Machine Learning Course - CS-433

Regression

Sep 19, 2017

minor changes by Martin Jaggi 2016

minor changes by Martin Jaggi 2017

Last updated on: September 18, 2017

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

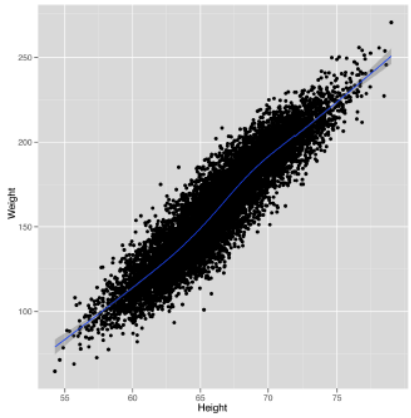
What is regression?

Regression is to relate input variables to the output variable, to either predict outputs for new inputs and/or to understand the effect of the input on the output.

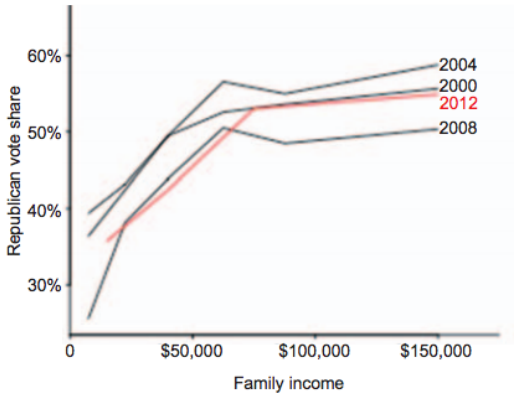
Dataset for regression

In regression, data consists of pairs $(\mathbf{x}_n, y_n)$, where y_n is the n'th output and $\mathbf{x}_n$ is a vector of D inputs. The number of pairs N is the data-size and D is the dimensionality.

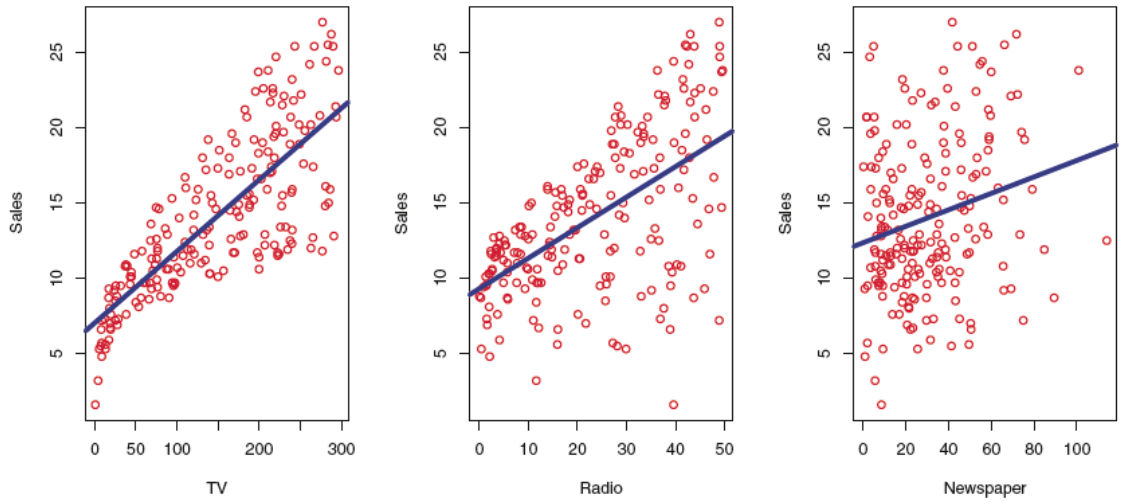
Examples of regression



(a) Height is correlated with weight. Taken from "Machine Learning for Hackers"



(b) Do rich people vote for republicans? Taken from Avi Feller et. al. 2013, Red state/blue state in 2012 elections.



(c) How does advertisement in TV, radio, and newspaper affect sales? Taken from the book "An Introduction to statistical learning"

Two goals of regression

In prediction, we wish to predict the output for a new input vector, e.g. what is the weight of a person who is 170 cm tall?

In interpretation, we wish to understand the effect of inputs on output, e.g. are taller people heavier too?

The regression function

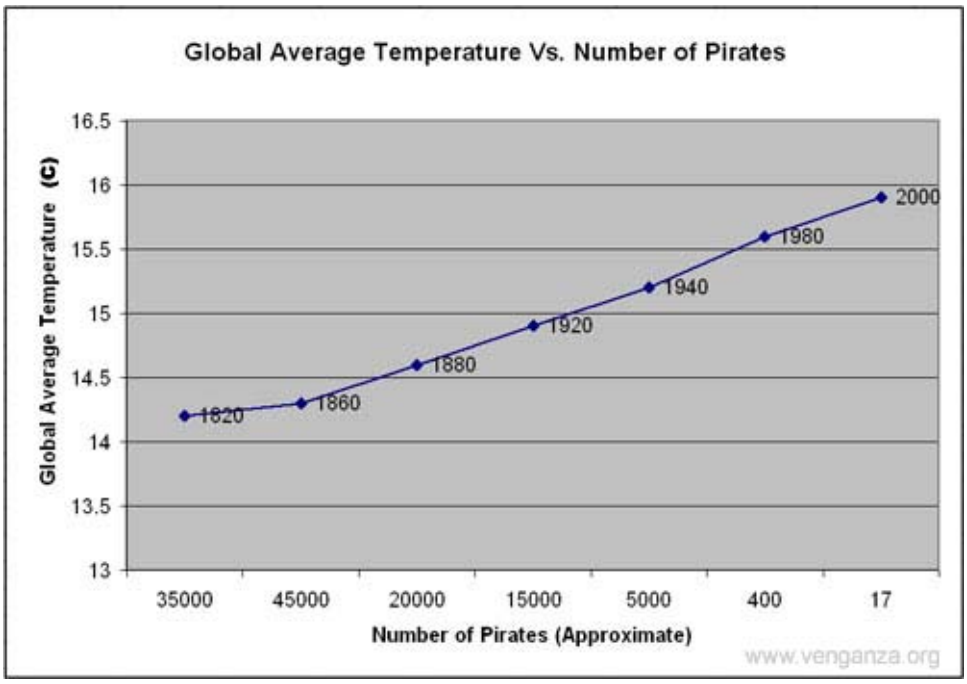
For both the goals, we need to find a function that approximates the output "well enough" given inputs.

$$y_n \approx f(\mathbf{x}_n), \text{ for all } n$$

Additional Notes

Correlation $\neq$ Causation

Regression finds correlation not a causal relationship, so interpret your results with caution.



