IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Optimisation: Coursework 1

Authors:
James Dorricott (CID: 01009425)
Richard Mardiat (CID: 01159059)

Date: February 17, 2020

1 Part 1

1.1 Proof 1

Claim: Log-sum-exp is convex.
Proof. We prove this by decomposing the Hessian, and showing that $v^T \nabla^2 f(x) v \geq 0$ for all $x, v \in \mathbb{R}^n$, where $f(x) = \log(\sum_{i=1}^n \exp(x_i))$ and $x = (x_1, ..., x_n)^T$, using the Cauchy-Schwartz inequality. First, note that

$$\nabla f(x) = (\frac{\exp(x_1)}{\sum_{i=1}^n \exp(x_i)}, ..., \frac{\exp(x_n)}{\sum_{i=1}^n \exp(x_i)})^T \tag{1}$$

and that

$$\nabla^2 f(x) = [\frac{-\exp(x_i)\exp(x_j)}{(\sum_{k=1}^n \exp(x_k))^2}] + \frac{\exp(x_i)}{\sum_{k=1}^n \exp(x_k)} Id_n, \tag{2}$$

where the second term on the RHS of the equation above comes from the $\frac{\partial^2 f}{\partial x_i{}^2}$ and Id_n is the $n \times n$ identity matrix.
Now fix $v \in \mathbb{R}^n$, let $w_i = \exp(x_i)$, and consider the inner-product of v with itself with respect to the Hessian of f using the equation above:

$$\begin{aligned} v^T(\nabla^2 f)v &= (v_1, ..., v_n)[(\frac{-w_i w_j}{(\sum_{k=1}^n w_k)^2}) + \frac{w_i}{\sum_{k=1}^n w_k} Id_n](v_1, ..., v_n)^T \\ &= \sum_{i=1}^n \sum_{j=1}^n \frac{-w_i w_j v_i v_j + v_i^2 w_i (\sum_{k=1}^n w_k)}{(\sum_{k=1}^n w_k)^2}. \end{aligned} \tag{3}$$

Now $\sum_{i=1} \sum_{j=1} w_i w_j v_i v_j = (\sum_{l=1} w_l v_l)^2$. And since the denominator in the equation above is nonnegative (it's a square), it follows that the inner-product of v with itself with respect to the Hessian of f is non-negative if and only if the numerator in the equation above is positive. Thus we rewrite the equation above and obtain:

$$v^T(\nabla^2 f)v = \frac{(\sum_{l=1}^n v_l^2 w_l)(\sum_{l=1}^n w_l) - (\sum_{l=1}^n w_l v_l)^2}{(\sum_{l=1}^n w_l)^2}. \tag{4}$$

But the numerator in the equation above is nonnegative by the Cauchy-Schwartz inequality. And since the denominator is also nonegative, it follows that

$$v^T(\nabla^2 f)v \geq 0, \tag{5}$$

for all $v \in \mathbb{R}^n$. This proves that the Log-sum-exp is convex.

1.2 Proof 2

Claim: The composition of a convex function with an affine mapping is convex.
Proof. Let f be a convex function and let $g(x) = b^T x + c$ be affine. We show that $h(x) = f(g(x))$ is convex.

Choose $\alpha \in (0,1)$, and pick x and y in $\mathbb{R}^n$. Then by the definition of g, h, and the convexity of f, we have:

$$\begin{aligned} h(\alpha x + (1-\alpha)y) &= f(g(\alpha x + (1-\alpha)y)) \\ &= f(\alpha b^T x + \alpha c + (1-\alpha)b^T y + (1-\alpha)c) \\ &= f(\alpha g(x) + (1-\alpha)g(y)) \\ &\leq \alpha f(g(x)) + (1-\alpha)f(g(y)) \\ &= \alpha h(x) + (1-\alpha)h(y), \end{aligned} \tag{6}$$

which proves that h is convex.

1.3 Proof 3

Claim: Let $g(x) = b^T x + c$ be an affine function from $\mathbb{R}^n$ to $\mathbb{R}$. Then g is convex.
Proof. Choose $x, y \in \mathbb{R}^n$ and $\alpha \in (0,1)$. Then by the definition of g,

$$\begin{aligned} g(\alpha x + (1-\alpha)y) &= \alpha b^T x + (1-\alpha)b^T y + c \\ &= \alpha b^T x + \alpha c + (1-\alpha)b^T y + (1-\alpha)c \\ &= \alpha g(x) + (1-\alpha)g(y) \\ &\leq \alpha g(x) + (1-\alpha)g(y), \end{aligned} \tag{7}$$

showing the convexity of g. This proves the claim.

