Norm of a vector

- Why Norm in Data Science?

  To summarize or compress the information about a data vector to a single number .

  Example:

❖ To find **magnitude** of a data point(row vector)

❖ To find the **loss** in machine learning model

❖ To compute the **error** in a predictive model

Norm:

- It is the magnitude/length/size of a vector (row or column vector of a data matrix.)
- Definition:

A norm on a vector space V denoted by ||·|| is a function from V to R

which assigns each vector x its length $||x||$,such that the following properties hold :

1. $||\lambda x|| = |\lambda|\,||x|| \quad \forall\ \lambda \epsilon R$
2. $||x + y|| \leq ||x|| + ||y||$.......**(Triangle Inequality)**
3. $||x|| \geq 0\ \ \&\ \ ||x|| = 0\,, \Leftrightarrow x = 0$ ....**(Positive definiteness)**

Distance:

It is defined as the norm of difference between two vectors.

If x & y are any two vectors then

$$d(x,y) = ||x - y||$$

Types of Norms

- *Consider a vector* $x \in R^n$

L_P **Norm:**

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} = \sqrt[p]{|x_1|^p + |x_2|^p + \cdots \ldots . |x_n|^p}$$

where $p \in R$, $p \geq 1$

❑ Case p $= 1$: L_1 **Norm**

$\|x\|_1 = \sum_{i=0}^{n} |x_i| = |x_1| + |x_2| + \cdots \ldots \ldots \ldots |x_n|$

Example: Consider $x = (1, -1, 2) \in R^3$ then

$\|x\|_1 = \sum_{i=1}^{3} |x_i| = |1| + |-1| + |2| = 1 + 1 + 2 = 4$

Uses of L_1 Norm

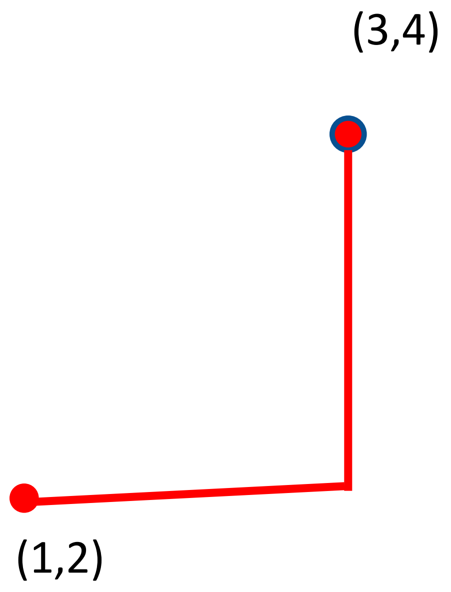
- Used to check if the vector is a zero vector or a very small vector which can be mistaken as zero.
- Used to calculate **Manhattan distance** or **city block distance**:

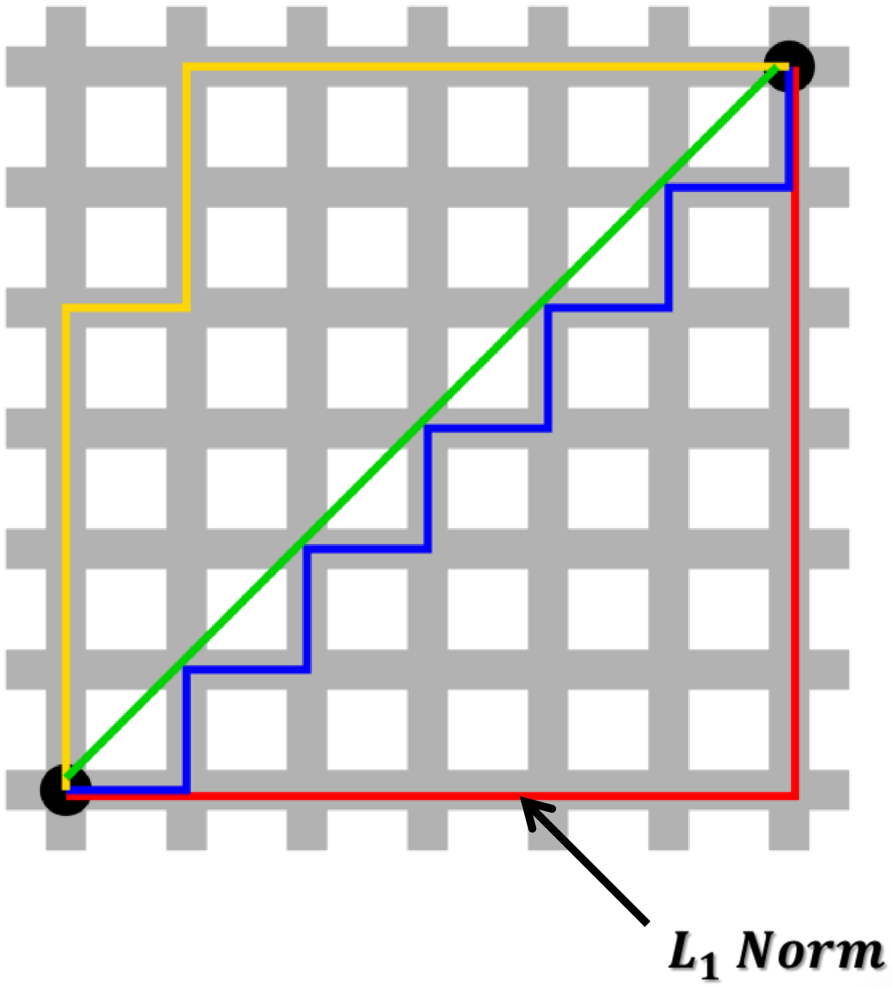
If $x = (x_1, x_2, \dots \dots x_n)$ & $y = (y_1, y_2, \dots \dots y_n)$ are two data points then Manhattan distance is given by

$$d(x, y) = \|x - y\|_1 = \sum_{i=0}^{n} |x_i - y_i|$$

Example: IF $x = (1, 2)$& $y = (3,4)$ are two data points then $d(x, y) = |1 - 3| + |2 - 4| = 2 + 2 = 4$ is the Manhattan distance between them.

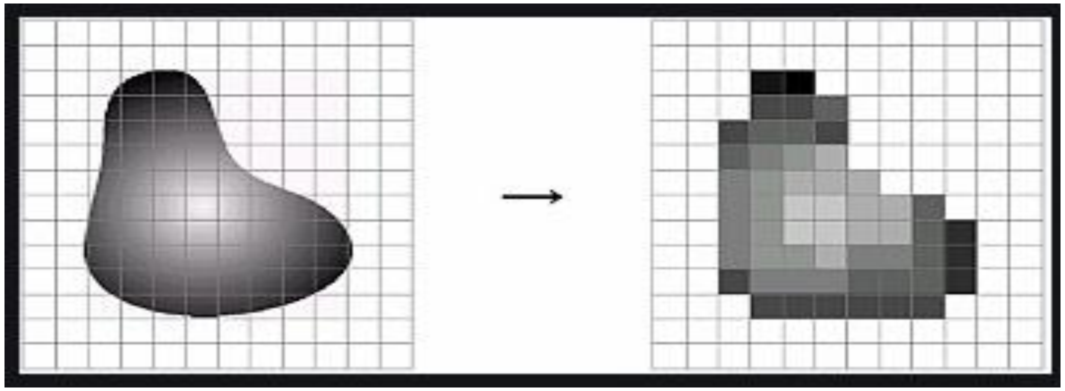
Geometric view:

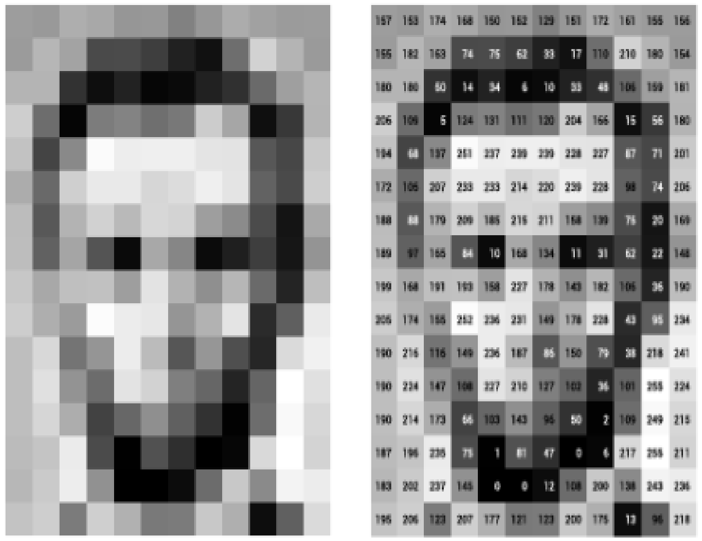


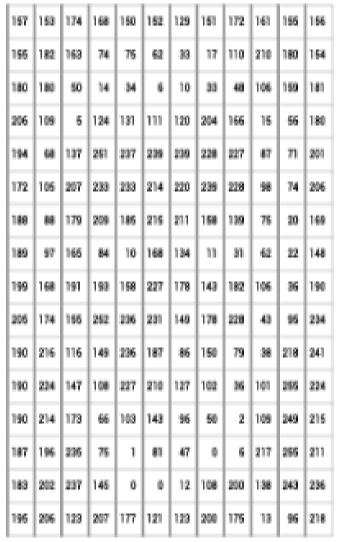


Source: Wikipedia

Manhatten in Pixel Processing







157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

❑ Case $p = 2$: L_2 Norm

- $\|x\|_2 = \sqrt{\sum_{i=1}^{n}(x_i)^2} = \sqrt{x^T x} = \sqrt{x_1^2 + x_2^2 + \cdots \ldots\ldots + x_n^2}$

Also called as **Euclidean norm.**

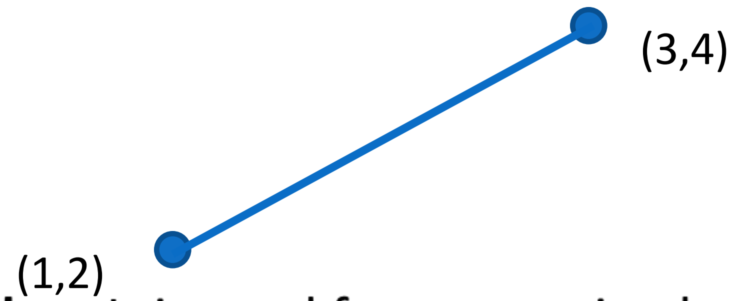
Example: If $x = (1, -1, 2) \in R^3$ then

$$\|x\|_2 = \sqrt{\sum_{i=1}^{3}(x_i)^2} = \sqrt{(1)^2 + (-1)^2 + (2)^2} = \sqrt{1 + 1 + 4}$$
$$= \sqrt{6}$$

Uses of L_2 Norm

- To find straight line distance between two points.

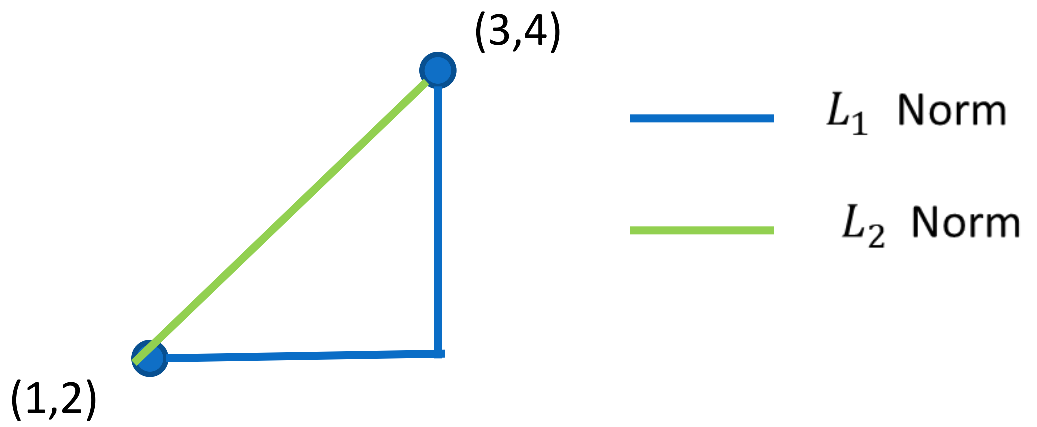
Example: IF $x = (1,2)$& $y = (3,4)$ are two data points then $d(x,y) = \sqrt{(1-3)^2+(2-4)^2} = \sqrt{4+4} = \sqrt{8}$ is the Euclidean distance distance between them.