This image is taken from `http://www.venganza.org/`. You can see many more examples at this page: Spurious correlations page.

Machine Learning Jargon for Regression

Input variables are also known as features, covariates, independent variables, explanatory variables, exogenous variables, predictors, regressors. *Output variables* are also known as target, label, response, outcome, dependent variable, endogenous variables, measured variable, regressands.

Prediction vs Interpretation

Some questions to think about: are these prediction tasks or interpretation task?

1. What is the life-expectancy of a person who has been smoking for 10 years?
2. Does smoking cause cancer?
3. When the number of packs a smoker smokes per day doubles, their life span gets cut in half?
4. A massive scale earthquake will occur in California within next 30 years.
5. More than 300 bird species in north America could reduce their habitat by half or more by 2080.

Machine Learning Course - CS-433

Linear Regression

Sep 21, 2017

minor changes by Martin Jaggi 2016

minor changes by Martin Jaggi 2017

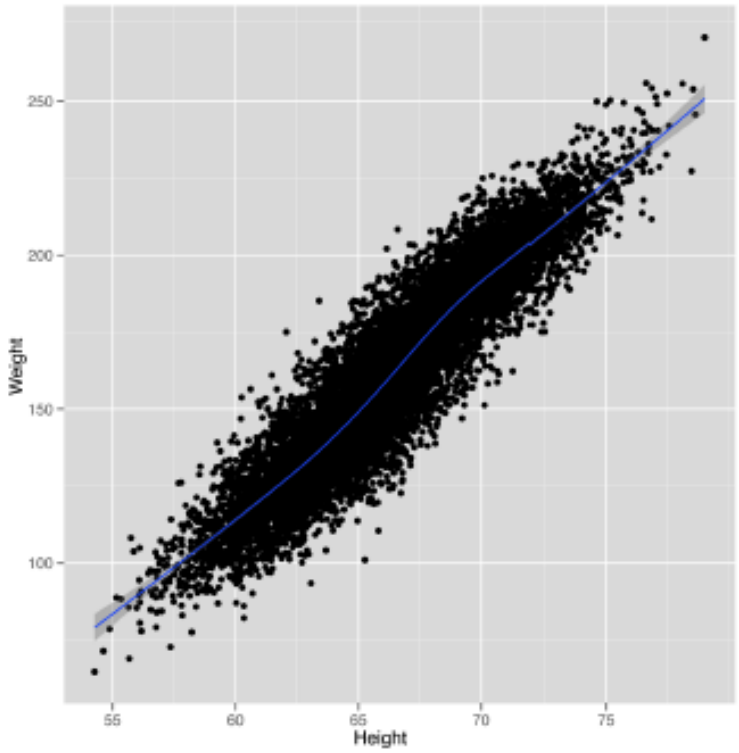
Last updated on: September 21, 2017

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

1 Model: Linear Regression

What is it?

Linear regression is a model that assumes a linear relationship between inputs and the output.



Why learn about linear regression?

Plenty of reasons: simple, easy to understand, most widely used, easily generalized to non-linear models. Most importantly, you can learn almost all fundamental concepts of ML with regression alone.

Simple linear regression

With only one input dimension, we get simple linear regression.

$$y_n \approx f(\mathbf{x}_n) := w_0 + w_1 x_{n1}$$

Here, $\mathbf{w} = (w_0, w_1)$ are the two parameters of the model.

Multiple linear regression

If our data has multiple input dimensions, we obtain multivariate linear regression.

$$\begin{aligned} y_n &\approx f(\mathbf{x}_n) \\ &:= w_0 + w_1 x_{n1} + \ldots + w_D x_{nD} \\ &= w_0 + \mathbf{x}_n^\top \begin{pmatrix} w_1 \\ \vdots \\ w_D \end{pmatrix} \\ &=: \tilde{\mathbf{x}}_n^\top \mathbf{w} \end{aligned}$$

Note that we add a tilde over the input vector to indicate it contains an offset term (a.k.a. bias term).

Learning / Estimation / Fitting

Given data, we would like to find $\mathbf{w} = [w_0, w_1, \ldots, w_D]$. This is called learning or estimating the parameters or fitting the model.
To do so, we need an optimization algorithm, which we will discuss in the chapter after the next.

Additional Notes

Alternative when not using an 'offset' term

Above we have used $D + 1$ model parameters, to fit data of dimension D. An alternative also often used in practice, in particular for high-dimensional data, is to ignore the offset term w_0.

$$\begin{aligned} y_n \approx f(\mathbf{x}_n) &:= w_1 x_{n1} + \ldots + w_D x_{nD} \\ &= \mathbf{x}_n^\top \mathbf{w} \end{aligned}$$

in this case, we have just D parameters to learn, instead of $D + 1$.

As a warning, you should be aware that for some machine learning models, the number of weight parameters (elements of $\mathbf{w}$) can in general be very different from D (being the dimension of the input data). For an ML model where they do not coincide, think for example of a neural network (more details later).

Matrix multiplication

To go any further, one must revise matrix multiplication. Remember that multiplication of $M \times N$ matrix with a $N \times D$ matrix results in a $M \times D$ matrix. Also, two matrices of size $M \times N_1$ and $N_2 \times M$ can only be multiplied when $N_1 = N_2$.

The $D > N$ Problem

Consider the following simple situation: You have $N = 1$ and you want to fit $y_1 \approx w_0 + w_1 x_{11}$, i.e. you want to find $\mathbf{w} = (w_0, w_1)$ given one pair (y_1, x_{11}). Is it possible to find such a line?

This problem is related to something called $D > N$ problem (in statistics typically named $p > n$). It means that the number of parameters exceeds number of data examples. In other words, we have more variables than we have data information. For many models, such as linear regression, this makes the task *under-determined*.

These problems are all solved by using regularization, which we will learn later.

Machine Learning Course - CS-433

Cost Functions

Sep 21, 2017

minor changes by Martin Jaggi 2016

minor changes by Martin Jaggi 2017

Last updated on: September 21, 2017

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Motivation

Consider the following models.

