Optimization

Let $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ be a scalar-valued function. $(f(x_1, \dots, x_d$

Gradient:

$$\underset{d \times 1}{\nabla_x f(x)} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}$$

Hessian:

$$\underset{d \times d}{\nabla_x^2 f(x)} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & & & \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \cdots & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}$$

⊛ In the case of

$f : \mathbb{R}^d \longrightarrow \mathbb{R}^m$ being a vector-valued function, i.e.:

$$f(x) = \left(f_1(x_1, \dots, x_d), f_2(x_1, \dots, x_d, \dots, f_m(x_1, x_2, \dots, x_d) \right)$$

Jacobian:

$$\underset{d \times m}{\nabla_x f(x)} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & & \\ \frac{\partial f_1}{\partial x_d} & \cdots & \frac{\partial f_m}{\partial x_d} \end{bmatrix}$$

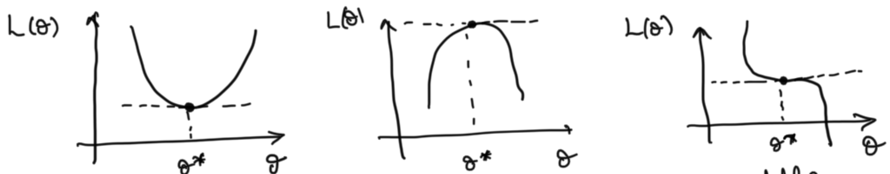
Setup: Given a model with $\overset{\text{parameters}}{\vartheta \in \mathbb{R}^d}$, i.e. $\vartheta = (\vartheta_1, \vartheta_2, \dots, \vartheta_d)$ and a loss/likelihood $\mathcal{L}(\vartheta)$, then our goal is to identify a set of parameters ϑ^* such that:

$$\vartheta^* = \arg\min_{\vartheta} \mathcal{L}(\vartheta)$$

We need to identify the critical points for which:

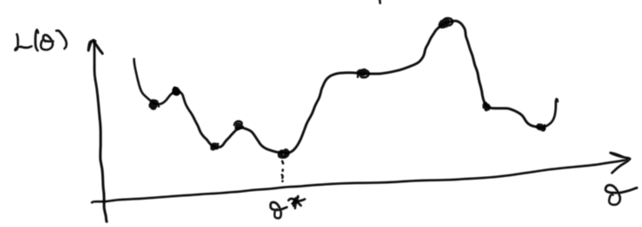
$$\nabla_\theta L(\theta) = 0 \qquad L : \mathbb{R}^d \to \mathbb{R}$$

This condition is true for: (i) minima, (ii) maxima, (iii) sadd poin



minimum maximum saddle point.

e.g. in practice:

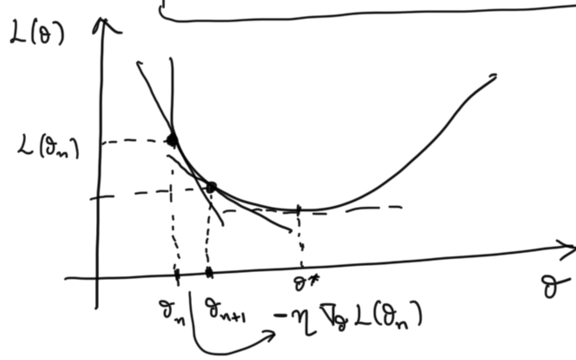


Gradient descent:

We want to minimize $L(\theta)$, i.e. $\theta^* = \arg\min_\theta L(\theta)$

Pick an initial guess θ_0 and use update rule:

Update Rule: $$\boxed{\underset{d\times 1}{\theta_{n+1}} = \underset{d\times 1}{\theta_n} - \underset{1\times 1}{\eta} \underset{d\times 1}{\nabla_\theta L(\theta_n)}} \quad \underset{\text{Forward Euler}}{\overset{\text{discretize version}}{\rightleftarrows}} \quad \underbrace{\frac{d\theta}{dt} = -\nabla_\theta L(\theta)}_{\text{gradient flow system}}$$



