Gradient Descent

Motivation

In Linear Regression, we studied how to minimize the error function

$$E = \sum_{j=1}^{N} (y_j - \sum_{s=1}^{k+1} x_{js} m_s)^2$$

and obtained an exact solution. In other cases we will encounter later, such an exact solution is not feasible and we will have to use a method to approximate an exact solution. One of the most common methods for this purpose is *gradient descent*.

Basic Algorithm

Consider a function $E : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{w} = (w_1, w_2, \ldots, w_n) \to E(\boldsymbol{w})$. The *gradient* ∇E of E is defined by

$$\nabla E := \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n} \right).$$

Proposition : *Assume that $E(\boldsymbol{w})$ is differentiable in a neighborhood of $\boldsymbol{w}$. Then the function $E(\boldsymbol{w})$ decreases fastest in the direction of $-\nabla E(\boldsymbol{w})$.*

Proof: For a unit vector $\boldsymbol{u}$, the directional derivative $D_{\boldsymbol{u}}E$ is given by

$$D_{\boldsymbol{u}}E = \nabla E \cdot \boldsymbol{u} = |\nabla E||\boldsymbol{u}| \cos\theta = |\nabla E| \cos\theta,$$

where θ is the angle between ∇E and $\boldsymbol{u}$. The minimum value of $D_{\boldsymbol{u}}E$ occurs when $\cos\theta$ is -1. $\square$

Set

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \nabla E(\boldsymbol{w}_k)$$

where $\eta > 0$ is the step size or *learning rate*. Then

$$E(\boldsymbol{w}_{k+1}) \leq E(\boldsymbol{w}_k).$$

Under some moderate conditions,

$$E(\boldsymbol{w}_k) \to \text{local minimum} \qquad \text{as} \quad k \to \infty.$$

In particular, this is true when E is convex or when ∇E is Lipschitz continuous.

Example

Consider $E(\boldsymbol{w}) = E(w_1, w_2) = w_1^4 + w_2^4 - 16w_1w_2$. Then $\nabla E(\boldsymbol{w}) = [4w_1^3 - 16w_2, 4w_2^3 - 16w_1]$. Choose $\boldsymbol{w}_0 = (1, 1)$ and $\eta = 0.01$. Then

$$\boldsymbol{w}_{30} = (1.99995558586289, 1.99995558586289)$$

and we get

$$E(\boldsymbol{w}_{30}) = -31.9999999368777.$$

We see that $\boldsymbol{w}_k \to (2, 2)$ and $E(2, 2) = -32$. Indeed, using multi-variable calculus, one can verify that when $\boldsymbol{w} = (2, 2)$, a local minimum of $E(\boldsymbol{w})$ is -32.

Exercise: Using multi-variable calculus, find all the local minima of $E(\boldsymbol{w})$.

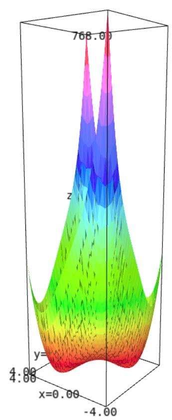

Figure 1: Graph of $E(\boldsymbol{w})$

Example: Linear Regression revisited

We will apply gradient descent to linear regression in the lab session.

Exercise: Define $\sigma(x) = \dfrac{e^x}{e^x + 1} = \dfrac{1}{1 + e^{-x}}$. In Logistic Regression we will minimize the following error function

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\},$$

where we write $\boldsymbol{w} = (w_1, w_2, \ldots, w_{k+1})$ and $y_n = \sigma(w_1 x_{n1} + w_2 x_{n2} + \cdots + w_k x_{nk} + w_{k+1})$. Compute the gradient $\nabla E(\boldsymbol{w})$.

Newton's Method

Let us first consider the single-variable case.

k	w1	w2	E(w1,w2)
1	1.12000000000000	1.12000000000000	-16.9233612800000
2	1.24300288000000	1.24300288000000	-19.9465014818312
3	1.36506297054983	1.36506297054983	-22.8698545020842
4	1.48172688079195	1.48172688079195	-25.4876645161458
5	1.58867706472624	1.58867706472624	-27.6422269714610
6	1.68247924276483	1.68247924276483	-29.2656452783487
7	1.76116971206054	1.76116971206054	-30.3861816086105
8	1.82445074094736	1.82445074094736	-31.0984991504577
9	1.87344669354831	1.87344669354831	-31.5194128485897
10	1.91018104795404	1.91018104795404	-31.7533053700606
11	1.93701591038872	1.93701591038872	-31.8770223901250
12	1.95622873443784	1.95622873443784	-31.9400248989010
13	1.96977907222858	1.96977907222858	-31.9712142030755
14	1.97923168007769	1.97923168007769	-31.9863406140263
15	1.98577438322011	1.98577438322011	-31.9935701975589
16	1.99027812738069	1.99027812738069	-31.9969902100773
17	1.99336647981957	1.99336647981957	-31.9985965516271
18	1.99547865709166	1.99547865709166	-31.9993473166738
19	1.99692058430943	1.99692058430943	-31.9996970174121
20	1.99790372262623	1.99790372262623	-31.9998595272283
21	1.99857347710339	1.99857347710339	-31.9999349274762
22	1.99902947615421	1.99902947615421	-31.9999698732955
23	1.99933981776146	1.99933981776146	-31.9999860577045
24	1.99955097148756	1.99955097148756	-31.9999935493971
25	1.99969461222478	1.99969461222478	-31.9999970160815
26	1.99979231393118	1.99979231393118	-31.9999986198712
27	1.99985876312152	1.99985876312152	-31.9999993617137
28	1.99990395413526	1.99990395413526	-31.9999997048203
29	1.99993468659806	1.99993468659806	-31.9999998634976
30	1.99995558586289	1.99995558586289	-31.9999999368777