1-parameter model: $y_n \approx w_0$
2-parameter model: $y_n \approx w_0 + w_1 x_{n1}$

How can we estimate (or guess) values of $\mathbf{w}$ given the data $\mathcal{D}$?

What is a cost function?

A cost function (or energy, loss, training objective) is used to learn parameters that explain the data well. The cost function quantifies how well our model does - or in other words how costly our mistakes are.

Two desirable properties of cost functions

When the target y is real-valued, it is often desirable that the cost is symmetric around 0, since both positive and negative errors should be penalized equally.

Also, our cost function should penalize "large" mistakes and "very-large" mistakes similarly.

Statistical vs computational trade-off

If we want better statistical properties, then we have to give-up good computational properties.

Mean Square Error (MSE)

MSE is one of the most popular cost functions.

$$\text{MSE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^{N}\left[y_n - f(\mathbf{x}_n)\right]^2$$

Does this cost function have both mentioned properties?

An exercise for MSE

Compute MSE for 1-param model:

$$\mathcal{L}(w_0) := \frac{1}{N}\sum_{n=1}^{N}\left[y_n - w_0\right]^2$$

	1	2	3	4	5	6	7
$y_1 = 1$ $y_2 = 2$ $y_3 = 3$ $y_4 = 4$							
$\text{MSE}(\mathbf{w}) \cdot N$							
$y_5 = 20$							
$\text{MSE}(\mathbf{w}) \cdot N$							

Some help: $19^2 = 361, 18^2 = 324, 17^2 = 289, 16^2 = 256, 15^2 = 225, 14^2 = 196, 13^2 = 169$.

Outliers

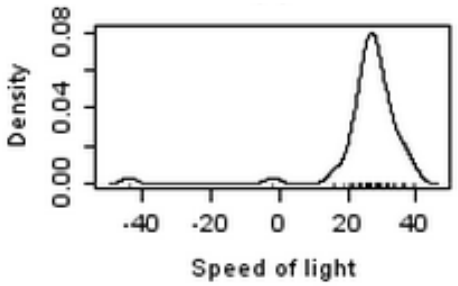
Outliers are data examples that are far away from most of the other examples. Unfortunately, they occur more often in reality than you would want them to!

MSE is not a good cost function when outliers are present.

Here is a real example on speed of light measurements (Gelman's book on Bayesian data analysis)

28	26	33	24	34	-44	27	16	40	-2
29	22	24	21	25	30	23	29	31	19
24	20	36	32	36	28	25	21	28	29
37	25	28	26	30	32	36	26	30	22
36	23	27	27	28	27	31	27	26	33
26	32	32	24	39	28	24	25	32	25
29	27	28	29	16	23				

(a) Original speed of light data done by Simon Newcomb.



(b) Histogram showing outliers.

Handling outliers well is a desired *statistical* property.

Mean Absolute Error (MAE)

$$\text{MAE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^{N}\left|y_n - f(\mathbf{x}_n)\right|$$

Repeat the exercise with MAE.

	1	2	3	4	5	6	7
$y_1 = 1$ $y_2 = 2$ $y_3 = 3$ $y_4 = 4$							
$\text{MAE}(\mathbf{w}) \cdot N$							
$y_5 = 20$							
$\text{MAE}(\mathbf{w}) \cdot N$							

Can you draw MSE and MAE for the above example?

Convexity

Roughly, a function is convex iff a line joining two points never intersects with the function anywhere else.

A function $f(\mathbf{u})$ with $\mathbf{u} \in \mathcal{X}$ is convex, if for any $\mathbf{u}, \mathbf{v} \in \mathcal{X}$ and for any $0 \leq \lambda \leq 1$, we have:

$$f(\lambda \mathbf{u} + (1-\lambda)\mathbf{v}) \leq \lambda f(\mathbf{u}) + (1-\lambda) f(\mathbf{v})$$

A function is strictly convex if the inequality is strict.

Importance of convexity

A strictly convex function has a unique global minimum $\mathbf{w}^\star$. For convex functions, every local minimum is a global minimum.

Sums of convex functions are also convex. Therefore, MSE is convex.

Convexity is a desired *computational* property.

Can you prove that the MAE is convex? (as a function of the parameters $\mathbf{w} \in \mathbb{R}^D$, for linear regression $f(\mathbf{x}) := \mathbf{x}^\top \mathbf{w}$)

Computational VS statistical trade-off

So which loss function is the best?

