CM50264 Machine Learning 1, Lecture 6

Optimisation Basics 2

Xi Chen

UNIVERSITY OF BATH

Optimisation formulation

Optimisation (minimisation) problem

Given a function $f(\cdot) : \mathcal{X} \subset \mathbb{R}^n \mapsto \mathbb{R}$.

Find an element $\mathbf{x}_*$ such that

$$f(\mathbf{x}_*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}. \tag{1}$$

- Maximisation can be converted to a minimisation by multiplying $f(\cdot)$ by -1.

Constrained optimisation problem

- Optimisation problem can be accompanied by constraints:

$$
\begin{aligned}
\mathbf{x}_* = \arg\min_{\mathbf{x}\in\mathcal{X}} & f(\mathbf{x}), \\
\text{s.t. } & g_i(\mathbf{x}) \leq 0, i = \{1,\dots,k\}, \\
& h_j(\mathbf{x}) = 0, j = \{1,\dots,l\}.
\end{aligned}
$$

$\{g_i(\cdot)\}$ and $\{h_j(\cdot)\}$ are (inequality & equality) *constraint functions*.

Optimisation problem types

$$
\begin{aligned}
\mathbf{x}_* = \arg\min_{\mathbf{x}\in\mathcal{X}} & f(\mathbf{x}), \\
\text{s.t. } & g_i(\mathbf{x}) \leq 0, i = \{1, \ldots, k\}, \\
& h_j(\mathbf{x}) = 0, j = \{1, \ldots, l\}.
\end{aligned}
$$

- Constrained vs. unconstrained optimization.
- Discrete vs. continuous optimization.
- Deterministic vs. stochastic optimization.

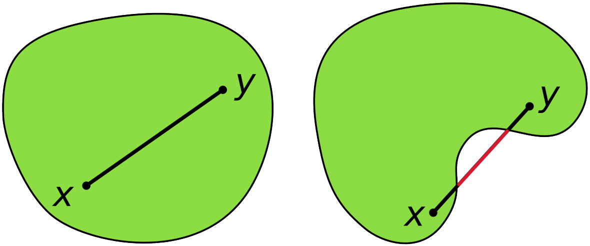
Convex

Convex set

A subset C of a vector space is called convex if $\forall \mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$

$$\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in C.$$

An example:



Convex

Convex function

A function on a convex set C is

- convex if

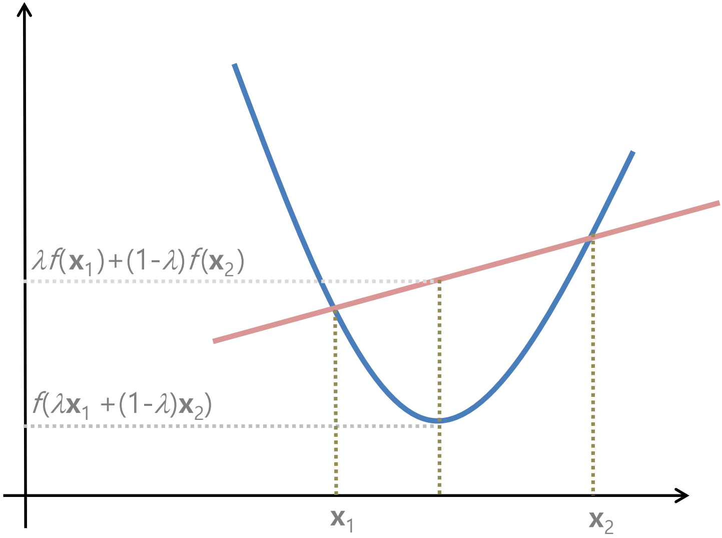
$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}),$$

- strictly convex if

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) < \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y})$$

$\forall \mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$.

