 /	80.61	81.41	75.50
0.1 /	74.06	73.55	75.90
1 /	71.03	70.83	76.29

Table: Test accuracy of 6-layer CNN (%)

Memory	bsize 1024	bsize 512	bsize 258
10% sub-sampled Gv	7.2GB	3.8GB	2.1GB
5% sub-sampled Gv	7.2GB	3.8GB	2.1GB
1% sub-sampled Gv	7.2GB	3.8GB	2.1GB
SGD	7.2GB	3.8GB	2.1GB

Table: Memory consumption

Robustness of Newton Method

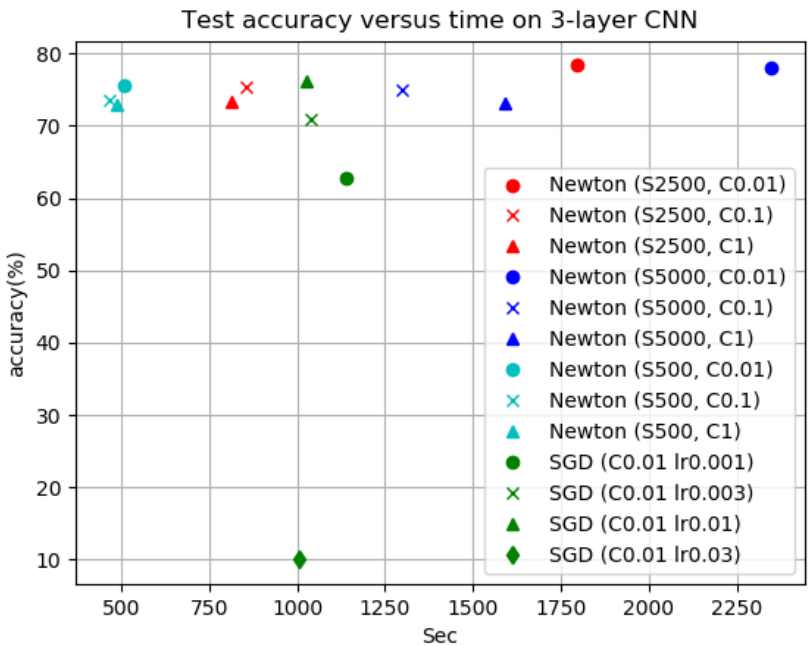

Figure: Final accuracy versus training time on 3-layer CNN

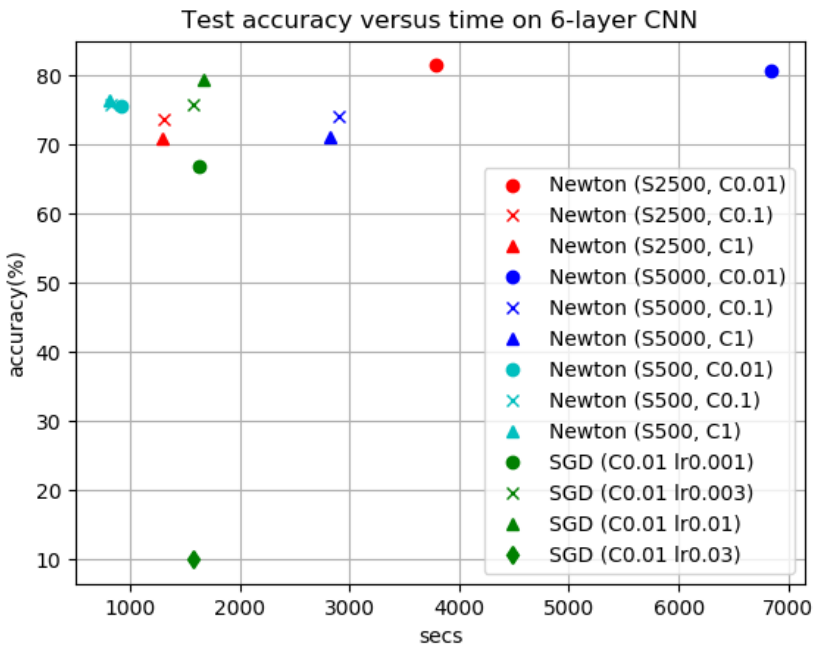

Figure: Final accuracy versus training time on 6-layer CNN

Future works

- We already made our codes public:
  `https://github.com/cjlin1/simpleNN`.
- There are some future works:
  - Solve the quadratic function with standard SGD + momentum.
  - Adaptive batching strategies
  - Reduce the overhead of calculating Jacobian-vector product
  - Effective pre-conditioner

Reference

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

[2] 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.

[3] Goyal P, Dollár P, Girshick R, et al. Accurate, large minibatch sgd: Training imagenet in 1 hour[J]. arXiv preprint arXiv:1706.02677, 2017.

[4] Hestenes M R, Stiefel E. Methods of conjugate gradients for solving linear systems[M]. Washington, DC: NBS, 1952.

[5] Wang C C, Tan K L, Lin C J. Newton Methods for Convolutional Neural Networks[J]. arXiv preprint arXiv:1811.06100, 2018.

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