Newton Methods for Neural Networks: Introduction

Chih-Jen Lin
National Taiwan University

Last updated: May 2, 2020
Outline

1. Introduction
2. Newton method
3. Hessian and Gaussian-Newton Matrices
Outline

1. Introduction
2. Newton method
3. Hessian and Gaussian-Newton Matrices
Optimization Methods Other than Stochastic Gradient

- We have explained why stochastic gradient is popular for deep learning.
- The same reasons may explain why other methods are not suitable for deep learning.
- But we also notice that from the simplest SG to what people are using many modifications were made.
- Can we extend other optimization methods to be suitable for deep learning?
Outline

1. Introduction
2. Newton method
3. Hessian and Gaussian-Newton Matrices
Newton Method

Consider an optimization problem

$$\min_{\theta} f(\theta)$$

Newton method solves the 2nd-order approximation to get a direction $d$

$$\min_d \nabla f(\theta)^T d + \frac{1}{2} d^T \nabla^2 f(\theta) d$$

If $f(\theta)$ isn’t strictly convex, (1) may not have a unique solution
Newton Method (Cont’d)

- We may use a positive-definite $G$ to approximate $\nabla^2 f(\theta)$.

- Then (1) can be solved by

$$Gd = -\nabla f(\theta)$$

- The resulting direction is a descent one

$$\nabla f(\theta)^T d = -\nabla f(\theta)^T G^{-1}\nabla f(\theta) < 0$$
Newton Method (Cont’d)

The procedure:

\[ \text{while stopping condition not satisfied do} \]
\[ \text{Let } G \text{ be } \nabla^2 f(\theta) \text{ or its approximation} \]
\[ \text{Exactly or approximately solve} \]
\[ Gd = -\nabla f(\theta) \]
\[ \text{Find a suitable step size } \alpha \]
\[ \text{Update} \]
\[ \theta \leftarrow \theta + \alpha d. \]
\[ \text{end while} \]
Step Size I

- Selection of the step size $\alpha$: usually two types of approaches
  - Line search
  - Trust region (or its predecessor: Levenberg-Marquardt algorithm)
- If using line search, details are similar to what we had for gradient descent
- We gradually reduce $\alpha$ such that

$$f(\theta + \alpha d) < f(\theta) + \nu \nabla f(\theta)^T (\alpha d)$$
Newton versus Gradient Descent I

- We know they use second-order and first-order information respectively.
- What are their special properties?
- It is known that using higher order information leads to faster final local convergence.
Newton versus Gradient Descent II

- An illustration (modified from Tsai et al. (2014)) presented earlier

```
<table>
<thead>
<tr>
<th>distance to optimum</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>◦</td>
<td>×</td>
</tr>
</tbody>
</table>
```

Slow final convergence  Fast final convergence
But the question is for machine learning why we need fast local convergence?

The answer is no

However, higher-order methods tend to be more robust

Their behavior may be more consistent across easy and difficult problems

It’s known that stochastic gradient is sometimes sensitive to parameters

Thus what we hope to try here is if we can have a more robust optimization method
The Newton linear system

\[ Gd = -\nabla f(\theta) \]  

(2)

can be large.

\[ G \in \mathbb{R}^{n \times n}, \]

where \( n \) is the total number of variables

Thus \( G \) is often too large to be stored
Difficulties of Newton for NN II

- Evan if we can store $G$, calculating
  \[ d = -G^{-1}\nabla f(\theta) \]
  is usually very expensive
- Thus a direct use of Newton for deep learning is hopeless
Existing Works Trying to Make Newton Practical

- Many works tried to address this issue
- Their approaches significantly vary
- I roughly categorize them to two groups
  - Hessian-free (Martens, 2010; Martens and Sutskever, 2012; Wang et al., 2019; Henriques et al., 2018)
  - Hessian approximation (Martens and Grosse, 2015; Botev et al., 2017; Zhang et al., 2017)

In particular, diagonal approximation
Existing Works Trying to Make Newton Practical II

- There are many others where I didn’t put into the above two groups for various reasons (Osawa et al., 2019; Wang et al., 2018; Chen et al., 2019; Wilamowski et al., 2007)
- There are also comparisons (Chen and Hsieh, 2018)
- With the many possibilities it is difficult to reach conclusions
- We decide to first check the robustness of standard Newton methods on small-scale data
- Then we don’t need approximations
Existing Works Trying to Make Newton Practical III