1.4 Proof 4

Claim: The L1 norm is convex.
Proof. Choose $x, y \in \mathbb{R}^n$ and $\alpha \in (0,1)$. Then by the definition of the L1 norm, the absolute value function, and using the triangle identity, we have

$$\begin{aligned} \|\alpha x + (1-\alpha)y\|_1 &= \sum_{i=1}^{n} |\alpha x_i - (1-\alpha)y_i| \\ &\leq \sum_{i=1}^{n} \alpha|x_i| + (1-\alpha)|y_i| \\ &= \alpha \sum_{i=1}^{n} |x_i| + (1-\alpha) \sum_{i=1}^{n} |y_i| \\ &= \alpha\|x\|_1 + (1-\alpha)\|y\|_1. \end{aligned} \tag{8}$$

This proves the convexity of the L1 norm.

1.5 Proof 5

Claim: The optimization problem is convex.

Proof. The problem is convex if and only if the objective function and all the constraints are convex. Since no constraints explicitly appear, we need only show the objective function is convex. But by Proof 1, the log-exp-sum is convex. By Proof 4, the regularization term, the L1 norm, is also convex. Finally, we showed in class, and the proof is in the course slides, that a sum of convex functions is convex. And these observations together imply the convexity of the objective function, proving that the problem is convex.

2 Part 2

2.1 Demonstration 1

Claim: The L1 norm, $\|\beta_k\|_1$, is not $\mathcal{C}^1$. Proof. By definition,

$$\|\beta_k\|_1 = \sum_{i=1}^{n} |\beta_{i,k}|. \tag{9}$$

But the absolute value function appearing in the sum above is not differentiable at 0, and, therefore, the sum defining the L1 norm is not differentiable at 0. Hence, $\|\beta_k\|_1 \notin \mathcal{C}^1$, because $\mathcal{C}^1$ functions are everywhere differentiable with continuous derivatives. This proves the claim.

2.2 Demonstration 2

First note that since the logarithm, the exponential, and affine functions are all $\mathcal{C}^1$, it follows that sums of their composition are $\mathcal{C}^1$. Now because the squared L2 norm of a vector is a sum of squares of its components, it is also $\mathcal{C}^1$, and any sum of $\mathcal{C}^1$ functions is $\mathcal{C}^1$. Hence the objective function, being a sum of such functions, is also $\mathcal{C}^1$.
Moreover, the problem is convex if $\|\beta\|_2^2$ is. This is because we showed in class (1) that the sum of convex functions is convex; (2) by proof (1) that Log-sum-exp is convex; and (3) by proof (3) that affine functions are convex. Hence if the L2 norm is convex, then the objective function is a sum of convex functions which is therefore convex. To prove the convexity of the squared L2 norm, choose $\alpha \in (0,1)$, and observe that

$$\begin{aligned} \nabla\|x\|_2^2 &= \nabla \sum_{i=1}^{n} x_i^2 \\ &= (2x_1, ..., 2x_n)^T \end{aligned} \tag{10}$$

and therefore that $\frac{\partial^2 \|x\|_2^2}{\partial x_i x_j} = 2$ if $i = j$ and 0 otherwise. Hence the Hessian, $\nabla^2\|x\|_2^2$, of $\|x\|_2^2$ is equal to $2Id_n$ (two times the identity matrix), so that for any $v \in \mathbb{R}^n$, $v^T\nabla^2\|x\|_2^2 v = 2(v^T Id_n v) = 2v^T v = 2\|v\|_2^2 \geq 0$.
The problem, therefore, is $\mathcal{C}^1$ and convex.