- **RMS Value:** It is used for comparing lengths of two vectors & it is defined as

$$\boldsymbol{RMS(x)} = \frac{\|\boldsymbol{x}\|_2}{\sqrt{\boldsymbol{n}}} = \frac{\sqrt{x_1^2+x_2^2+\cdots.....+x_n^2}}{\sqrt{\boldsymbol{n}}}$$

L_1 *vs* L_2 Norm



- As can be seen from the figure L_2 Norm ≤ L_2 Norm
- L_1 norm is more robust(to outliers) than L_2 norm (as they are squared in L_2 norm)
- L_2 norm produces stable solutions than L_1 norm.(since L_1 contains absolute values & is not differentiable)

Applications of norms in data science:

- Used in classification algorithms such as KNN to find nearest points .
- In K means clustering each data point is assigned to its nearest centroid using Euclidean distance.$(L_2\ norm)$
- Used in regularization to keep the coefficients of the model small
- Used in optimization to calculate cost/loss function.

Optimization

- In real world problems, optimization is a process of making something as good or effective as possible. This process involves choosing inputs that give the best or desired output.
- Example :
  - Choosing optimum location of warehouse to reduce shipment time
  - Designing a bridge that can carry maximum possible/desired load
  - Selecting stocks that will create maximum returns

Optimization

- In mathematics, optimization is a process of optimizing i.e. maximizing or minimizing a function of one or more variables on a given domain. Here the function to be optimized is called an *objective function* which is a quantitative measure of performance.

  For example: to maximize profits, minimize time, minimize costs, maximize sales are some optimization problems in day to day life

  Cost function or loss function are some frequently used terms for objective function used in machine learning. The input parameters to the cost function are called *decision/design variables*.

Mathematical formulation of optimization problem:

-
- $Maximixe\ /Minimize\ f(x)$ $\longrightarrow$ Objective function

  $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ $\longrightarrow$ Decision variable

Subject to $h(x) = 0$ $\longrightarrow$ Equality constraints

$g(x) \leq 0$ $\longrightarrow$ Inequality constraints

Where ‘x’ can be a single variable or a vector (of more than one variables)

Types of Optimization:

- optimization problems can be classified into various categories based on nature of objective function, decision variables and constraints , physical structure of the problem etc.
- Few important classifications are:
  - ❑ **Constrained Vs. unconstrained optimization:** If there are one or more constraints on decision variables then it is constrained optimization problem and in case there are no constraints then it is unconstrained optimization.
  - ❑ **Linear Vs. Non-linear Optimization :** If the objective function and all the constraints are linear functions of decision variables then the optimization problem is called linear programming problem(**LPP**) otherwise it is called non linear programming (**NLP**) problem.

Types of Optimization:

- **Discrete Vs. Continuous optimization:**

  If some or all the decision variables are restricted to take only integer values then the optimization problem is called integer programming problem or discrete optimization Otherwise it is called continuous optimization where the decision variables take continuous set of values.

- **Deterministic Vs. Stochastic Optimization:**

  In deterministic optimization, it is assumed that the data for the given problem are known accurately. However, for many actual problems, the data cannot be known accurately for a variety of reasons. The first reason is due to simple measurement error. The second and more fundamental reason is that some data represent information about the future (e. g., product demand or price for a future time period) and simply cannot be known with certainty. In stochastic optimization, the uncertainty is incorporated into the model.

Unconstrained Optimization

- Formulation: Maximize/ Minimize $f(x)$ $x \in R^n$
- It is possible to reduce all maximization problems into minimization since

$$\text{Maximize } f(x) \quad \Rightarrow \quad \text{Minimize } -f(x)$$

so that henceforth we can talk of only minimization.

