- We have discussed sub-sampled Newton method to address the memory issue
- Another technique to address the memory difficulty will be discussed later
- Now we discuss several other considerations to make Newton methods practical

Levenberg-Marquardt Method I

- Besides backtracking line search, in optimization another way to adjust the direction is the Levenberg-Marquardt method (Levenberg, 1944; Marquardt, 1963)
- It modifies the linear system to

$$(G^S + \lambda \mathcal{I})\boldsymbol{d} = -\nabla f(\boldsymbol{\theta})$$

- The value λ is decided by how good the function reduction is.
- It is updated by the following settings.

Levenberg-Marquardt Method II

- If $\boldsymbol{\theta} + \boldsymbol{d}$ is the next iterate after line search, we define

$$\rho = \frac{f(\boldsymbol{\theta} + \boldsymbol{d}) - f(\boldsymbol{\theta})}{\nabla f(\boldsymbol{\theta})^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T G^S \boldsymbol{d}}$$

as the ratio of

$$\frac{\text{actual function reduction}}{\text{predicted reduction}}$$

Levenberg-Marquardt Method III

- By using ρ, the parameter λ_{next} for the next iteration is decided by

$$\lambda_{\text{next}} = \begin{cases} \lambda \times \text{drop} & \rho > \rho_{\text{upper}}, \\ \lambda & \rho_{\text{lower}} \leq \rho \leq \rho_{\text{upper}}, \\ \lambda \times \text{boost} & \text{otherwise}, \end{cases}$$

  where

$$\text{drop} < 1, \text{boost} > 1$$

  are given constants.
- In our code you can see

Levenberg-Marquardt Method IV

```
param.drop = 2/3;
param.boost = 3/2;
```

and

$$\rho_{\text{upper}} = 0.75, \rho_{\text{lower}} = 0.25$$

- If the function-value reduction is not satisfactory, λ is enlarged and the resulting direction is closer to the negative gradient.
- In optimization practice, if backtracking line search has been applied, usually there is no need to apply this LM method

Levenberg-Marquardt Method V

- However, some past works (e.g., Martens, 2010; Wang et al., 2018) on fully-connected networks seem to show that applying both is useful
- The use of LM in training NN is still an issue to be investigated

Function and Gradient Evaluation I

- Recall in gradient evaluation the following main steps are conducted:

$$\Delta \leftarrow \text{mat}(\text{vec}(\Delta)^T P_{\text{pool}}^{m,i})$$

$$\frac{\partial \xi_i}{\partial W^m} = \Delta \cdot \phi(\text{pad}(Z^{m,i}))^T$$

$$\Delta \leftarrow \text{vec}\left((W^m)^T \Delta\right)^T P_{\phi}^m P_{\text{pad}}^m$$

$$\Delta \leftarrow \Delta \odot I[Z^{m,i}]$$

Function and Gradient Evaluation II

- Clearly we must store Z_i, or even $\phi(\text{pad}(Z^{m,i}), \forall i$ after the forward process.
- This is fine for stochastic gradient as we use a small batch of data
- However, for Newton we need the full gradient so we can check the sufficient decrease condition
- The memory cost is then

$$\propto \ \# \text{ total data}$$

- This is not feasible

Function and Gradient Evaluation III

- Fortunately we can calculate the gradient by the sum of sub-gradients

$$\frac{\partial f}{\partial W^m} = \frac{1}{C} W^m + \frac{1}{l} \sum_{i=1}^{l} \frac{\partial \xi_i}{\partial W^m}, \tag{1}$$

$$\frac{\partial f}{\partial \boldsymbol{b}^m} = \frac{1}{C} \boldsymbol{b}^m + \frac{1}{l} \sum_{i=1}^{l} \frac{\partial \xi_i}{\partial \boldsymbol{b}^m}. \tag{2}$$

- Thus we can split the index set $\{1, \ldots, l\}$ of data to, for example, R equal-sized subsets $S_1, \ldots, S_R$

Function and Gradient Evaluation IV

- We sequentially calculate the result corresponding to each subset and accumulate them for the final output.
- For example, to have $Z^{m,i}$ needed in the backward process for calculating the gradient, we must store them after the forward process for function evaluation.
- By using a subset, only $Z^{m,i}$ with i in this subset are stored, so the memory usage can be dramatically reduced.

The Overall Procedure I

- See the Newton method code at https://github.com/cjlin1/simpleNN/blob/master/MATLAB/opt/newton.m

Discussion I

- We have known that at each iteration

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

  is considered
- The remaining issues are
  - How to calculate

$$J^i, \forall i \in S$$

  - How to calculate

$$(J^i)^T \left(B^i(J^i \boldsymbol{v})\right)$$