- This is a first-order method as it relies on a linear approximation of $L(\theta)$ around some point θ_n.
- It is guaranteed to converge to a critical point (e.g. local min / max) of $L(\theta)$, assuming that η is properly chose

η : step-size / learning rate (user need to adjust)

Newton's algorithm:

Let's use a Taylor expansion of $L(\theta)$ around θ_n:

where:

$$\underset{1\times 1}{\mathcal{L}(\theta)} \approx \mathcal{L}(\theta_n) + \underset{1\times d}{g_n^T}\,\underset{d\times 1}{(\theta-\theta_n)} + \frac{1}{2}\,\underset{1\times d}{(\theta-\theta_n)^T}\,\underset{d\times d}{H_n}\,\underset{d\times 1}{(\theta-\theta_n)}$$

where:
$g_n := \nabla_\theta \mathcal{L}(\theta_n$
$H_n := \nabla_\theta^2 \mathcal{L}($

$$\approx \mathcal{L}(\theta_n) + g_n^T(\theta-\theta_n) + \frac{1}{2}\left[\theta^T H_n \theta - 2\theta^T H_n \theta_n + \theta_n^T H_n \theta_n\right]$$

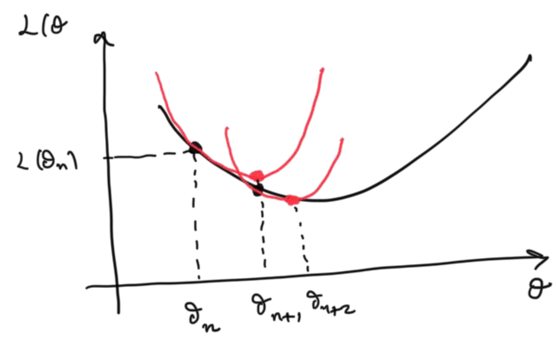
Let us now find critical points:

$$\nabla_\theta \underbrace{\mathcal{L}(\theta)} = 0 \implies 0 + g_n^T + H_n\theta - H_n\theta_n = 0$$

$$\implies \underline{\underline{\theta}} = \theta_n - H_n^{-1} g_n^T$$

critical point of $\underbrace{\text{the Taylor}}_{\text{quadratic}}$ approximation to $\mathcal{L}(\theta)$

$$\boxed{\text{Newton update rule}: \quad \theta_{n+1} = \theta_n - H_n^{-1} g_n^T}$$



- No need for tuning parameters, but extra cost for computing and inverting the Hessian.
- 2^{nd}-order algorithm, faster convergence vs gradient descent
- utilizes the underlying geometry by exploiting curvature information

Example: Linear regression

Recall the loss function for MLE estimation:

$$\mathcal{L}(\theta) := -\log p(y \mid \underline{\underline{w}}^T x, \underline{\underline{\sigma}}^2), \quad \theta := \{\underset{d\times 1}{w}, \underset{1\times 1}{\sigma^2}\}$$

Gradient: $\nabla_w \mathcal{L}(w) = -X^T y + X^T X w$

Hessian: $\nabla_w^2 \mathcal{L}(w) = X^T X$

Gradient descent: $w_{n+1} = w_n - \eta\left[-X^T y + X^T X w_n\right]$

Newton: $w_{n+1} = w_n - (X^T X)^{-1}\left[X^T X w - X^T y\right]$

⊛ Remark : Similar updates can be formulated for σ^2 !

Limitations :

- Gradient descent converges slowly. Choosing η is an art.
- ⊙ Scalibility to $\underbrace{\text{big data}}_{\substack{\text{evaluating the loss} \\ \text{over a big data-set} \\ \text{is expensive.}}}$ $\theta_{n+1} = \theta_n - \eta \nabla_\theta \underbrace{L(\theta_n)}$

  typically : $\underbrace{\propto \sum_{i=1}^{n} (y_i - w_i^T}$

- Exact Hessians are often very hard to compute $\xrightarrow{\text{appox}}$ Quasi-Newton methods.

Stochastic gradient descent (SGD)

In many ML applications the loss functions factorize across data points, i.e. they can be written as a summation over individual data points : $L(\theta) \propto \sum_{i=1}^{n} L_i(\theta)$ (e.g. linear regression

(this follows from assuming an i.i.d likelihood).