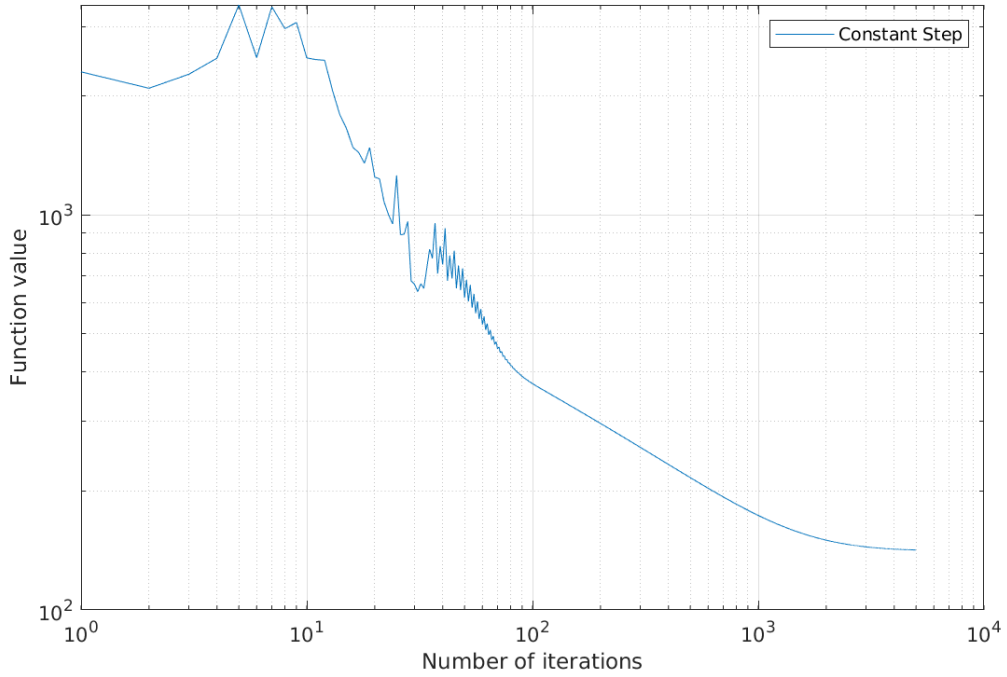

Figure 1: Gradient descent: A log-log plot of function evaluations against iterations with a constant step-size of 0.0001.

2.3 Tolerance Conditions: Implementation

Please see the MATLAB file `SolveMNIST_Gradient.m`, where the default 2-norm has been used when computing vector norms.

2.4 Link between Tolerance and Optimality Conditions

The first tolerance condition,

$$\|\nabla f(\boldsymbol{B}^{(j)})\|_2 < \epsilon, \tag{11}$$

corresponds to the FONC. That is to say that if $\nabla f(\boldsymbol{B}^{(j)}) = 0$ for some j, then $\boldsymbol{B}^{(j)}$ satisfies the FONC. In practice, we check whether the norm of the gradient is below some predefined threshold ϵ.
If we think about the second tolerance condition,

$$\|\boldsymbol{B}^{(j+1)} - \boldsymbol{B}^{(j)}\|_2 < \epsilon, \tag{12}$$

in terms of the update rule, i.e., re-writing (12) as $\| - \alpha \nabla f(\boldsymbol{B}^{(j)})\|_2$, then this also corresponds to the FONC, and is in fact a more strict stopping criterion if $\alpha < 1$.
Finally, the last tolerance condition,

$$|f(\boldsymbol{B}^{(j+1)}) - f(\boldsymbol{B}^{(j)})| < \epsilon, \tag{13}$$

looks at the absolute difference between function values at two successive iterations. We are unsure how to unpack (13), but believe it is another practical method of testing the FONC.