- We will see more details in the project description
Outline

1. Introduction
2. Newton method
3. Hessian and Gaussian-Newton Matrices
Introduction

- We will check techniques to address the difficulty of storing or inverting the Hessian
- But before that let’s derive the mathematical form
For CNN, the gradient of $f(\theta)$ is

$$\nabla f(\theta) = \frac{1}{C} \theta + \frac{1}{l} \sum_{i=1}^{l} (J^i)^T \nabla_{z^{L+1,i}} \xi(z^{L+1,i} \mid y^i, Z^{1,i}),$$

where

$$J^i = \begin{bmatrix}
\frac{\partial z^{L+1,i}_1}{\partial \theta_1} & \cdots & \frac{\partial z^{L+1,i}_1}{\partial \theta_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial z^{n_{L+1},i}_{n_{L+1}}}{\partial \theta_1} & \cdots & \frac{\partial z^{n_{L+1},i}_{n_{L+1}}}{\partial \theta_n}
\end{bmatrix}_{n_{L+1} \times n}, \quad i = 1, \ldots, l, \quad (4)$$
Hessian Matrix II

is the Jacobian of $z^{L+1,i}(\theta)$.

The Hessian matrix of $f(\theta)$ is

$$\nabla^2 f(\theta) = \frac{1}{C} \mathcal{I} + \frac{1}{l} \sum_{i=1}^{l} (J^i)^T B^i J^i$$

$$+ \frac{1}{l} \sum_{i=1}^{l} \sum_{j=1}^{n_L} \frac{\partial \xi(z^{L+1,i}; y^i, Z^{1,i})}{\partial Z_j^{L+1,i}}$$

$$\begin{bmatrix}
\frac{\partial^2 z_j^{L+1,i}}{\partial \theta_1 \partial \theta_1} & \cdots & \frac{\partial^2 z_j^{L+1,i}}{\partial \theta_1 \partial \theta_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial^2 z_j^{L+1,i}}{\partial \theta_n \partial \theta_1} & \cdots & \frac{\partial^2 z_j^{L+1,i}}{\partial \theta_n \partial \theta_n}
\end{bmatrix}.$$
Hessian Matrix III

where $I$ is the identity matrix and $B^i$ is the Hessian of $\xi(\cdot)$ with respect to $z^{L+1,i}$:

$$B^i = \nabla^2_{z^{L+1,i},z^{L+1,i}} \xi(z^{L+1,i}; y^i, Z^{1,i})$$

- More precisely,

$$B^i_{ts} = \frac{\partial^2 \xi(z^{L+1,i}; y^i, Z^{1,i})}{\partial z_t^{L+1,i} \partial z_s^{L+1,i}}, \forall t, s = 1, \ldots, n_{L+1}. \quad (5)$$

- Usually $B^i$ is very simple.
For example, if the squared loss is used,

$$\xi(z^{L+1,i}; y^i) = \|z^{L+1,i} - y^i\|^2.$$ 

then

$$B^i = \begin{bmatrix} 2 \\ \vdots \\ 2 \end{bmatrix}$$

Usually we consider a convex loss function

$$\xi(z^{L+1,i}; y^i)$$

with respect to \(z^{L+1,i}\).
Thus $B^i$ is positive semi-definite

The last term of $\nabla^2 f(\theta)$ may not be positive semi-definite

Note that for a twice differentiable function $f(\theta)$

\[
f(\theta) \text{ is convex if and only if } \\
\nabla^2 f(\theta) \text{ is positive semi-definite}
\]
The Jacobian matrix of $z^{L+1,i}(\theta) \in R^{n_{L+1}}$ is

$$J^i = \begin{bmatrix}
\frac{\partial z_1^{L+1,i}}{\partial \theta_1} & \cdots & \frac{\partial z_1^{L+1,i}}{\partial \theta_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial z_{n_L}^{L+1,i}}{\partial \theta_1} & \cdots & \frac{\partial z_{n_L}^{L+1,i}}{\partial \theta_n}
\end{bmatrix} \in R^{n_{L+1} \times n}, \ i = 1, \ldots, l.$$

- $n_{L+1}$: number of neurons in the output layer
- $n$: number of total variables
- $n_{L+1} \times n$ can be large
The Hessian matrix $\nabla^2 f(\theta)$ is now not positive definite.

We may need a positive definite approximation.

This is a deep research issue.

Many existing Newton methods for NN has considered the Gauss-Newton matrix (Schraudolph, 2002)

$$G = \frac{1}{C} \mathcal{I} + \frac{1}{l} \sum_{i=1}^{l} (J^i)^T B^i J^i$$

by removing the last term in $\nabla^2 f(\theta)$.
The Gauss-Newton matrix is positive definite if $B^i$ is positive semi-definite.

This can be achieved if we use a convex loss function in terms of $z^{L+1,i}(\theta)$.

We then solve

$$Gd = -\nabla f(\theta)$$


References II


