Imperial College London

COURSEWORK 2

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Mathematics for Machine Learning

Author:
Alexander Gaskell (CID: 01813313)

Date: October 27, 2019

1 Differentiation

1.a

[3 marks] Write $f_1(x)$ in the completed square form $(\mathbf{x}-\mathbf{c})^{\mathbf{T}}\mathbf{C}(\mathbf{x}-\mathbf{c})+\mathbf{c_0}$, i.e., determine $\mathbf{C}, \mathbf{c}, \mathbf{c_0}$.

$$f_1(x) = \mathbf{x}^{\mathbf{T}}\mathbf{B}\mathbf{x} - \mathbf{x}^{\mathbf{T}}\mathbf{x} + \mathbf{a}^{\mathbf{T}}\mathbf{x} - \mathbf{b}^{\mathbf{T}}\mathbf{x} \tag{1}$$

$$= \mathbf{x}^{\mathbf{T}}(\mathbf{B}-\mathbf{I}_2)\mathbf{x} + \mathbf{a}^{\mathbf{T}}\mathbf{x} - \mathbf{b}^{\mathbf{T}}\mathbf{x} \tag{2}$$

Objective: write equation 2 in completed square form. We can expand the completed square form:

$$(\mathbf{x}-\mathbf{c})^{\mathbf{T}}\mathbf{C}(\mathbf{x}-\mathbf{c}) + \mathbf{c_0} = \mathbf{x}^{\mathbf{T}}\mathbf{C}\mathbf{x} - \mathbf{c}^{\mathbf{T}}\mathbf{C}\mathbf{x} - \mathbf{x}^{\mathbf{T}}\mathbf{C}\mathbf{c} + \mathbf{c}^{\mathbf{T}}\mathbf{C}\mathbf{c} + \mathbf{c_0} \tag{3}$$

Equating coefficients on $\mathbf{x}^{\mathbf{T}}\mathbf{x}$ term gives:

$$\mathbf{C} = \mathbf{B} - \mathbf{I}_2 = \begin{bmatrix} 3 & -2 \\ -2 & 3 \end{bmatrix} \tag{4}$$

We can exploit the symmetry of $\mathbf{C}$ to write equation 3 as:

$$(\mathbf{x}-\mathbf{c})^{\mathbf{T}}\mathbf{C}(\mathbf{x}-\mathbf{c}) + \mathbf{c_0} = \mathbf{x}^{\mathbf{T}}\mathbf{C}\mathbf{x} - 2\mathbf{c}^{\mathbf{T}}\mathbf{C}\mathbf{x} + \mathbf{c}^{\mathbf{T}}\mathbf{C}\mathbf{c} + \mathbf{c_0} \tag{5}$$

Comparing coefficients on $\mathbf{x}$ terms in equations 2 and 5 we can see that:

$$-2\mathbf{c}^{\mathbf{T}}\mathbf{C} = \mathbf{a}^{\mathbf{T}} - \mathbf{b}^{\mathbf{T}} \tag{6}$$

Solve for $\mathbf{c}$:

$$\mathbf{c} = -\frac{1}{2}\mathbf{C}^{-\mathbf{1}}(\mathbf{a}-\mathbf{b}) \tag{7}$$

$$= -\frac{1}{2}\left(\tfrac{1}{5}\begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}\right)\begin{bmatrix} 2 \\ 0 \end{bmatrix} \tag{8}$$

$$= \begin{bmatrix} -3/5 \\ -2/5 \end{bmatrix} \tag{9}$$

Comparing equations 2 and 5 we can solve for c_0:

$$\mathbf{c_0} = -\mathbf{c}^{\mathbf{T}}\mathbf{C}\mathbf{c} \tag{10}$$

$$= -3/5 \tag{11}$$

Hence:

$$\mathbf{C} = \begin{bmatrix} 3 & -2 \\ -2 & 3 \end{bmatrix} \quad , \quad \mathbf{c} = \begin{bmatrix} -3/5 \\ -2/5 \end{bmatrix} \quad , \quad \mathbf{c_0} = -3/5 \tag{12}$$

1.b

[2 marks] Explain how you can tell that f_1 has a minimum point. State the minimum value of f_1 and find the input which achieves this minimum.

Begin by computing the gradient of $f_1(x)$ by use of the product rule:

$$\frac{df_1}{d\mathbf{x}} = g^T \frac{\partial h}{\partial \mathbf{x}} + h^T \frac{\partial g}{\partial \mathbf{x}} \quad \text{where} \quad h(\mathbf{x}) = \mathbf{x} - \mathbf{c}, \quad g(\mathbf{x}) = \mathbf{C}(\mathbf{x} - \mathbf{c}) \tag{13}$$

$$= (\mathbf{x} - \mathbf{c})^{\mathbf{T}} \mathbf{C}^{\mathbf{T}} + (\mathbf{x} - \mathbf{c})^{\mathbf{T}} \mathbf{C} \tag{14}$$

$$= 2(\mathbf{x} - \mathbf{c})^{\mathbf{T}} \mathbf{C} \quad \in \mathbb{R}^{1x2} \tag{15}$$

