Lecture 10, Feb 27, 2024

Quasi-Newton Methods (Symmetric Rank 1 (SR1))

- Recall that for quasi-Newton methods, since computing the inverse Hessian is expensive, we use approximations to speed up computation
- Idea: use an iterative update routine to approximate the inverse Hessian
- Start with a quadratic approximation of the objective function $f(\boldsymbol{\theta})$ at the current iteration, $m_k(\boldsymbol{\theta})$
- $m_k(\boldsymbol{\theta}) = f(\boldsymbol{\theta}_k) + \vec{\nabla}^T f(\boldsymbol{\theta}_k)(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T \boldsymbol{B}_k(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$
  - $\boldsymbol{B}_k \in \mathbb{R}^{n \times n}$ is an approximation of the inverse Hessian
  - When $\boldsymbol{\theta} = \boldsymbol{\theta}_k$ we have $m_k = f$ and $\vec{\nabla} m_k = \vec{\nabla} f$
  - These conditions are known as the *zero* and *first-order consistency conditions*
- The minimum of the quadratic model can be obtained by differentiating and setting to zero as usual
  - $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{B}_k^{-1} \vec{\nabla} f(\boldsymbol{\theta}_k)$
- Using the new minimum, we construct a new quadratic approximation $m_{k+1}(\boldsymbol{\theta})$ using the same formula and the new approximate Hessian $\boldsymbol{B}_{k+1}$
- To update the approximate Hessian, we impose the constraint that $m_{k+1}(\boldsymbol{\theta})$ matches the gradient of $f(\boldsymbol{\theta})$ at both $\boldsymbol{\theta}_k$ and $\boldsymbol{\theta}_{k+1}$
  - We want $\vec{\nabla} m_{k+1}(\boldsymbol{\theta}_{k+1}) = \vec{\nabla} f(\boldsymbol{\theta}_{k+1}) + \boldsymbol{B}_{k+1}(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k+1}) = \vec{\nabla} f(\boldsymbol{\theta}_k)$
  - Rearrange to get $\boldsymbol{B}_{k+1}(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k+1}) = \vec{\nabla} f(\boldsymbol{\theta}_k) - \vec{\nabla} f(\boldsymbol{\theta}_{k+1})$
    - This is known as the *secant equation*
- Since the Hessian is SPD, we want $\boldsymbol{B}_{k+1}$ to also be SPD
  - Symmetry gives us $\frac{1}{2}n(n+1)$ independent entries, but the secant equation only gives a system of n equations
  - To obtain a unique solution, we impose the constraint that $\boldsymbol{B}_{k+1}$ should be closest to $\boldsymbol{B}_k$
  - Therefore we use the update formula: $\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \boldsymbol{u}\boldsymbol{u}^T$
    - $\boldsymbol{u}\boldsymbol{u}^T$ is the "symmetric rank 1" matrix
    - This guarantees that $\boldsymbol{B}_{k+1}$ is close to $\boldsymbol{B}_k$ in terms of rank
- Let $\boldsymbol{s}_k = \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k, \boldsymbol{y}_k = \vec{\nabla} f(\boldsymbol{\theta}_{k+1}) - \vec{\nabla} f(\boldsymbol{\theta}_k)$
  - Rewrite the secant equation as $\boldsymbol{B}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k$
  - Plugging in the SR1 update, $\boldsymbol{B}_k\boldsymbol{s}_k + \boldsymbol{u}\boldsymbol{u}^T\boldsymbol{s}_k = \boldsymbol{y}_k \implies \boldsymbol{u}(\boldsymbol{u}^T\boldsymbol{s}_k) = \boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k$
    - This means $\boldsymbol{u} = \gamma(\boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k)$ where $\gamma = \boldsymbol{u}^T\boldsymbol{s}_k$ is a scalar
  - Plug this back in: $\gamma^2(\boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k)(\boldsymbol{s}_k^T(\boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k)) = \boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k$
  - $\gamma^2 = \dfrac{1}{\boldsymbol{s}_k^T(\boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k)}$
- The final update formula is $\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \dfrac{(\boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k)(\boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k)^T}{(\boldsymbol{y}_k - \boldsymbol{B}_k\boldsymbol{s}_k)^T\boldsymbol{s}_k}$
  - However, this only gives the approximate Hessian and not its inverse (inverting at each iteration would be too expensive)

Theorem

Sherman-Morrison-Woodbury (SMW) Formula: Given $\boldsymbol{A} \in \mathbb{R}^{n \times n}, \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{n \times p}$, then

$$(\boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{u}(\boldsymbol{1} + \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1}$$