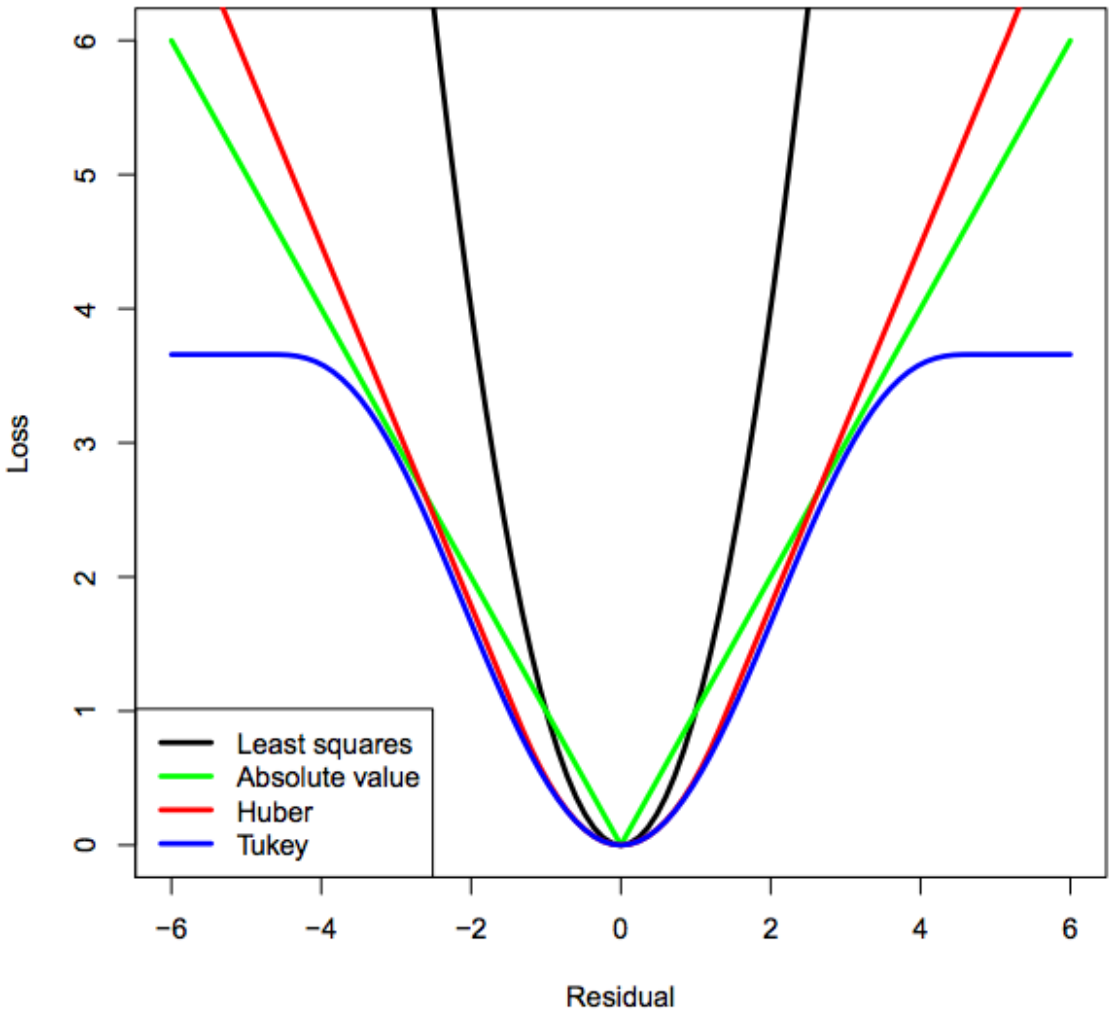


Figure taken from Patrick Breheny's slides.

If we want better statistical properties, then we have to give-up good computational properties.

Additional Reading

Other cost functions

Huber loss

$$Huber := \begin{cases} \frac{1}{2}e^2 & , \text{if } |e| \leq \delta \\ \delta|e| - \frac{1}{2}\delta^2 & , \text{if } |e| > \delta \end{cases} \tag{1}$$

Huber loss is convex, differentiable, and also robust to outliers. However, setting δ is not an easy task.

Tukey's bisquare loss (defined in terms of the gradient)

$$\frac{\partial \mathcal{L}}{\partial e} := \begin{cases} e\{1 - e^2/\delta^2\}^2 & , \text{if } |e| \leq \delta \\ 0 & , \text{if } |e| > \delta \end{cases} \tag{2}$$

Tukey's loss is non-convex, but robust to outliers.

Additional reading on outliers

- Wikipedia page on "Robust statistics".
- Repeat the exercise with MAE.
- Sec 2.4 of Kevin Murphy's book for an example of robust modeling

Nasty cost functions: Visualization

See Andrej Karpathy Tumblr post for many cost functions gone "wrong" for neural networks. http://lossfunctions.tumblr.com/.

Machine Learning Course - CS-433

Optimization

Sep 26+28, 2017

minor changes by Martin Jaggi 2017

Last updated on: September 28, 2017

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Learning / Estimation / Fitting

Given a cost function $\mathcal{L}(\mathbf{w})$, we wish to find $\mathbf{w}^\star$ which minimizes the cost:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad \text{subject to } \mathbf{w} \in \mathbb{R}^D$$

This means the *learning* problem is formulated as an optimization problem.
We will use an optimization algorithm to solve the problem (to find a good $\mathbf{w}$).

Grid Search

Grid search is one of the simplest optimization algorithms. We compute the cost over all values $\mathbf{w}$ in a grid, and pick the best among those.

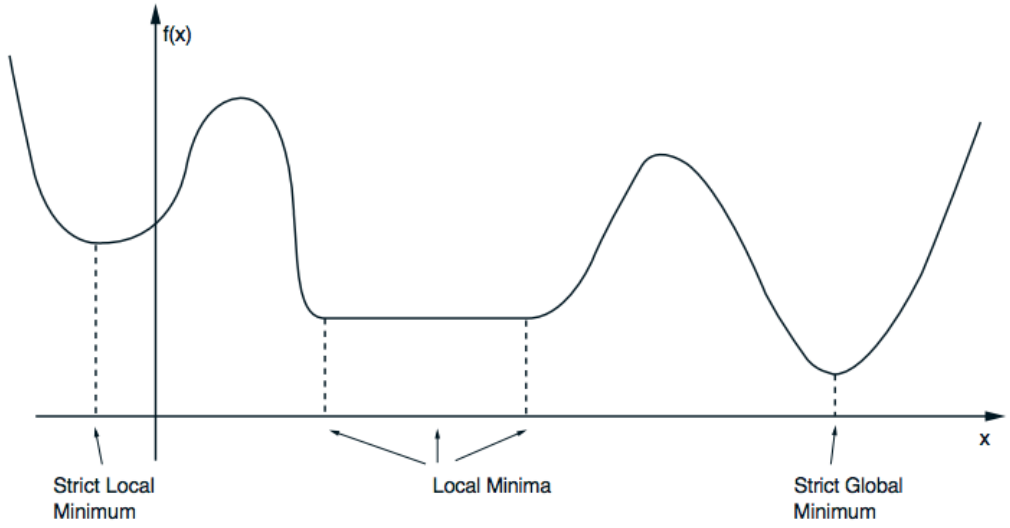
This is brute-force, but extremely simple and works for any kind of cost function when we have very few parameters and the cost is easy to compute.

For a large number of parameters D, however, grid search has too many "for-loops", resulting in an exponential computational complexity:

If we decide to use 10 possible values for each dimension of $\mathbf{w}$, then we have to check 10^D points. This is clearly impossible for most practical machine learning models, which can often have $D \approx$ millions of parameters. Choosing a good range of values for each dimension is another problem.

Other issues: No guarantee can be given that we end up close to an optimum.

Optimization Landscapes



The above figure is taken from Bertsekas, Nonlinear programming.