The final step exploits the symmetry of $\mathbf{C}$. If f_1 has a minimum, then the Hessian will be positive definite. Hence we need to find the Hessian and check if it is positive definite:

$$\mathbf{H} = \left[\frac{\partial^2 f}{\partial \mathbf{x}^2}\right] \tag{16}$$

$$= \frac{\partial}{\partial \mathbf{x}} (2(\mathbf{x} - \mathbf{c})^{\mathbf{T}} \mathbf{C}) \tag{17}$$

$$= 2\mathbf{C}^{\mathbf{T}} \quad \in \mathbb{R}^{2x2} \tag{18}$$

We can compute the Eigenvalues of $\mathbf{H}$ to be 10 and 2. As both are positive, the Hessian is positive definite, so f_1 will have a minimum point.

The minimum is found by setting the gradient to zero:

$$\frac{df_1}{d\mathbf{x}} = 0 = 2(\mathbf{x} - \mathbf{c})^{\mathbf{T}} \mathbf{C} \tag{19}$$

$$\iff \mathbf{x} = \mathbf{c} = \begin{bmatrix} -3/5 \\ -2/5 \end{bmatrix} \tag{20}$$

At the minimum point:

$$f_1(\mathbf{x}) = \mathbf{c_0} = -3/5 \tag{21}$$

1.c

[6 marks] Write three python functions grad_f1(x), grad_f2(x) and grad_f3(x) that return the gradient for each of the functions above. All functions must accept numpy (2,) array inputs and return numpy (2,) outputs.

As found in 1.b, the gradient of f_1 is:

$$\frac{df_1}{d\mathbf{x}} = 2(\mathbf{x}-\mathbf{c})^{\mathbf{T}}\mathbf{C} \quad \in \mathbb{R}^{1x2} \tag{22}$$

The gradient of f_2 is:

$$\frac{df_2}{d\mathbf{x}} = -\sin\Big((\mathbf{x}-\mathbf{b})^{\mathbf{T}}(\mathbf{x}-\mathbf{b})\Big)\Big(2(\mathbf{x}-\mathbf{b})^{\mathbf{T}}\Big) + 2(\mathbf{x}-\mathbf{a})^{\mathbf{T}}\mathbf{B} \quad \in \mathbb{R}^{1x2} \tag{23}$$

The gradient of f_3 is:

$$\begin{aligned}\frac{df_3}{d\mathbf{x}} &= \exp\Big((\mathbf{x}-\mathbf{a})^{\mathbf{T}}(\mathbf{x}-\mathbf{a})\Big)\Big(2(\mathbf{x}-\mathbf{a})^{\mathbf{T}}\Big) + \exp\Big((\mathbf{x}-\mathbf{b})^{\mathbf{T}}\mathbf{B}(\mathbf{x}-\mathbf{b})\Big)\Big(2(\mathbf{x}-\mathbf{b})^{\mathbf{T}}\mathbf{B}\Big) \\ &\quad + \frac{1}{10}\frac{2}{\mathbf{x}^{\mathbf{T}}\mathbf{x}+\frac{1}{100}}\mathbf{x}^{\mathbf{T}} \quad \in \mathbb{R}^{1x2}\end{aligned} \tag{24}$$

The gradient for f_3 takes $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and exploits the fact that the log determinant term of f_3 can be written as:

$$\log\det\Big(\tfrac{1}{100}I_2 + \mathbf{x}\mathbf{x}^{\mathbf{T}}\Big) = \log\left(\begin{bmatrix} x_1^2+\frac{1}{100} & x_1x_2 \\ x_2x_1 & x_2^2+\frac{1}{100} \end{bmatrix}\right) \tag{25}$$

$$= \log\Big((x_1^2+\tfrac{1}{100})(x_2^2+\tfrac{1}{100}) - x_1^2x_2^2\Big) \tag{26}$$

$$= \log\Big(\tfrac{1}{100}(x_1^2+x_2^2+\tfrac{1}{100})\Big) \tag{27}$$

$$= \log\Big(\tfrac{1}{100}(\mathbf{x}^{\mathbf{T}}\mathbf{x}+\tfrac{1}{100})\Big) \tag{28}$$

1.d

[4 marks] Use your gradients to implement a gradient descent algorithm with 50 iterations to find a local minimum for both f_2 and f_3. Show the steps of your algorithm on a contour plot of the function. Start from the point (0.3, 0) and state the step size you used. Produce separate contour plots for the two functions, using first component of x on the x axis and the second on the y.