- Using the SMW formula: $\boldsymbol{B}_{k+1}^{-1} = \boldsymbol{B}_k^{-1} + \dfrac{(\boldsymbol{s}_k - \boldsymbol{B}_k^{-1}\boldsymbol{y}_k)(\boldsymbol{s}_k - \boldsymbol{B}_k^{-1}\boldsymbol{y}_k)^T}{(\boldsymbol{s}_k - \boldsymbol{B}_k^{-1}\boldsymbol{y}_k)^T\boldsymbol{y}_k}$
  - This gives a cost of $\mathcal{O}(n^2)$ (compared to $\mathcal{O}(n^3)$ for matrix inversion)
- An alternative approach to compute $\boldsymbol{B}_{k+1}$ is to formulate it as a constrained optimization problem
  - $\boldsymbol{B}_{k+1} = \min_{\boldsymbol{B}} \|\boldsymbol{B} - \boldsymbol{B}_k\|$ subject to $\boldsymbol{B} = \boldsymbol{B}^T, \boldsymbol{B}(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k+1}) = \vec{\nabla} f(\boldsymbol{\theta}_k) - \vec{\nabla} f(\boldsymbol{\theta}_{k+1})$
  - The choice of matrix norm to use leads to different variations of the method:
    - Davidon-Fletcher-Powell (DFP): $\boldsymbol{B}_{k+1}^{-1} = \boldsymbol{B}_k^{-1} - \boldsymbol{A}_k + \boldsymbol{C}_k$

    - $\boldsymbol{A}_k = \dfrac{\boldsymbol{B}_k^{-1}\boldsymbol{y}_k\boldsymbol{y}_k^T\boldsymbol{B}_k^{-1}}{\boldsymbol{y}_k^T\boldsymbol{B}_k^{-1}\boldsymbol{y}_k}$
    - $\boldsymbol{C}_k = \dfrac{\boldsymbol{s}_k\boldsymbol{s}_k^T}{\boldsymbol{s}_k^T\boldsymbol{y}_k}$
    - This is a rank-2 method
  * Broyden–Fletcher–Goldfarb–Shanno (BFGS):
    - $\boldsymbol{B}_{k+1}^{-1} = \left(\mathbf{1} - \dfrac{\boldsymbol{s}_k\boldsymbol{y}_k^T}{\boldsymbol{s}_k^T\boldsymbol{y}_k}\right)\boldsymbol{B}_k^{-1}\left(\mathbf{1} - \dfrac{\boldsymbol{s}_k\boldsymbol{y}_k^T}{\boldsymbol{s}_k^T\boldsymbol{y}_k}\right) + \dfrac{\boldsymbol{s}_k\boldsymbol{s}_k^T}{\boldsymbol{s}_k^T\boldsymbol{y}_k}$
- Quasi-Newton methods generally have between linear and quadratic convergence; we calls this *superlinear*
- In problems where n is very large such that $\mathcal{O}(n^2)$ is impractical, limiting-memory quasi-Newton methods compute the search step directly

Constrained Optimization – Penalty Methods

- Consider the problem of minimizing $f(\boldsymbol{\theta})$ subject to constraints $g_i(\boldsymbol{\theta}) \geq 0, h_j(\boldsymbol{\theta}) = 0$ where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, q$ and $\boldsymbol{\theta}_l \leq \boldsymbol{\theta} \leq \boldsymbol{\theta}_u$
- Penalty methods minimize $\pi(\boldsymbol{\theta}, \rho_k) = f(\boldsymbol{\theta}) + \rho_k\phi(\boldsymbol{\theta})$
  - $\phi(\boldsymbol{\theta})$ is the *penalty function* and ρ_k is the *penalty parameter*
  - We want $\boldsymbol{\phi}(\theta)$ equal to zero when no constraints are violated and positive when constraints are violated
  - We need to ensure that the objective and the penalty function are appropriately scaled, so one doesn't dominate the other
- Quadratic penalty function: $\phi(\boldsymbol{\theta}_k) = \sum_{i=1}^{m}(\max(0, -g_i(\boldsymbol{\theta})))^2 + \sum_{i=1}^{q}(h_i(\boldsymbol{\theta}))^2$
- Penalty methods template:
  1. Check termination conditions
  2. Minimize $\pi(\boldsymbol{\theta}, \rho_k)$ to find $\boldsymbol{\theta}_{k+1}$
  3. Increment the penalty parameter, $\rho_{k+1} > \rho_k$
     - Typically we multiply by a factor of 1.4 to 10, but this is problem dependent

Nonlinear Least Squares

- $\min_{\boldsymbol{\theta}\in\mathbb{R}^n} = \dfrac{1}{2}\sum_{i=1}^{N} r_i(\boldsymbol{\theta})^2$ where $r_i(\boldsymbol{\theta}) = \hat{f}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) - y^{(i)}$
  - Assume $N > n$, i.e. we have more data points than dimensions
- Let $\boldsymbol{r} = \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} \in \mathbb{R}^N$ and $f(\boldsymbol{\theta}) = \dfrac{1}{2}\|\boldsymbol{r}(\boldsymbol{\theta})\|_2^2$
- $\vec{\nabla} f(\boldsymbol{\theta}) = \sum_{j=1}^{N} r_j(\boldsymbol{\theta})\vec{\nabla} r_j(\boldsymbol{\theta}) = \boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{r}(\boldsymbol{\theta})$
  - $\boldsymbol{J}(\boldsymbol{\theta}) = \left[\dfrac{\partial r_j}{\partial r_i}\right] \in \mathbb{R}^{N\times n}$ is the Jacobian
- $\vec{\nabla}^2 f(\boldsymbol{\theta}) = \sum_{j=1}^{N} \vec{\nabla} r_j(\boldsymbol{\theta})\vec{\nabla} r_j(\boldsymbol{\theta})^T + \sum_{j=1}^{N} r_j(\boldsymbol{\theta})\vec{\nabla}^2\boldsymbol{r}_j(\boldsymbol{\theta})$