A vector $\mathbf{w}^\star$ is a local minimum of $\mathcal{L}$ if it is no worse than its neighbors; i.e. there exists an $\epsilon > 0$ such that,

$$\mathcal{L}(\mathbf{w}^\star) \leq \mathcal{L}(\mathbf{w}), \quad \forall \mathbf{w} \text{ with } \|\mathbf{w} - \mathbf{w}^\star\| < \epsilon$$

A vector $\mathbf{w}^\star$ is a global minimum of $\mathcal{L}$ if it is no worse than all others,

$$\mathcal{L}(\mathbf{w}^\star) \leq \mathcal{L}(\mathbf{w}), \quad \forall \mathbf{w} \in \mathbb{R}^D$$

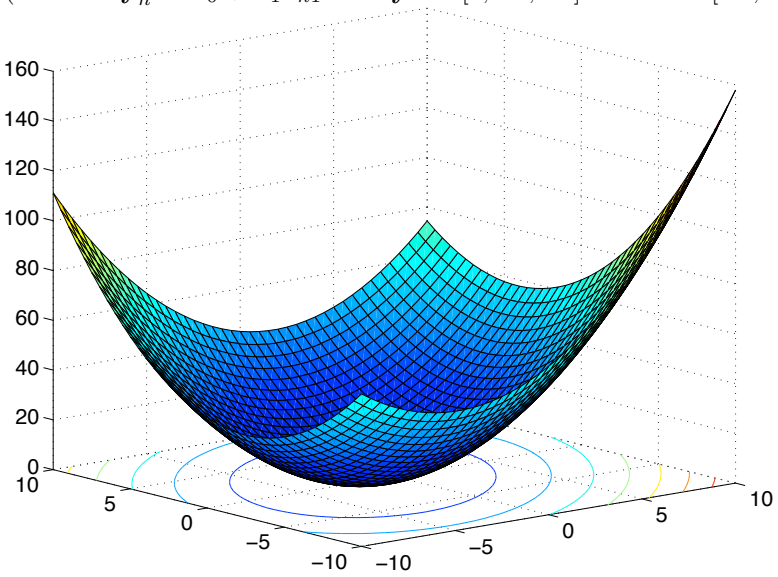
A local or global minimum is said to be strict if the corresponding inequality is strict for $\mathbf{w} \neq \mathbf{w}^\star$.

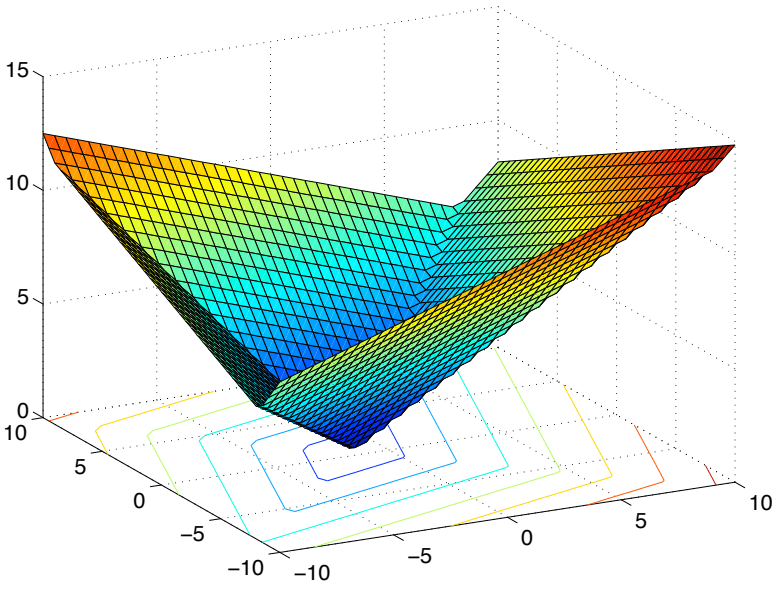
Smooth Optimization

Follow the Gradient

A gradient (at a point) is the slope of the tangent to the function (at that point). It points to the direction of largest increase of the function.

For a 2-parameter model, MSE($\mathbf{w}$) and MAE($\mathbf{w}$) are shown below.
(We used $\mathbf{y}_n \approx w_0 + w_1 x_{n1}$ with $\mathbf{y}^\top = [2, -1, 1.5]$ and $\mathbf{x}^\top = [-1, 1, -1]$).





Definition of the gradient:

$$\nabla \mathcal{L}(\mathbf{w}) := \left[\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_D} \right]^\top$$

This is a vector, $\nabla \mathcal{L}(\mathbf{w}) \in \mathbb{R}^D$.

Gradient Descent (Batch)

To minimize the function, we iteratively take a step in the (opposite) direction of the gradient

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma \nabla \mathcal{L}(\mathbf{w}^{(t)})$$

where $\gamma > 0$ is the step-size (or learning rate). Then repeat with the next t.

Example: Gradient descent for 1-parameter model to minimize MSE:

$$w_0^{(t+1)} := (1 - \gamma) w_0^{(t)} + \gamma \bar{y}$$

where $\bar{y} := \sum_n y_n / N$. When is this sequence guaranteed to converge?

Gradient Descent for Linear MSE

For linear regression

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1D} \\ x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix}$$

We define the error vector $\mathbf{e}$:

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$$

and MSE as follows:

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &:= \frac{1}{2N} \sum_{n=1}^{N} \left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2 \\ &= \tfrac{1}{2N} \mathbf{e}^\top \mathbf{e} \end{aligned}$$

then the gradient is given by

$$\nabla \mathcal{L}(\mathbf{w}) = -\tfrac{1}{N} \mathbf{X}^\top \mathbf{e}$$

Computational cost. What is the complexity (# operations) of computing the gradient?

a) starting from $\mathbf{w}$ and

b) given $\mathbf{e}$ and $\mathbf{w}$?

Variant with offset. Recall: Alternative trick when also incorporating an offset term for the regression:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \widetilde{\mathbf{X}} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix}$$

Stochastic Gradient Descent

Sum Objectives. In machine learning, most cost functions are formulated as a sum over the training examples, that is

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n(\mathbf{w}) \ ,$$

where $\mathcal{L}_n$ is the cost contributed by the n-th training example.

Q: What are the $\mathcal{L}_n$ for linear MSE?

The SGD Algorithm. The stochastic gradient descent (SGD) algorithm is given by the following update rule, at step t:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma \nabla \mathcal{L}_n(\mathbf{w}^{(t)}) \ .$$

Theoretical Motivation. *Idea:* Cheap but unbiased estimate of the gradient!
In expectation over the random choice of n, we have

$$\mathbb{E}\left[\nabla \mathcal{L}_n(\mathbf{w})\right] = \nabla \mathcal{L}(\mathbf{w})$$

which is the true gradient direction. (check!)