Convex


$\lambda f(\mathbf{x}_1)+(1-\lambda)f(\mathbf{x}_2)$
$f(\lambda\mathbf{x}_1+(1-\lambda)\mathbf{x}_2)$
$\mathbf{x}_1$
$\mathbf{x}_2$

Linear least-squares regression

Given a set of data points (pairs of input and output)

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y} \subset \mathbb{R}^n \times \mathbb{R},$$

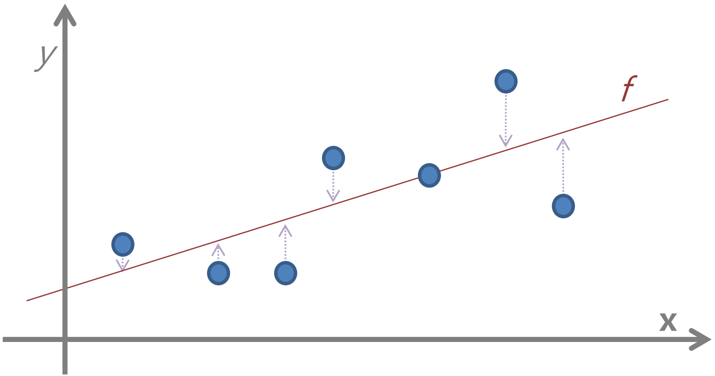
The goal is to find the best-fitting line that minimises the sum of squared errors (SSE):

$$SSE = \sum_i (Prediction - Output)^2.$$

Equivalent expression is:

$$\arg\min_{f(\cdot)} \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2.$$

Linear least-squares regression



Linear least-squares regression

The linear function is defined as:

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 x_0 + w_1 x_1 + \cdots + w_M x_M = \mathbf{w}^\top \mathbf{x}$$

M is the number of input dimensions. The optimum solution can be written as:

$$\mathbf{w}_* = \arg\min_{\mathbf{w} \in \mathbb{R}^n} \sum_{i=1}^{N} (\mathbf{w}^\top \mathbf{x}_i - y_i)^2. \tag{2}$$

With data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and label vector $\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$.

Equation (2) can be rewritten as

$$\mathbf{w}_* = \arg\min_{\mathbf{w} \in \mathbb{R}^n} \|\mathbf{X}^\top \mathbf{w} - \mathbf{y}\|^2.$$

Show that $\|\mathbf{X}^\top \mathbf{w} - \mathbf{y}\|^2 = \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2$.

Linear least-squares regression - closed form solution

Solution using Normal Equations (closed form solution) is:

$$\mathbf{X}\mathbf{X}^\top \mathbf{w}_* = \mathbf{X}\mathbf{y}$$

Derive it.

Then, what is the steepest descent solution?

Linear least-squares regression - steepest descent solution

From our last lecture, the way to update $\mathbf{w}$ is:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \mathbf{p}_t$$

How do we decide

- direction to move $\mathbf{p}_t$,
- step size α_t,
- when to stop (termination condition)?

Linear least-squares regression - steepest descent solution

How do we know that $f(\cdot)$ increases most rapidly along ∇f?

The optimum direction is simply the descent direction, which is $\mathbf{p}_t = -\nabla f_{\mathbf{w}}(\mathbf{x})$, and leads to:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha_t \nabla f_{\mathbf{w}}(\mathbf{x});$$

Linear least-squares regression - steepest descent solution

- How do we decide the step size $\alpha_t > 0$?

  $\rightarrow$ A simple solution (among others): fix it to a constant value.

- When to terminate the optimisation process?

  $\rightarrow$ a simple solution (among others):
  terminate when $\|\nabla f_{\mathbf{w}}(\mathbf{x})\| < \epsilon$, where $\epsilon > 0$ is a small prescribed criteria.

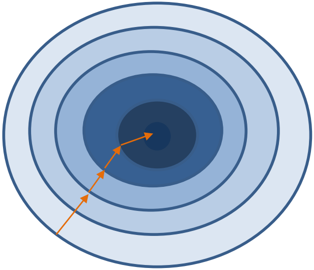
Linear least-squares regression - steepest descent solution

Input: the stopping condition parameter $\epsilon > 0$ and step size $\alpha > 0$;

1. $t = 0$; Make an initial guess $\mathbf{w}_t$;
2. Iterate until $\|\nabla f_{\mathbf{w}}(\mathbf{x})\| < \epsilon$.
   1. $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla f_{\mathbf{w}}(\mathbf{x})$;
   2. $t = t + 1$;