  $= \boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{J}(\boldsymbol{\theta}) + \sum_{j=1}^{N} r_j(\boldsymbol{\theta})\vec{\nabla}^2 r_j(\boldsymbol{\theta})$
  - The Jacobian is easy to compute, which gets us most of the way to the Hessian
  - Often the second term is small so we can ignore it altogether and use the Jacobian to approximate the Hessian
    * This happens when the initial residual is small

Gauss-Newton Method

- This is similar to a modified Newton's method with line search
- Use $\vec{\nabla}^2 f(\boldsymbol{\theta}) \approx \boldsymbol{J}(\boldsymbol{\theta})^T \boldsymbol{J}(\boldsymbol{\theta})$ as an approximation of the Hessian
- Solve $\boldsymbol{J}(\boldsymbol{\theta})^T \boldsymbol{J}(\boldsymbol{\theta}) \boldsymbol{p}_k = -\boldsymbol{J}(\boldsymbol{\theta})^T \boldsymbol{r}(\boldsymbol{\theta}_k)$ for the search direction
  - Update $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \boldsymbol{p}_k$ where α_k is chosen via line search
- In the case where the initial residual is small or approximately linear in $\boldsymbol{\theta}$, the Gauss-Newton method can perform similar to the full Newton's method, despite only computing first-order derivatives
- If $\boldsymbol{J}(\boldsymbol{\theta}_k)$ is full-rank and $\vec{\nabla} f(\boldsymbol{\theta}_k) \neq 0$, the search direction is always a valid direction

Stochastic Gradient Descent (SGD)

- In general we have loss function $\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \dfrac{1}{N} \displaystyle\sum_{i=1}^{N} l(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) + \lambda R(\boldsymbol{\theta})$ given data $\mathcal{D} = \{ (\boldsymbol{x}^{(i)}, y^{(i)}) \}_{i=1}^{N}$
  - This consists of the empirical loss and a regularization term
- $\vec{\nabla} \mathcal{L} = \dfrac{1}{N} \displaystyle\sum_{i=1}^{N} \vec{\nabla} l(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) + \lambda \vec{\nabla} R(\boldsymbol{\theta})$
- Applying the steepest descent method, we get the update $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \vec{\nabla} \mathcal{L}(\boldsymbol{\theta}_k; \mathcal{D})$
  - η_k is the *learning rate*
    - In classical methods we use backtracking line search to find this, but in machine learning we typically choose this heuristically, as a constant
    - e.g. start with a sensible value like $\eta = 0.1$; take smaller steps if objective gets worse or we see oscillation; take larger steps if objective reduces too slowly
  - Since we are computing the gradient over the full dataset $\mathcal{D}$, this is known as *full-batch gradient descent*
- Full-batch gradient descent is typically very expensive since we need to compute the gradient over the entire dataset
- Procedure of SGD:
  1. Shuffle training indices $\{ 1, \ldots, N \}$
  2. Initialize $\boldsymbol{\theta}_0$
  3. Repeat until we reach some convergence criteria:
     - For i from 1 to N, $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \vec{\nabla} l(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)})$
- Each iteration of the outer loop is an *epoch*, where we loop over the full dataset
- SGD essentially uses only one datapoint at a time
  - This works because the gradient using one datapoint is an unbiased estimator of the full gradient
  - Let $\boldsymbol{g}_t = l(\boldsymbol{\theta}_k; \boldsymbol{x}^{(t)}, y^{(t)})$, we have that $\mathbb{E}[\boldsymbol{g}_t] = \mathcal{L}(\boldsymbol{\theta}_k; \mathcal{D})$
- Consider gradient descent over a GLM with M terms
  - The cost of full-batch gradient descent is $\mathcal{O}(NM)$, and converges in $\mathcal{O}\left(\log \dfrac{1}{\rho}\right)$
  - The cost of stochastic gradient descent is only $\mathcal{O}(M)$, and converges in $\mathcal{O}\left(\dfrac{1}{\rho}\right)$ iterations
    - Even though SGD takes more iterations to converge (sub-linearly), it's cheaper overall when factoring in the cost per iteration
    - Sometimes it's not practical to do full-batch gradient descent due to the size of the dataset
- In *mini-batch gradient descent* we compute the gradient over a mini-batch that is smaller than the full dataset, but more than 1 sample, in each iteration
  - This is a compromise between full-batch gradient descent and SGD
  - The larger the batch size, the closer we get to full-batch and the faster we converge (in iterations)