Mini-batch SGD. There is an intermediate version, using the update direction being

$$\mathbf{g} := \frac{1}{|B|} \sum_{n \in B} \nabla \mathcal{L}_n(\mathbf{w}^{(t)})$$

again with

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\, \mathbf{g} \ .$$

In the above gradient computation, we have randomly chosen a subset $B \subseteq [N]$ of the training examples. For each of these selected examples n, we compute the respective gradient $\nabla \mathcal{L}_n$, at the same current point $\mathbf{w}^{(t)}$.

The computation of $\mathbf{g}$ can be parallelized easily. This is how current deep-learning applications utilize GPUs (by running over $|B|$ threads in parallel).
Note that in the extreme case $B := [N]$, we obtain (batch) gradient descent, i.e. $\mathbf{g} = \nabla \mathcal{L}$.

SGD for Linear MSE

See Exercise Sheet 2.

Computational cost. For linear MSE, what is the complexity (# operations) of computing the stochastic gradient?
(using only $|B| = 1$ data examples)

a) starting from $\mathbf{w}$?

Non-Smooth Optimization

An alternative characterization of *convexity*, for differentiable functions is given by

$$\mathcal{L}(\mathbf{u}) \geq \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})^\top(\mathbf{u}-\mathbf{w}) \quad \forall \mathbf{u}, \mathbf{w}$$

meaning that the function must always lie above its linearization.

Subgradients

A vector $\mathbf{g} \in \mathbb{R}^D$ such that

$$\mathcal{L}(\mathbf{u}) \geq \mathcal{L}(\mathbf{w}) + \mathbf{g}^\top(\mathbf{u} - \mathbf{w}) \quad \forall \mathbf{u}$$

is called a subgradient to the function $\mathcal{L}$ at $\mathbf{w}$.

This definition makes sense for objectives $\mathcal{L}$ which are not necessarily differentiable (and not even necessarily convex).

If $\mathcal{L}$ is differentiable at $\mathbf{w}$, then the only subgradient at $\mathbf{w}$ is $\mathbf{g} = \nabla\mathcal{L}(\mathbf{w})$.

Subgradient Descent

Identical to the gradient descent algorithm, but using a subgradient instead of gradient. Update rule

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\, \mathbf{g}$$

for $\mathbf{g}$ being a subgradient to $\mathcal{L}$ at the current iterate $\mathbf{w}^{(t)}$.

Example: Optimizing Linear MAE

1. Compute a subgradient of the absolute value function

   $h : \mathbb{R} \to \mathbb{R}$, $h(e) := |e|$.

2. Recall the definition of the mean absolute error:

   $$\mathcal{L}(\mathbf{w}) = \mathrm{MAE}(\mathbf{w}) := \frac{1}{N} \sum_{n=1}^{N} \left| y_n - f(\mathbf{x}_n) \right|$$

   For linear regression, its (sub)gradient is easy to compute using the chain rule. Compute it!

See Exercise Sheet 2.

Stochastic Subgradient Descent

Stochastic SubGradient Descent (still abbreviated SGD commonly).

Same, $\mathbf{g}$ being a subgradient to the randomly selected $\mathcal{L}_n$ at the current iterate $\mathbf{w}^{(t)}$.

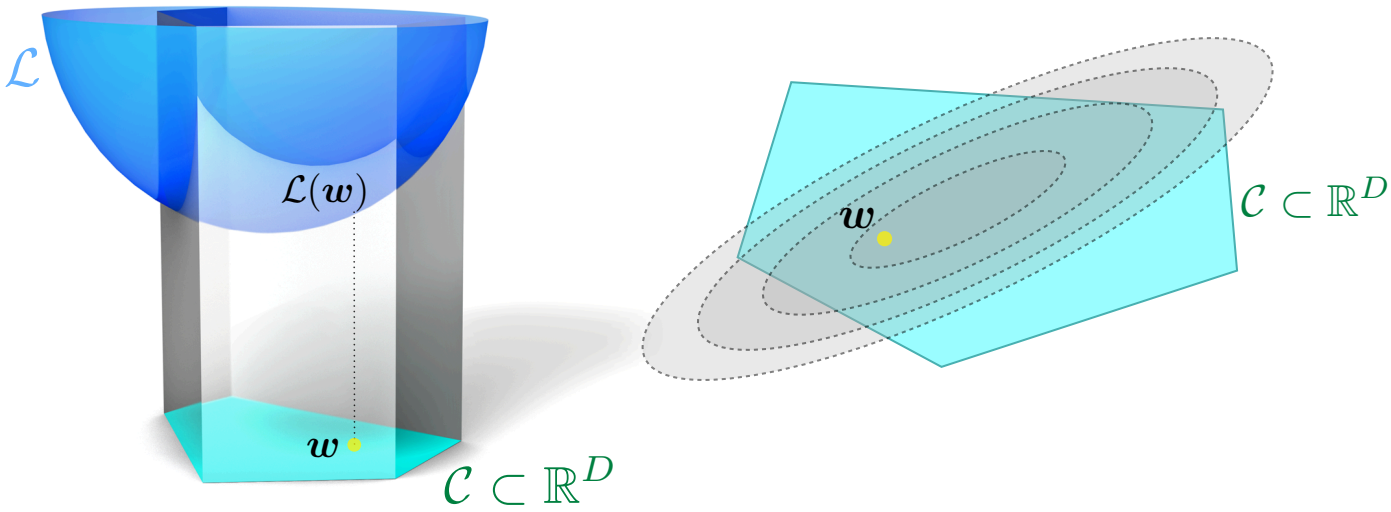
Exercise: Compute the SGD update for linear MAE.

Constrained Optimization

Sometimes, optimization problems come posed with additional constraints:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}), \quad \text{subject to } \mathbf{w} \in \mathcal{C}.$$

The set $\mathcal{C} \subset \mathbb{R}^D$ is called the constraint set.