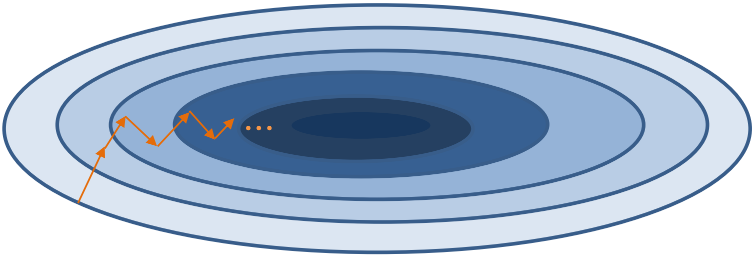
Linear least-squares regression - steepest descent solution

Good for isotropic functions.



Linear least-squares regression - steepest descent solution

Not good for anisotropic functions.



Linear least-squares regression - steepest descent solution

Not good for extreme anisotropic functions.

Linear least-squares regression - steepest descent solution

Solution using Steepest descent is:

Given ϵ and α,

1. $t = 0$; Make an initial guess $\mathbf{w}_0$;
2. Iterate until $\|\nabla f_{\mathbf{w}}(\mathbf{x})\| < \epsilon$.
   1. $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \left(2\mathbf{X}(\mathbf{X}^\top \mathbf{w}_t - \mathbf{y})\right)$;
   2. $t = t + 1$;

How to derive this?

Linear least-squares regression - summary

- Steepest descent:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \left(2\mathbf{X}\mathbf{X}^\top \mathbf{w}_t - 2\mathbf{X}\mathbf{y}\right).$$

  Complexity: $O(M \times N)$ per iteration (why?).
  N: $\#$ data points, M: data dimensionality.

- Closed form solution:

$$\mathbf{X}\mathbf{X}^\top \mathbf{w}_* = \mathbf{X}\mathbf{y}.$$

  Complexity: $O(M^3 + N \times M^2)$.

Disadvantage of vanilla GD

Steepest descent is the vanilla version of gradient descent (GD).

Now we mainly have three variants:

- Batch GD
- Stochastic GD
- Mini-batch GD

Batch GD

First method is to use batch gradient descent.

We use N data points together.

The new updating equation:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha_t \sum_{i=1}^{N} \nabla f_{\mathbf{w}}(\mathbf{x}_i, y_i);$$

Batch GD

How about it, what is the adv. and the disadv.?

- Faster, but can be very slow when the size of data points is huge.
- stable direction, as the direction is from the average of all samples in the batch.

SGD

Second method is to use stochastic gradient descent.

We have the updating equation similar to vanilla GD:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha_t \nabla f_{\mathbf{w}}(\mathbf{x}_{i_s}, y_{i_s});$$

where i_s represents the randomly selected sample.

SGD

How about SGD, what is the adv. and the disadv.?

- Fast.
- Unstable direction.
- Can easily trapped in local optimum.

Mini-batch GD

Third method is a trade-off, to use mini-batch gradient descent.

If we have a mini-batch of B data points, where B is the number of points in the batch and smaller than the total number of points N.

We have the new updating equation:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha_t \sum_{i=1}^{B} \nabla f_{\mathbf{w}}(\mathbf{x}_i, y_i);$$

Mini-batch GD

Why does it work?

Data are correlated! $\nabla f_{\mathbf{w}}(\mathbf{x})$ of a mini-batch is a good approximation of the gradient of the full batch.

- SGD now often refers to a GD with mini-batch.
- The mini-batch size varies, but often set as numbers like 32, 128, 256 etc.

Questions

- What is the difference between the three gradient descent variants?
- Why people usually want to use numbers with power 2 for mini-batch size?
- Why should you do gradient descent when you want to minimise a function?

Reading list

Hin Geoff Hinton's lecture notes.
`https://www.ics.uci.edu/~smyth/courses/cs274/readings/optimization/hinton.pdf`

Goo I. Goodfellow et. al. *Deep Learning Book, Chapter 4 Numerical Computation*,
`http://www.deeplearningbook.org/contents/numerical.html`

Noc J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer (second edition).

Teu Teukolsky, Vetterling, and Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press (any edition).