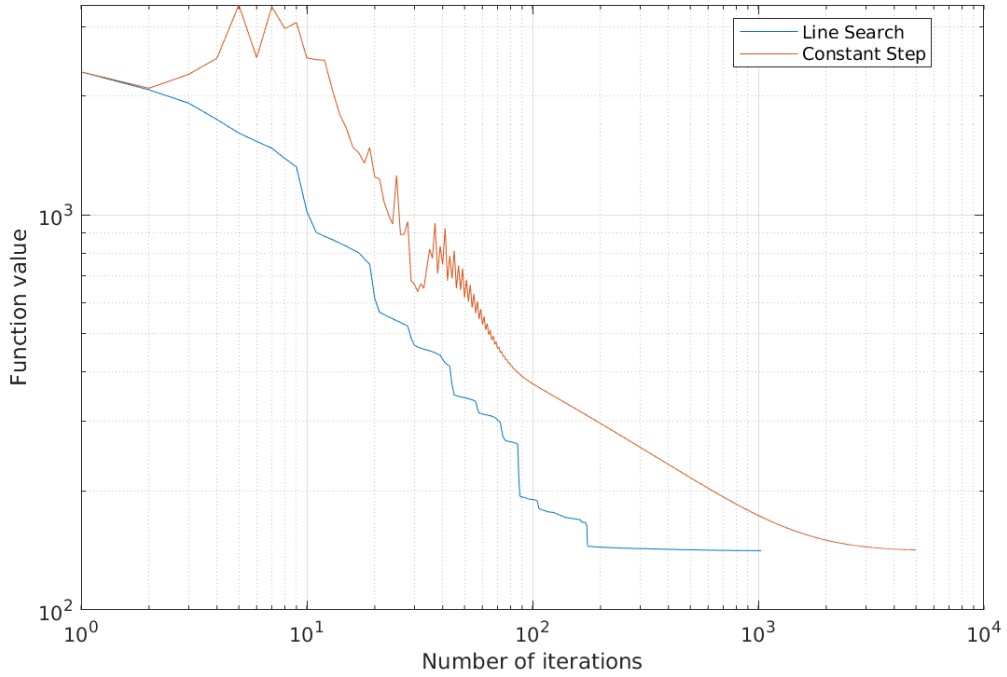

Figure 2: Gradient descent: A log-log plot of function evaluations against iterations with an adaptive step-size. This step-size was found with the Backtracking algorithm.

To test both the SONC and SOSC, we would need to characterise the Hessian, which requires that the problem is $\mathcal{C}^2$, which is not stated. However, because we have proved that the problem is convex, any local minimum is also global minimum. Hence, we simply need to meet the FONC, and (11), (12) and (13) are three ways of implementing this is practice.

2.5 Gradient Descent with Constant Step: Implementation

Please see the MATLAB file `SolveMNIST_Gradient.m`,where gradient descent has been implemented, with a constant step-size $\alpha = 0.0001$ and according to the iterative procedure

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha \nabla f(\boldsymbol{x}^k). \tag{14}$$

Here, $\boldsymbol{x}$ corresponds to the concatenated vector of decision variables/feature weights in matrix $\boldsymbol{B} \in \mathbb{R}^{nxd}$, while $\nabla f(\boldsymbol{x}^k)$ is the direction of *steepest* descent.
Figure 1 shows the result of this implementation. While the algorithm converges on the minimum (approx. 141.30) of $f(\boldsymbol{B})$, the function evaluations are noisy until around iteration 2,000, after which they decrease linearly in logarithmic terms.

2.6 Exact Line Search: Formulation

Exact line searches are those that yield the optimal step-size $\alpha_k \geq 0$ for some function $f : \mathbb{R}^n \to \mathbb{R}$ we wish to minimise. More specifically, α_k is chosen to minimise $\phi_k(\alpha) =$

$f(\boldsymbol{x}^k + \alpha \boldsymbol{d}^k)$, where the vector $\boldsymbol{d}^k$ is called the search direction. We can write this as the following optimisation problem:

$$\alpha_k = \arg\min_{\alpha \geq 0} \phi_k(\alpha) \tag{15}$$

The step-size can be determined exactly by setting $\phi_k'(\alpha) = 0$, or inexactly through methods such as the Percentage Test, Armijo's Rule, Backtracking, etc.

2.7 Line Search Strategy: Implementation

For our inexact line search, we chose to implement the Backtracking algorithm to find the best step-size at each iteration of gradient descent. Backtracking works by updating the step-size until Armijo's criterion is met. That is, at each iteration k, reset $\alpha_k = 1$ and perform the following test for some $0 < c_\alpha < 1$:

$$f(\boldsymbol{x}^k) - c_\alpha \alpha_k \| f(\nabla \boldsymbol{x}^k) \|_2^2 \leq f(\boldsymbol{x}^k - \alpha_k \nabla f(\boldsymbol{x}^k)) \tag{16}$$

If the condition is not met then shrink α_k by the factor c_β. Once the loop has terminated and a suitable $\alpha = \alpha_k$ has been found, this step-size is used to perform the next update of gradient descent:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha \nabla f(\boldsymbol{x}^k). \tag{17}$$

Please see the MATLAB file `backtracking.m` for our implementation, where values of $c_\alpha = 0.5$ and $c_\beta = 1/1.2$ have been used. In Figure 2 we show the result of this implementation against the constant step-size strategy discussed above. We note that gradient descent with Backtracking terminates at iteration 1,029, at which point the tolerance condition in (13) is met. This is a clear improvement over gradient descent with a constant step-size, which used the full 5,000 number of iterations to reach the minimum. What's nice about Backtracking is that the resulting step-size ensures a sufficient decrease in the objective function $\phi_k(\alpha)$, while preventing it from being *too* short, given the accepted value is within a fraction of the last trial value. However, this strategy comes at the cost of increased computation and a dependence on the two parameters c_α and c_β.

2.8 Running the Code

To execute gradient descent with a fixed step-size, please comment out lines `48-50` and uncomment line `45` of `SolveMNIST_Gradient.m`. To execute gradient descent with an adaptive step-size, do the reverse.