Solving Constrained Optimization Problems

A) Projected Gradient Descent

B) Transform it into an *unconstrained* problem

Convex Sets

A set $\mathcal{C}$ is convex *iff*
the line segment between any two points of $\mathcal{C}$ lies in $\mathcal{C}$, i.e., if for any $\mathbf{u}, \mathbf{v} \in \mathcal{C}$ and any θ with $0 \leq \theta \leq 1$, we have

$$\theta\mathbf{u} + (1 - \theta)\mathbf{v} \in \mathcal{C}.$$

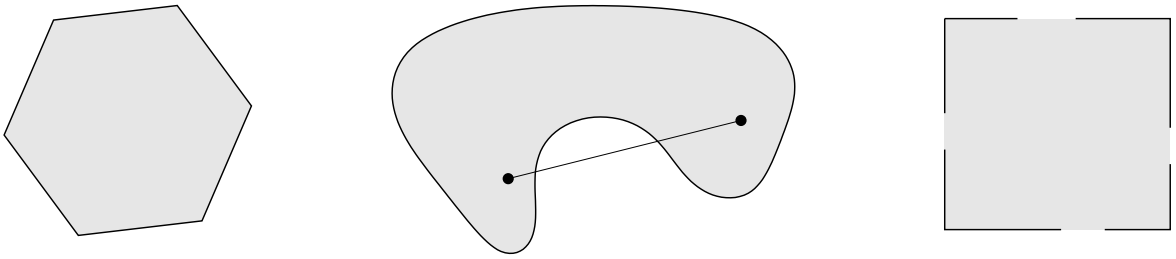


*Figure 2.2 from S. Boyd, L. Vandenberghe

Properties of Convex Sets

- Intersections of convex sets are convex
- Projections onto convex sets are *unique*.
  (and often efficient to compute)
  Formal definition:
  $P_{\mathcal{C}}(\mathbf{w}') := \arg\min_{\mathbf{v} \in \mathcal{C}} \|\mathbf{v} - \mathbf{w}'\|$.

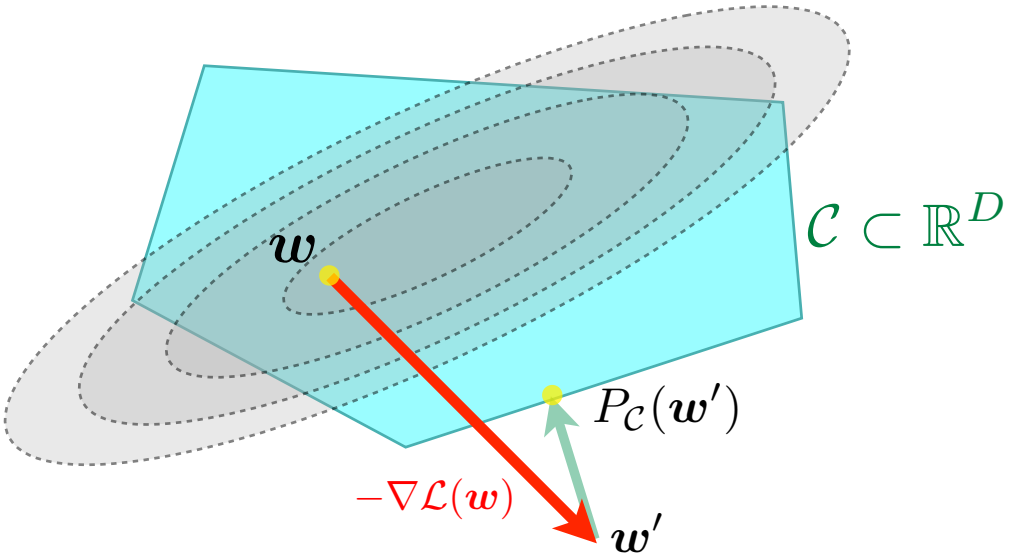
Projected Gradient Descent

Idea: add a projection onto $\mathcal{C}$ after every step:

$$P_{\mathcal{C}}(\mathbf{w}') := \arg\min_{\mathbf{v}\in\mathcal{C}} \|\mathbf{v} - \mathbf{w}'\| \ .$$

Update rule:

$$\mathbf{w}^{(t+1)} := P_{\mathcal{C}}\left[\mathbf{w}^{(t)} - \gamma\, \nabla\mathcal{L}(\mathbf{w}^{(t)})\right] \ .$$



Projected SGD. Same SGD step, followed by the projection step, as above. Same convergence properties.

Computational cost of projection? Crucial!

Turning Constrained into Unconstrained Problems

(Alternatives to projected gradient methods)

Use penalty functions instead of directly solving $\min_{\mathbf{w}\in\mathcal{C}} \mathcal{L}(\mathbf{w})$.

- "*brick wall*" (indicator function)

$$I_{\mathcal{C}}(\mathbf{w}) := \begin{cases} 0 & \mathbf{w} \in \mathcal{C} \\ \infty & \mathbf{w} \notin \mathcal{C} \end{cases}$$

$$\Rightarrow \min_{\mathbf{w}\in\mathbb{R}^D} \mathcal{L}(\mathbf{w}) + I_{\mathcal{C}}(\mathbf{w})$$

(disadvantage: non-continuous objective)

- Penalize error. *Example:* $\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^D \mid A\mathbf{w} = \mathbf{b}\}$

$$\Rightarrow \min_{\mathbf{w}\in\mathbb{R}^D} \mathcal{L}(\mathbf{w}) + \lambda \|A\mathbf{w} - \mathbf{b}\|^2$$

- Linearized Penalty Functions (see Lagrange Multipliers)

Implementation Issues

For Gradient Methods:

Stopping criteria: When $\nabla \mathcal{L}(\mathbf{w})$ is (close to) zero, we are (often) close to the optimum.

Optimality: If the second-order derivative is positive (positive semi-definite to be precise), then it is a (possibly local) minimum. If the function is also convex, then this condition implies that we are at a global optimum. See the supplementary section on Optimality Conditions.

Step-size selection: If γ is too big, the method might diverge. If it is too small, convergence is slow. Convergence to a local minimum is guaranteed only when $\gamma < \gamma_{min}$ where γ_{min} is a fixed constant that depends on the problem.

Line-search methods: For some objectives $\mathcal{L}$, we can set step-size automatically using a line-search method. More details on "backtracking" methods can be found in Chapter 1 of Bertsekas' book on "nonlinear programming".

Feature normalization and preconditioning: Gradient descent is very sensitive to ill-conditioning. Therefore, it is typically advised to normalize your input features. In other words, we pre-condition the optimization problem. Without this, step-size selection is more difficult since different "directions" might converge at different speed.