In this case, a standard "full batch" gradient descent approach would take the form :

$$\theta_{n+1} = \theta_n - \eta \underbrace{\nabla_\theta L(\theta_n)}_{\text{true gradient}} = \theta_n - \eta \underbrace{\sum_{n=1}^{n} \nabla_\theta L_i(\theta_n}$$

In stochastic gradient descent,
 the true gradient is approximated using a single example :

$$\theta_{n+1} = \theta_n - \eta \underbrace{\nabla_\theta L_i(\theta_n)}_{\substack{\text{very noisy} \\ \text{approx. to} \\ \text{the true gradient.}}} ,$$

i is chosen at rando at each iteration of SGD algorithm

A compromise between these two extremes is to approximate the true gradient over a "mini-batch" of data :

$$\theta_{n+1} = \theta_n - \eta \frac{1}{m} \sum^{m} \nabla_\theta L_i(\theta_n) \qquad m << n$$

$\theta_{n+1} = \theta_n - \eta \frac{1}{m} \sum_{i=1}^{m} \ldots$

At every iteration we randomly pick a mini-batch of data by sub-sampling our full data-set.

- A complete looping cycle over the entire data-set is called an "epoch".

Modern variants of SGD

- **SGD with momentum:**

$$\begin{cases} u_{n+1} = \gamma u_n + \eta \nabla_\theta \mathcal{L}(\theta_n), \quad u_0 = 0 \\ \theta_{n+1} = \theta_n - u_{n+1} \end{cases}, \quad \begin{cases} \eta : \text{learning rate} \\ \gamma \begin{cases} 0 = \text{SGD} \\ 0.9 \text{ is a typical value in prac.} \end{cases} \end{cases}$$

u_n : momentum variable

- **Nesterov accelerated gradient method (NAG):**

$$\begin{cases} u_{n+1} = \gamma u_n + \eta \nabla_\theta \mathcal{L}(\theta_n - \gamma u_n) \\ \theta_{n+1} = \theta_n - u^{n+1} \end{cases}$$

Adaptive learning rate approaches:

1.) **RMS prop:** $\mathbb{E}[g^2]_n$:= average of the square gradients.

$\leftarrow \mathbb{E}\left[\left(\underbrace{\nabla_\theta \mathcal{L}(\theta_n)}_{d\times 1}\right)^2\right]$

running avg of the sq. gradients

$$\begin{cases} \underbrace{\mathbb{E}[g^2]_{n+1}}_{d\times 1} = \gamma\, \mathbb{E}[g^2]_n + (1-\gamma)\, g_n^2 \\ \theta_{n+1} = \theta_n - \dfrac{\eta}{\sqrt{\mathbb{E}[g^2]_{n+1} + \underbrace{\varepsilon}_{10^{-8}}}}\, g_n \end{cases}, \quad \text{typically} \begin{cases} \gamma = 0.9 \\ \eta = 10^{-3} \end{cases}$$

2.) **Adam** (adaptive moment estimation):

$$\begin{cases} \rightarrow m_{n+1} = \beta_1 m_n + (1-\beta_1) g_n \rightarrow \text{running average estimator of the expected gradie[nt]} \\ \rightarrow V_{n+1} = \beta_2 V_n + (1-\beta_2) g_n^2 \rightarrow \text{-"- of the variance of the g[radient]} \end{cases}$$

m, v are typically initialized to be zero ...

$\cdot, v_0$ are typically initialized to be zero, hence the estimates are biased towards zero. To counteract this bias, we can consider a correction:

$$\hat{m}_n = \frac{m_n}{1 - \beta_1^n} \quad , \quad \hat{v}_n = \frac{v_n}{1 - \beta_2^n}$$

$$\left.\begin{array}{l}\text{Adam update} \\ \text{rule}\end{array}\right\} \qquad \theta_{n+1} = \theta_n - \frac{\eta}{\underbrace{\sqrt{\hat{v}_{n+1} + \varepsilon}}_{d \times 1}} \hat{m}_{n+1}$$