ECE472 — Methods and tools for big data

Homework 6

Manuel — JI (Summer 2024)

Reminders

- Push to branch h6 and release with tag hw6
- Answer non-coding questions in a README.md
- Write in a complete style (subject, verb, and object)
- Explain your reasoning and be critical on your results

Ex. 1 — *Gradient descent*

In this exercise we want to investigate the *steepest descent*, a faster variant of gradient descend. We recall that the Taylor expansion of a function f at x_0 is given by

$$f(x) = f(x_0) + \nabla f(x)\big|_{x_0}(x - x_0) + \frac{1}{2}(x - x_0)^\top \nabla^2 f(x)\Big|_{x_0}(x - x_0) + \cdots, \tag{1.1}$$

where ∇ and ∇^2 denote the Jacobian and Hessian, respectively.

1. Calculus reminders.
   a) Recall the definition of the Jacobian and Hessian of a multi-variate function.
   b) Show that minimizing equation (1.1), without considering the derivatives of order higher or equal to three, yields *Newton direction* $-\left(\nabla^2 f(x_0)\right)^{-1}\nabla f(x_0)$.
2. Gradient descent.
   a) Explain how the above strategy can be applied to gradient descent.
   b) Considering the cost of inverting the Hessian, and the adjusted step-size in this version of gradient descent, discuss the speed up game provided by this approach.

Following the approach from the lectures, we now restrict our attention to least square optimization with $f(w) = \frac{1}{2}\sum_{i=1}^{m} \|x_i^\top w - y_i\|_2^2 = \frac{1}{2}\|r(w)\|_2^2$. Note that we added a factor $\frac{1}{2}$ in order to simply the subsequence calculations.

3. Jacobian and Hessian for least square.
   a) Determine the Jacobian of f in term of r.
   b) Express the Hessian of f in term of its Jacobian.

   Hint: remember that $r(w)$ is linear function expressed as a vector $\big(r_1(w), r_2(w), \cdots, r_m(w)\big)$.
4. We denote by $\mathbf{J}$ the Jacobian matrix corresponding to ∇f.
   a) Looking back at (1.1), show that $w = -\left(\mathbf{J}^\top\mathbf{J}\right)^{-1}\mathbf{J}^\top r(w)$.
   b) Applying SVD to $\mathbf{J}^\top\mathbf{J}$, show that there exist two orthogonal matrices U and V, as well as a diagonal matrix Σ, such that $w = -V(\Sigma^\top\Sigma)^{-1}\Sigma^\top U^\top\mathbf{J}^\top r(w)$.

   Hint: really apply SVD to $\mathbf{J}^\top\mathbf{J}$, i.e. **do not** apply it to $\mathbf{J}$.
   c) Show that if we appropriately truncate U into U_t, then $w = -V\Sigma^{-1}U_t\mathbf{J}^\top r(w)$.
5. Based on the previous questions derive an algorithm to compute gradient descent.
6. Without proving anything, explain why the above algorithm will perform better than gradient descent as introduced in the lectures.

Ex. 2 — *PRAM*

Let A be an array with $n = 2^l$ elements, for some l.

1. Sequential algorithm.
   a) Write a sequential algorithm to calculate $S = \oplus_{i=1}^{n} A[i]$, where $\oplus$ is the `xor` binary operation.
   b) Determine the work of this algorithm.
2. Parallel algorithm.
   a) Write a parallel algorithm to calculate $S = \oplus_{i=1}^{n} A[i]$, where $\oplus$ is the `xor` binary operation.
   b) Determine the work and depth of this algorithm.
   c) Evaluate the quality of your parallelized algorithm.

   Hint: think of Brent's theorem.

Ex. 3 — *Gradient descent in Spark*

In this exercise we want to implement and test various versions of gradient descent in Spark. You can code in Scala, Java, or Python using PySpark.

1. Implement batch gradient descent as initially presented in the slide, i.e. without broadcast variables (slide (5.26|5.190)).
2. Implement stochastic gradient descent as initially presented in the slide, i.e. without Hogwild!
3. Implement stochastic gradient descent using Hogwild!.
4. Implement batch gradient descent using broadcast variables.
5. Implement steepest gradient descent (cf. Exercise 1) with and without using broadcast variables.
6. Using the data set from l7, test all the previous variants of gradient descent and discuss your results.