Figure 2: Values of $E(\boldsymbol{w}_k)$

- Let $f : \mathbb{R} \longrightarrow \mathbb{R}$ be a single-variable (convex, differentiable) function.
- To find a local minimum $\iff$ To find x^* such that $f'(x^*) = 0$
  Make a guess x_0 for x^* and set $x = x_0 + h$.
- Using the Taylor expansion, we have

$$
\begin{aligned}
f(x) &= f(x_0 + h) \approx f(x_0) + f'(x_0)h + \tfrac{1}{2} f''(x_0)h^2 \\
f'(x) = \frac{df}{dh}\frac{dh}{dx} &\approx \frac{d}{dh}\left(f(x_0) + f'(x_0)h + \tfrac{1}{2} f''(x_0)h^2 \right) \\
&= f'(x_0) + f''(x_0)h
\end{aligned}
$$

  If $0 = f'(x_0) + f''(x_0)h$, we obtain

$$h = -f'(x_0)/f''(x_0).$$

- We have shown that

$$x_1 = x_0 - f'(x_0)/f''(x_0)$$

  is an approximation of x^*.
- Repeat the process to obtain

$$\boxed{x_{k+1} = x_k - f'(x_k)/f''(x_k)},$$

  and $x_k \to x^*$ as $k \to \infty$.

Now we consider the multi-variable case.

- Let $E(\boldsymbol{w})$ be a multi-variable function. The *Hessian matrix* of E is defined by

$$\mathbf{H}E = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_n} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_n \partial w_1} & \frac{\partial^2 E}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_n^2} \end{bmatrix}.$$

  That is, $\mathbf{H}E = [\frac{\partial^2 E}{\partial w_i \partial w_j}]$.

- Generalizing the single-variable case, we obtain

$$\boxed{\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \mathbf{H}E(\boldsymbol{w}_k)^{-1} \nabla E(\boldsymbol{w}_k)}.$$

- Using a step size η, the formula may be modified to be

$$\boxed{\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \mathbf{H}E(\boldsymbol{w}_k)^{-1} \nabla E(\boldsymbol{w}_k)}.$$

- Newton's method is much faster than Gradient Descent. However, it may be expensive to compute the inverse of the Hessian matrix.

Example

- Consider $E(\boldsymbol{w}) = E(w_1, w_2) = w_1^4 + w_2^4 - 16 w_1 w_2$. Then $\nabla E(\boldsymbol{w}) = [4w_1^3 - 16w_2, 4w_2^3 - 16w_1]^\top$.

$$\mathbf{H}E(\boldsymbol{w}) = \begin{bmatrix} 12w_1^2 & -16 \\ -16 & 12w_2^2 \end{bmatrix}$$

$$\mathbf{H}E(\boldsymbol{w})^{-1} = \frac{1}{9w_1^2 w_2^2 - 16} \begin{bmatrix} \frac{3}{4}w_2^2 & 1 \\ 1 & \frac{3}{4}w_1^2 \end{bmatrix}$$

$$\mathbf{H}E^{-1} \nabla E = \frac{1}{9w_1^2 w_2^2 - 16} \begin{bmatrix} 3w_1^3 w_2^2 - 8w_2^3 - 16w_1 \\ 3w_1^2 w_2^3 - 8w_1^3 - 16w_2 \end{bmatrix}$$

- Choose $\boldsymbol{w}_0 = (1.2, 1.2)$ and $\eta = 1$. Then $\boldsymbol{w}_9 = (2.00000004189571, 2.00000004189571)$, $E(\boldsymbol{w}_9) = -31.9999999999999$.

Stochastic Gradient Descent (SGD)

Typically in Machine Learning, the function $E(\boldsymbol{w})$ is given by a sum of the form

$$E(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} E_n(\boldsymbol{w}),$$

k	w1	w2	E(w1,w2)
0	1.20000000000000	1.20000000000000	-18.8928000000000
1	10.8000000000000	10.8000000000000	25343.5392000001
2	7.28325624421832	7.28325624421832	4778.98521693644
3	4.98069646698406	4.98069646698406	833.890570717962
4	3.50906808575457	3.50906808575457	106.230520855080
5	2.62345045192591	2.62345045192591	-15.3824765840014
6	2.16920289601164	2.16920289601164	-31.0047054152139
7	2.01793795417254	2.01793795417254	-31.9896107961456
8	2.00023638179330	2.00023638179330	-31.9999982117454
9	2.00000004189571	2.00000004189571	-31.9999999999999
10	2.00000000000000	2.00000000000000	-32.0000000000000

Figure 3: Values of $E(\boldsymbol{w}_k)$

where N is the number of elements in the training set. When N is large, computation of the gradient ∇E may be expensive.

The SGD selects a sample from the training set in each iteration step instead of using the whole batch of the training set, and use

$$\frac{1}{M}\sum_{i=1}^{M} \nabla E_{n_i}(\boldsymbol{w}),$$

where M is the size of the sample and $\{n_1, n_2, \ldots, n_M\} \subset \{1, 2, \ldots, N\}$. The SGD is commonly used in many Machine Learning algorithms.