VE472 Homework 6

August 8, 2022

Ex 1

Notation: the Jacobian of f with respect to r is written as $J_f|_r$

Ex 1.1

(a) Jacobian:

$$J_f = \nabla^T f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_m}]$$

Hessian is

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_m \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_m^2} \end{bmatrix}$$

(b)

$$g(x) = f(x_0) + \nabla^T f(x)|_{x_0}(x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 f(x)|_{x_0}(x - x_0)$$

$$g'(x) = \nabla f(x)|_{x_0} + \nabla^2 f(x)|_{x_0}(x - x_0) = 0$$

One can deduce that

$$x - x_0 = -(\nabla^2 f(x_0))^{-1} \nabla f(x_0)$$

Ex 1.2

(a) Previously, we calculate the direction as $-\nabla f(x_0)$, Here we change the direction as $-(\nabla^2 f(x_0))^{-1} \nabla f(x_0)$
(b)

1. It is time-consuming to calculate the Hessian matrix.
2. The Newton's method $-(\nabla^2 f(x_0))^{-1} \nabla f(x_0)$ is quick to be convergent.
3. We don't need to bother how long the step to take, exactly 1.
4. In sum, it takes less iteration, but each iteration may take more time than the gradient descent does.

Ex 1.3

(a) Jacobian:

$$J_f|_r = \nabla^T f|_r = r^T(w) = [r_1(w), ..., r_m(w)]$$

(b) Hessian:

$$\nabla^2 f|_r = \nabla r|_r = I_m$$

is the identity matrix

Ex 1.4

(a) One can simply get $J_r|_w = \nabla^T r(w)|_w = [x_1, x_2, ..., x_m]^T = X$. With the chain rule in vector calculus,

$$J_f|_w = J_f|_r \cdot J_r|_w = r^T(w)\, X,$$

where $X = [x_1, x_2, ..., x_m]^T$. So, it means

$$\nabla f|_w = (J_f|_w)^T = X^T r(w)$$

With the similar strategy, one can get that

$$\nabla^2 f|_w = \nabla(X^T r(w)) = \nabla(r(w))\, X = X^T X$$

Therefore,

$$w = -(X^T X)^{-1} X^T r(w) = -(J^T J)^{-1} J^T r(w)$$

(b) One applies SVD to the positive (semi-)definite matrix $J^T J = U \Sigma V^T$ and replace $A = J^T J = A^T$ in the previous formula

$$\begin{aligned} w &= -A^{-1} J^T r(w) \\ &= -A^{-1}(A^T)^{-1} A^T J^T r(w) \\ &= -(A^T A)^{-1} A^T J^T r(w) \\ &= -V(\Sigma^T \Sigma)^{-1} V^T V \Sigma^T U^T J^T r(w) \\ &= -V(\Sigma^T \Sigma)^{-1} \Sigma^T U^T J^T r(w) \end{aligned}$$

(c) select the first m non-zero diagonal elements of Σ as Σ and choose the first m rows of U^T as U_T As Σ is an invertible squared diagonal matrix,

$$w = -V(\Sigma^T \Sigma)^{-1} \Sigma^T U^T J^T r(w) = -V(\Sigma)^{-1} (\Sigma^T)^{-1} \Sigma^T U_T J^T r(w) = -V \Sigma^{-1} U_T J^T r(w)$$

Simply, we can write it as

$$w = -V \Sigma^{-1} U_T J^T\, r(w)$$

Ex 1.5

Algorithm

Ex 1.6

1. First, we can use SVD to $X^T X$ with the help of to get Σ, V, and then U_T which we only need to calculate the first n columns.
2. Calculate the direction as $w = -V \Sigma^{-1} U_T J^T\, r(w)$ and step
3. repeat until $w = -V \Sigma^{-1} U_T J^T\, r(w) = 0$

Ex 1.7

Each step is more accurate than gradient descent as it consider more terms in Taylor expansion. So, it means few iterations. Besides, SVD is calculated once in the whole procedure, but the cost of each iteration is almost same. So, it is better.

Ex 2

Ex 2.1

Algorithm 1: Sequential Algorithm

```
input  : An array $A[i]$ of length $n$
output: The xor value of the array
1 s ← 0;
2 for $i$ ← 1 to $n$ do
3     $s \leftarrow s \oplus A[i]$;
4 end
5 return $s$
```

(b) The work of the algorithm is $O(n)$

Ex 2.2

Algorithm 2: Paralleled Algorithm

```
input  : An array $A[i]$ of length $n = 2^l$
output: The xor value of the array

s ← 0;
for $i \leftarrow l$ to $1$ do
    parfor $j \leftarrow 1$ to $2^{i-1}$ do
        $A[j] \leftarrow A[j] \oplus A[2^{i-1} + j]$;
    end
end
return $A[1]$
```

(b) Work: $O(n)$, depth: $O(\log n) = O(l)$
(c) $T_1 = O(n)$ and $T(\infty) = O(l)$, so

$$O(\frac{n}{p}) \leq T(p) \leq O(\frac{n}{p} + \log n)$$

Ex 3

based upon Distributed-gradient-descent-with-PySpark.

Method	Iteration	Time (s)	Train Acc.	Test Acc.
BGD	100	13.5	0.762	0.764
BGD-B	100	12.7	0.762	0.764
SGD	100	7.7	0.763	0.763
SGD-H	100	7.9	0.754	0.758
StGD	10	6.2	0.763	0.762
BGD	10	11.7	0.617	0.618
StGD-B	10	6.2	0.763	0.762

1. The benefits from broadcast (ended with **-B**) is very limited as the data is relatively small if we only focus on the first 5 principal components.
2. SGD computes almost two times faster than GD.
3. Steepest GD iterates 10 times can achieve the effect of GD with 100 iterations. If both 10 iterations, Steepest GD (StGD) is 10% better.