- Solution: If x^* is a point at which the function takes optimum(in this case minimum) value then x^* is the solution of the given problem which is denoted as

$$x^* = \arg\min f(x)$$

which simply means that x^* is a value at which $f(x)$ takes minimum argument(value)

Stationary point

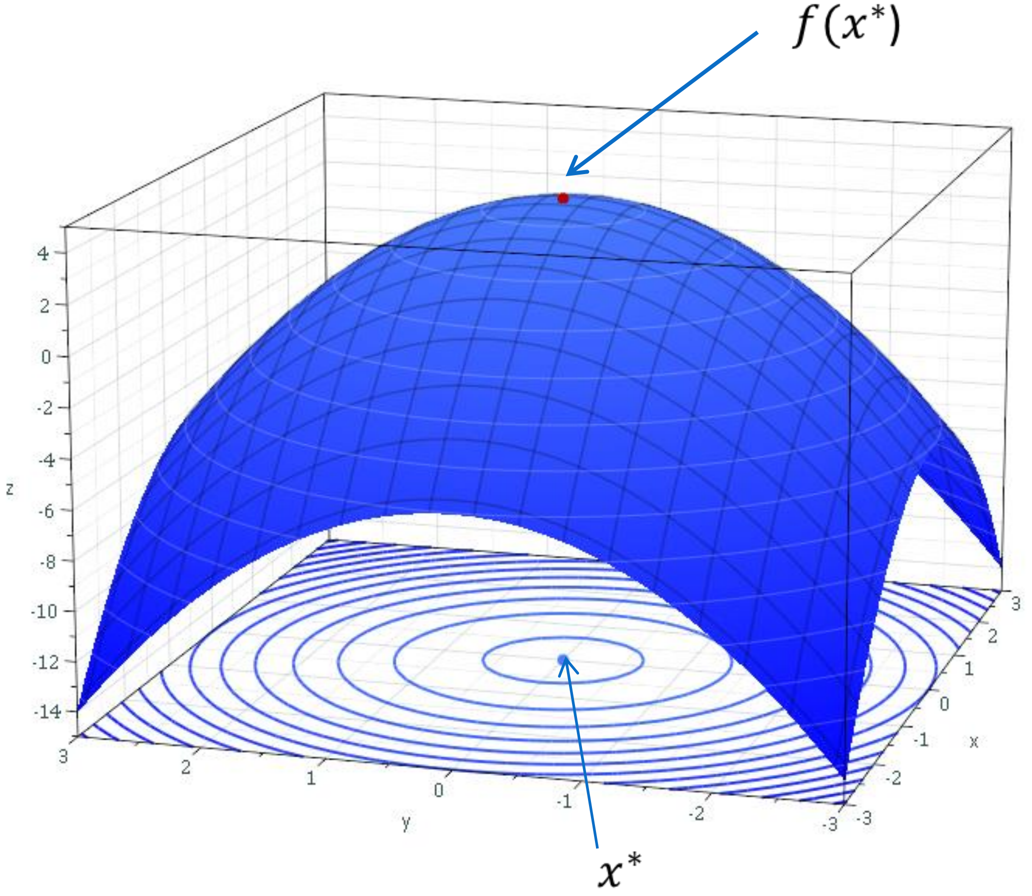
- It is the point x^* at which all first derivatives of $f(x)$ vanish i.e. $\nabla f(x^*) = 0$ where $x^* = (x^*{}_1, x^*{}_2, \ldots\ldots x^*{}_n) \in R^n$

$$\& \ \nabla f(x^*) = \begin{bmatrix} \frac{\partial f(x^*)}{\partial x_1} \\ \frac{\partial f(x^*)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x^*)}{\partial x_n} \end{bmatrix}$$ is called gradient vector of f $at\ x^*$.

Depending on the behavior of f at a stationary point x^* ,it is classified into following types:

1. Global maxima
2. Global minima
3. Local maxima
4. Local minima
5. Saddle point

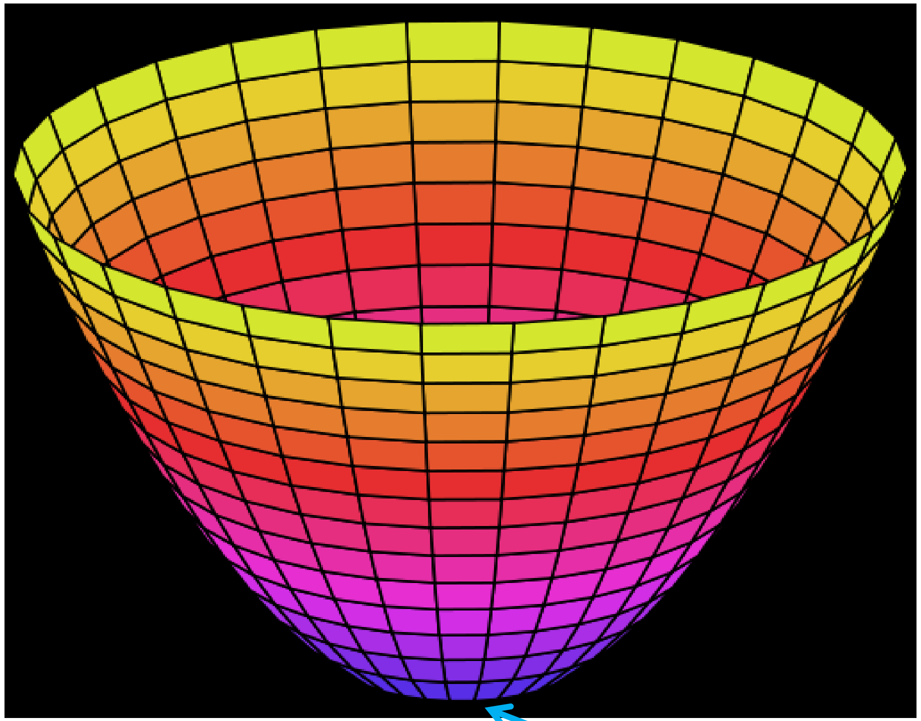
Global Maximum or maximum point:



x^* is called (global)maximum point if $f(x^*) \geq f(x) \quad \forall x \in R^n$

Global Minimum or minimum point:

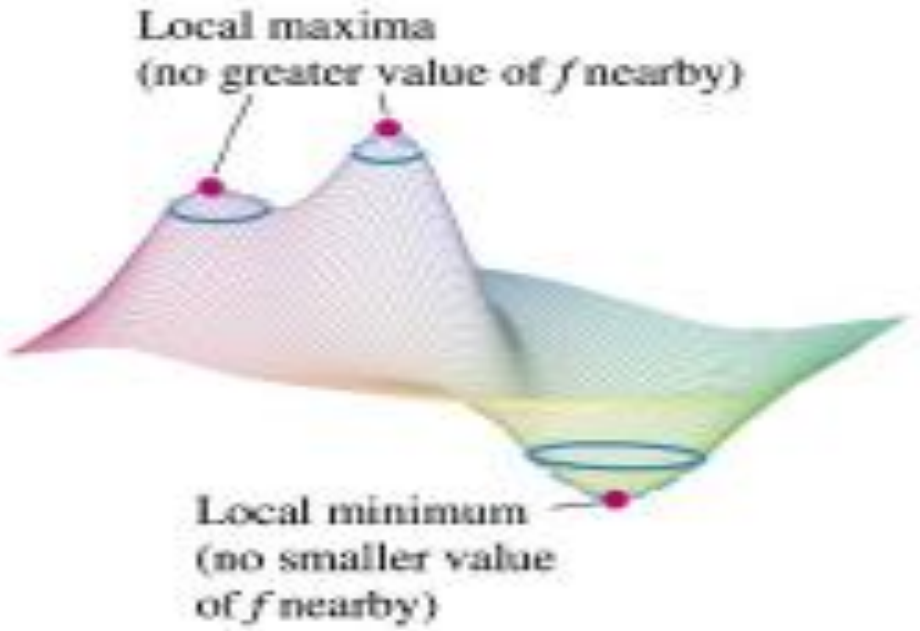
•



$f(x_*)$

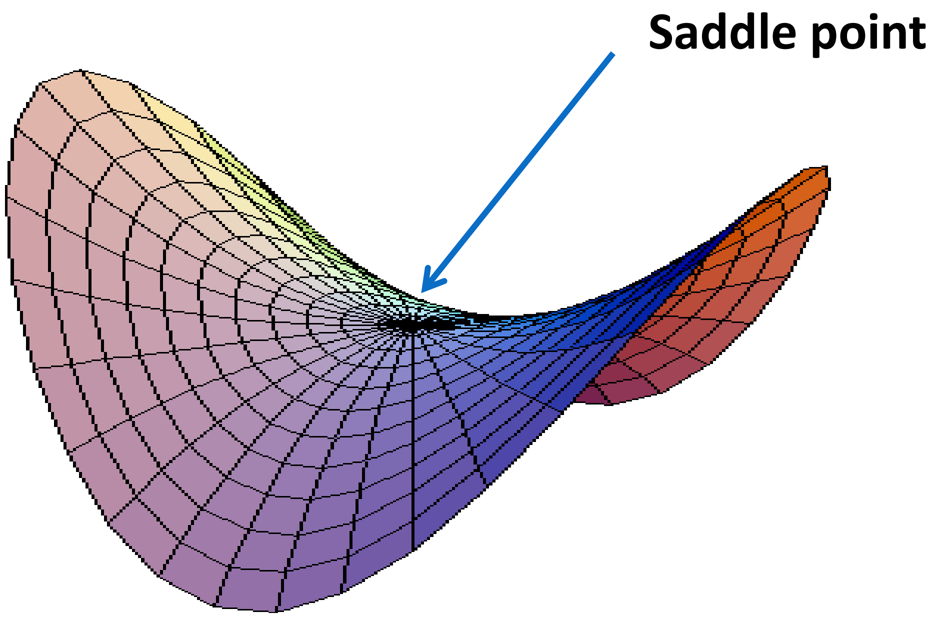
x^* is called (global)minimum point if $f(x^*) \leq f(x) \qquad \forall x \in R^n$

Relative Maximum & minimum point:



- x^* is called local maximum point if there exists an open ball centered at x^* such that for all x in this open ball the condition $f(x^*) \geq f(x)$ is satisfied.
- x^* is called local minimum point if there exists an open ball centered at x^* of such that for all x in this open ball the condition $f(x^*) \leq f(x)$ is satisfied.

Saddle point:

- 

- If x^* is neither (global/local)maximum nor (global/local)minimum is called a saddle point.

Optimization of function of one variable

- Problem: Minimize $f(x) \qquad x \in R$
- Solution:
  - ➢Find stationary points x^* by solving $f'(x) = 0$
  - ➢At each stationary point evaluate $f''(x)$
  - ➢Draw conclusion based on following table:

$f''(x^*)$	**Conclusion**
>0	Minima
<0	Maxima
0	Saddle point/inconclusive

- Example: Minimize $f(x) = x^2 + 3x + 2$

Optimization of function of two variables

- Problem: Minimize $f(x)$, $x = (x_1, x_2 \ldots \ldots .. x_n) \in R^n$
- Solution:
  - Find stationary points x^* by solving $\nabla f(x) = 0$
  - At each stationary point evaluate Hessian matrix $H(x^*)$ which is a matrix containing all second order partial derivatives of f w.r.t. $x_1, x_2 \ldots \ldots .. x_n$

$$H(x^*) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(x^*) & \frac{\partial^2 f}{\partial x_1 x_2}(x^*) & \cdots & \frac{\partial^2 f}{\partial x_1 x_n}(x^*) \\ \frac{\partial^2 f}{\partial x_2 x_1}(x^*) & \frac{\partial^2 f}{\partial x_2^2}(x^*) & & \frac{\partial^2 f}{\partial x_2 x_n}(x^*) \\ & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1}(x^*) & \frac{\partial^2 f}{\partial x_n x_1}(x^*) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(x^*) \end{bmatrix}_{n \times n}$$

- Draw conclusion based on following table:

$H(x^*)$	**Conclusion**
Positive definite	Minimum
Negative definite	Maximum
indefinite	Saddle point
Semidefinite	No conclusion

- Recall:

A square matrix is said to be

- positive definite if all its eigen values are positive(>0) ,
- negative definite if all its eigen values are negative(<0)
- Positive semidefinite if all its eigen values are ≥ 0
- Negative semidefinite if all its eigen values are ≤ 0
- indefinite if it has both positive & negative eigen values.

- Example: $f(x,y) = x^3 + y^3 + 3xy$
- Solution: $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 3x^2 + 3y \\ 3y^2 + 3x \end{bmatrix}$ & $H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 6x & 3 \\ 3 & 6y \end{bmatrix}$

Apply first derivative test to find stationary points:

$\nabla f = 0$ => $\begin{bmatrix} 3x^2 + 3y \\ 3y^2 + 3x \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ => $x^2 = -y$ & $y^2 = -x$

Solving simultaneously we get $(0,0)$ & $(-1,-1)$ as stationary points.
Now to check maxima and minima we evaluate Hessian matrix at these points

$H(0,0) = \begin{bmatrix} 0 & 3 \\ 3 & 0 \end{bmatrix}$ Eigen values of these matrix are 3 & -3(Check) i.e H is indefinite which implies that $(0,0)$ is saddle point

$H(-1,-1) = \begin{bmatrix} -6 & 3 \\ 3 & -6 \end{bmatrix}$ Eigen values of these matrix are -9 & -3(Check) i.e H is negative definite which implies that $(-1,-1)$ is a point of maxima.