Non-Convex Optimization

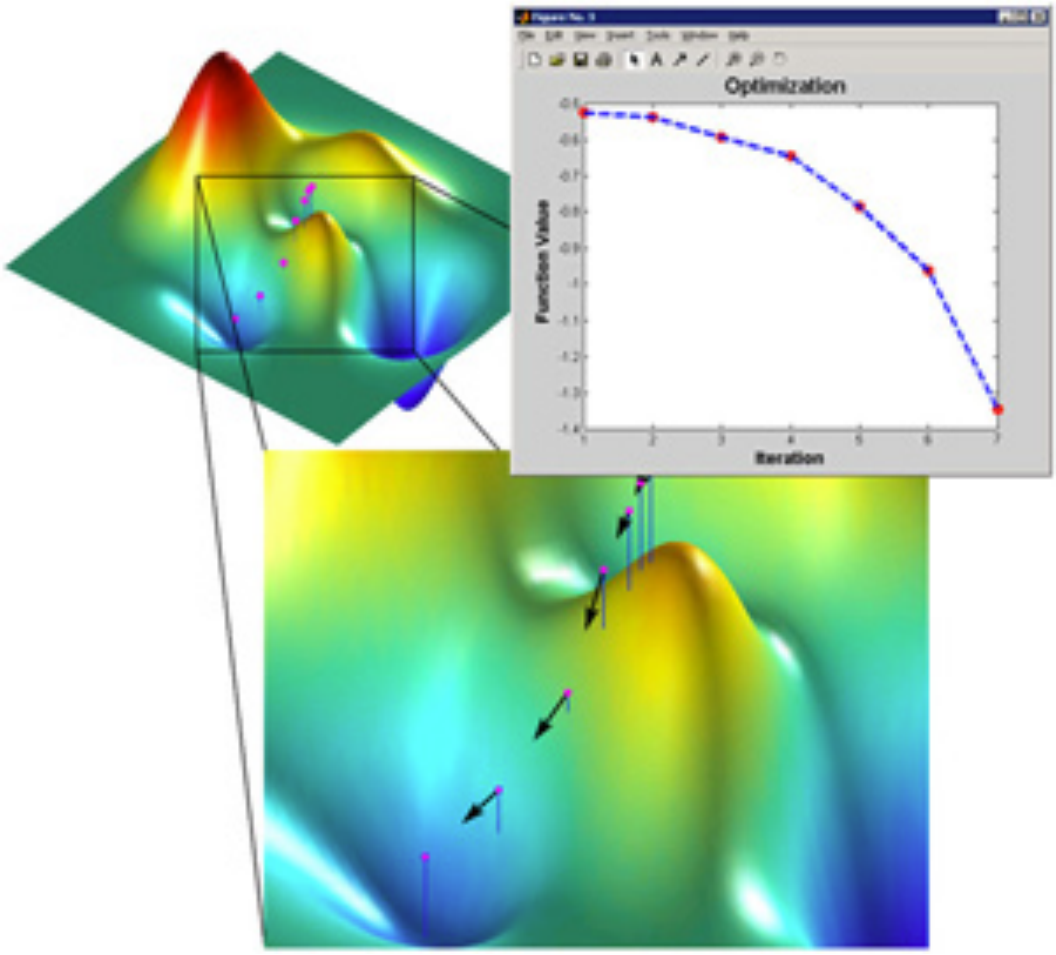


*image from mathworks.com

Real-world problems are not convex!

All we have learnt on algorithm design and performance of convex algorithms still helps us in the non-convex world.

Additional Notes

Grid Search and Hyper-Parameter Optimization

Read more about grid search and other methods for "hyperparameter" setting:
en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search.

Computational Complexity

The computation cost is expressed using the big-$\mathcal{O}$ notation. Here is a definition taken from Wikipedia. Let f and g be two functions defined on some subset of the real numbers. We write $f(x) = \mathcal{O}(g(x))$ as $x \to \infty$, if and only if there exists a positive real number c and a real number x_0 such that $|f(x)| \leq c|g(x)|, \quad \forall x > x_0$.

Please read and learn more from this page in Wikipedia:
en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#Matrix_algebra .

- What is the computational complexity of matrix multiplication?
- What is the computational complexity of matrix-vector multiplication?

Optimality Conditions

For a *convex* optimization problem, the first-order *necessary* condition says that at *an* optimum the gradient is equal to zero.

$$\nabla \mathcal{L}(\mathbf{w}^\star) = \mathbf{0} \tag{1}$$

The second-order *sufficient* condition ensures that the optimum is a minimum (not a maximum or saddle-point) using the Hessian matrix,

which is the matrix of second derivatives:

$$\mathbf{H}(\mathbf{w}^\star) := \frac{\partial^2 \mathcal{L}(\mathbf{w}^\star)}{\partial \mathbf{w} \partial \mathbf{w}^\top} \quad \text{is positive semi-definite.} \tag{2}$$

The Hessian is also related to the convexity of a function: a twice-differentiable function is convex if and only if the Hessian is positive semi-definite at all points.

SGD Theory

As we have seen above, when N is large, choosing a random training example $(\mathbf{x}_n, y_n)$ and taking an SGD step is advantageous:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \gamma^{(t)} \nabla \mathcal{L}_n(\mathbf{w}^{(t)})$$

For convergence, $\gamma^{(t)} \to 0$ "appropriately". One such condition called the Robbins-Monroe condition suggests to take $\gamma^{(t)}$ such that:

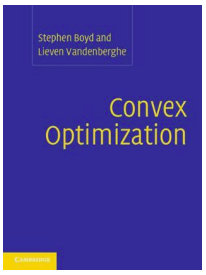
$$\sum_{t=1}^{\infty} \gamma^{(t)} = \infty, \qquad \sum_{t=1}^{\infty} (\gamma^{(t)})^2 < \infty \tag{3}$$

One way to obtain such sequences is $\gamma^{(t)} := 1/(t+1)^r$ where $r \in (0.5, 1)$.

More Optimization Theory

If you want, you can gain a deeper understanding of several optimization methods relevant for machine learning from this survey:

Convex Optimization: Algorithms and Complexity - by Sébastien Bubeck



(> 35 000 citations)

And also from the book of Boyd & Vandenberghe (both are free online PDFs)

Exercises

1. Chain-rule

   If it has been a while, familiarize yourself with it again.

2. Revise computational complexity (also see the Wikipedia link in Page 6 of lecture notes).

3. Derive the computational complexity of grid-search, gradient descent and stochastic gradient descent for linear MSE (# steps and cost per step).

4. Derive the gradients for the linear MSE and MAE cost functions.

5. Implement gradient descent and gain experience in setting the step-size.

6. Implement SGD and gain experience in setting the step-size.