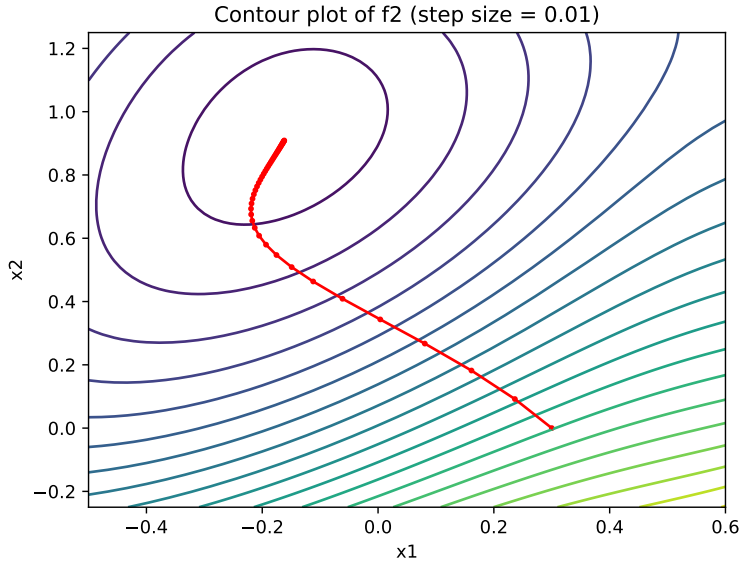

Figure 1: Contour plot for f_2 with gradient descent

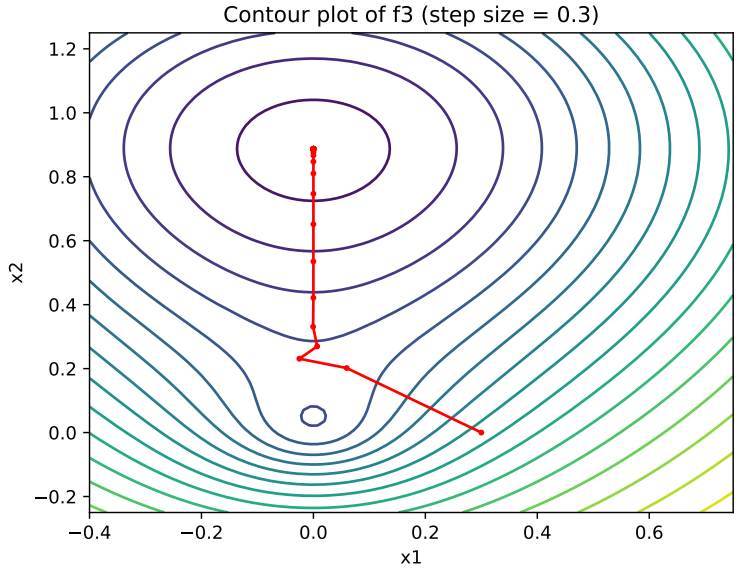

Figure 2: Contour plot for f_3 with gradient descent

1.e

[5 marks] For the two functions f2 and f3:

- **Discuss the qualitative differences you observe when performing gradient descent with step sizes varying between 0.01 and 1, again starting the point (0.3, 0)**
- **Briefly describe also what happens in the two cases with grossly mis-specified step-sizes (i.e. greater than 1).**

Varying the step size between 0.01 and 1 illustrates the careful balance that must be struck when choosing the step size for gradient descent: if the learning rate is too small, the function will take a large number of training iterations to converge and amy get stuck at local minima; if the learning rate is too large, gradient descent will overshoot the minimum point, meaning a long time to convergence or even divergence in some cases.

In figure 1, plotted gradient descent using a step size of 0.01 is plotted for f_2. This is an appropriate value to choose as we have a sufficient number of training iterations to converge to the (global) minimum. However, a learning rate of 0.01 is inappropriate to use for f_3 for two reasons: the step size is too small for convergence of any kind (given 50 training iterations the path does not reach any minimum point). Additionally, f_3 gets stuck at a local minimum (located at approx $(0.00, 0.05)$) with any learning rate less than approx 0.25. Hence having a learning rate which is too low can make gradient descent liable to getting stuck at local minima and expensive as more training iterations are required for convergence.

Having a step size which is too large presents other difficulties: gradient descent can overshoot the minimum point meaning more training iterations until convergence, with no convergence/divergence possible in some circumstances. For example, once learning rate is between 0.6 and 1 for f_3, then the path of convergence is very jagged and oscillates around the minimum point. Similar behaviour is seen from f_2 if we set the step size around 0.1. However, once the learning rate is above 0.175 for f_2, gradient descent exhibits divergence (i.e. oscillating around the minimum point but overshooting so dramatically with each step that the minimum gets further away with each iteration). This divergence behaviour becomes even more severe when we increase the step size further.

For f_2, if we severely mis-specify the step size (e.g. set = 2), then we see simply a more extreme version of the divergence described above. The point where divergence begins occurring is 0.175, so if the step size is larger than this we will see divergence. For f_3, we do not see divergence in the same way as we saw for f_2, even if the step-size is inappropriately high. For example, when we set the learning rate to 10, gradient descent massively overshoots the minimum and oscillates around it without properly converging; however, the gradient does not explode in the same way as it does for f_2.