Robustness of Newton Methods and Running Time Analysis

Last updated: June 8, 2020

Tensorflow Problem I

- Some pointed out that if no LM method, the Python code sometimes failed on department's servers
- Turns out this is not an issue of our code, but may be a problem of Tensorflow
- It is very good that someone identified the probable cause and submitted a bug report to Tensorflow people

General Comments I

- Some just show numbers/figures and their experimental settings
- But you need to give analysis and observations

Newton Running Time Analysis I

- For this part we would like to check if you understand some contents of our lectures
- Unfortunately some didn't even know what the problem is
- We are running the same algorithm. Only implementation on Gauss-Newton matrix-vector products are different
- Thus # iterations and # CGs should be almost the same
- In the log I checked (see link1 and link2), # CGs are

Newton Running Time Analysis II

176 and 177

- Because function and gradient evaluations are the same, all we need to newly analyze is the CG time
- Thus checking total time isn't very useful
- Now let's focus on CG
- Theoretical ratio

$$\frac{5\#\ \mathrm{CG}}{3n_{L+1} + 2\#\ \mathrm{CG}}$$

- In my case, this ratio is

Newton Running Time Analysis III

$$2.30$$

- For timing, I got
  - Jacobian stored:
    - 14s for construction
    - 24s for products
  - Jacobian not stored:
    - 56s for products
- So 24s and 56s would be the focus
- The ratio is

$$2.33$$

Newton Running Time Analysis IV

- Some may then say the practical running time is consistent with theoretical analysis
- But this isn't the case
- Among the 24s, 8.8s for

  ```
  p = sum(reshape(net.dzdS{m}, d*ab, nL,
  []) .* reshape(p, d*ab, 1, []),1);
  ```

  5.5s for

  ```
  u_m = reshape(net.dzdS{m}, [],
  nL*num_data) .* Jv';
  ```
- Such new bottlenecks are what I hope you can point out

Newton Running Time Analysis V

- The reason is if without them, then the implementation of not storing Jacobian should be even faster (as we don't have problems of matrix expansion or `accumarray` here)
- For the line

  ```
  p = sum(reshape(net.dzdS{m}, d*ab, nL,
  []) .* reshape(p, d*ab, 1, []),1);
  ```

  let's check the following our course slides

From Course Slides I

- To get

$$\begin{bmatrix} \frac{\partial \boldsymbol{z}^{L+1,1}}{\partial \text{vec}(S^{m,1})^T} \boldsymbol{p}^{m,1} \\ \vdots \\ \frac{\partial \boldsymbol{z}^{L+1,l}}{\partial \text{vec}(S^{m,l})^T} \boldsymbol{p}^{m,l} \end{bmatrix},$$

  we need l matrix-vector products
- There is no good way to transform it to matrix-matrix operations

From Course Slides II

- At this moment we calculate

$$J^{m,i}\boldsymbol{v}^m = \frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \text{vec}(S^{m,i})^T}\boldsymbol{p}^{m,i}, \ \ i = 1, \ldots, l. \qquad (1)$$

by summing up all rows of the following matrix

$$\left[\frac{\partial z_1^{L+1,i}}{\partial \text{vec}(S^{m,i})} \cdots \frac{\partial z_{n_{L+1}}^{L+1,i}}{\partial \text{vec}(S^{m,i})}\right]_{d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}} \times n_{L+1}} \odot$$

$$\left[\boldsymbol{p}^{m,i} \cdots \boldsymbol{p}^{m,i}\right]_{d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}} \times n_{L+1}}.$$

and extend this to cover all instances together

Newton versus SG: Presentation I

- It is better to give figures showing time versus accuracy
- Some give a table listing

  accuracy and time
- But a problem is when to terminate the optimization procedure
- In fact this is an important issue in deep learning training
- By a figure we can more clearly see the trend

Newton versus SG: Performance I

- Most of you found that SG diverges if the learning rate is too large
- This is right
- Selecting the initial learning rate is a painful issue in using SG
- However, Newton also has its own problems (not seen in this project), so it's not widely used yet
- Lots of research still need to be done