Application: Least squares

- In machine learning models used for prediction, often we need to solve the systems $Ax = b$
- This system is many a times inconsistent, in which case we have to look for the solution such that the difference between Ax & b(called residual) is minimized i.e. Solve the optimization problem $\min ||Ax - b||_p$

  i.e $\min ||Ax - b||_p^2$

For $p = 2$, the problem is called (L_2)***least squares*** & the term $||Ax - b||_2^2$ is called ***residual sum of squares(RSS)***

- Example: Solve $Ax = b$ where

$$A = \begin{bmatrix} 2 & 0 \\ -1 & 1 \\ 0 & 2 \end{bmatrix} \quad \& \; b = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

- **Solution:**

Observe that the system is inconsistent.

We will find the least squares solution.

The objective function is

$$\|Ax - b\|_2^2 = (2x - 1)^2 + (-x + y)^2 + (2y + 1)^2$$

Finding the minima by the method seen earlier , we get the solution as $x = \frac{1}{3}$, $y = -\frac{1}{3}$ (Check)

Algorithmic approach:

- In practice, computing and storing the full Hessian matrix takes large memory, which is infeasible for high-dimensional functions such as the loss function with large numbers of parameters. For such situations, first order ***algorithmic methods*** like ***gradient descent*** or second order methods like Newton's method have been developed.

- General structure of algorithms for unconstrained minimization :
- Choose a starting point x_0
- Beginning at x_0, generate a sequence of iterates $\{x_k\}_{k=0}^{\infty}$ with non-increasing function f value until a solution point with sufficient accuracy is found or until no further progress can be made.
- To generate the next iterate x_{k+1}, the algorithm uses information about the function at x_k and possibly earlier iterates.
-

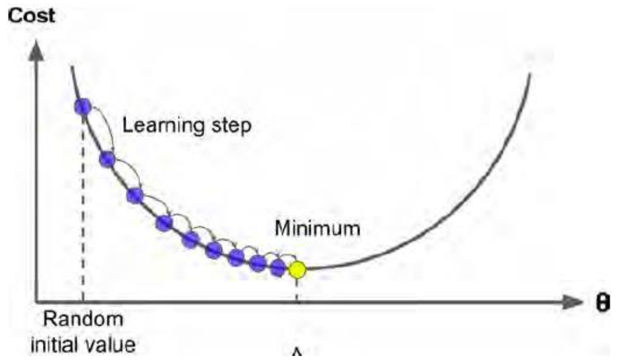
Gradient Descent:

- Algorithmic method for finding a local minimum of a differentiable function.
- The algorithm is initiated by choosing random values to the parameters
- Improve the parameters gradually by taking steps proportional to the negative of the gradient (or approximate gradient) of the cost function at the current point.
- Continue the process until the algorithm converges to a minimum i.e until the difference between the successive iterates becomes stable or reaches a threshold

❖ Note: If we instead take steps proportional to the *positive* of the gradient, we approach a local maximum of that function; the procedure is then known as **gradient ascent**.

Analogy:

- To get an idea of how Gradient Descent works, let us take an example. Suppose you are at the top of a mountain and want to reach the base camp which is all the way down at the lowest point of the mountain. Also, due to the bad weather, the visibility is really low and you cannot see the path at all. How would you reach the base camp?
- One of the ways is to use your feet to know where the land tends to descend. This will give an idea in what direction, the steep is low and you should take your first step. If you follow the descending path until you encounter a plain area or an ascending path, it is very likely you would reach the base camp.



Limitation:

- If there is a slight rise in the ground when you are going downhill you would immediately stop assuming that you reached the base camp (global minima), but in reality, you are still stuck at the mountain at a local minima.
- In other words , gradient descent does not guarantee finding global mimina(maxima) of the function
- However most of the objective functions used in machine learning such as cost function are convex functions which ensure that the local minimum is also a global minimum.

Mathematical formulation of the idea:

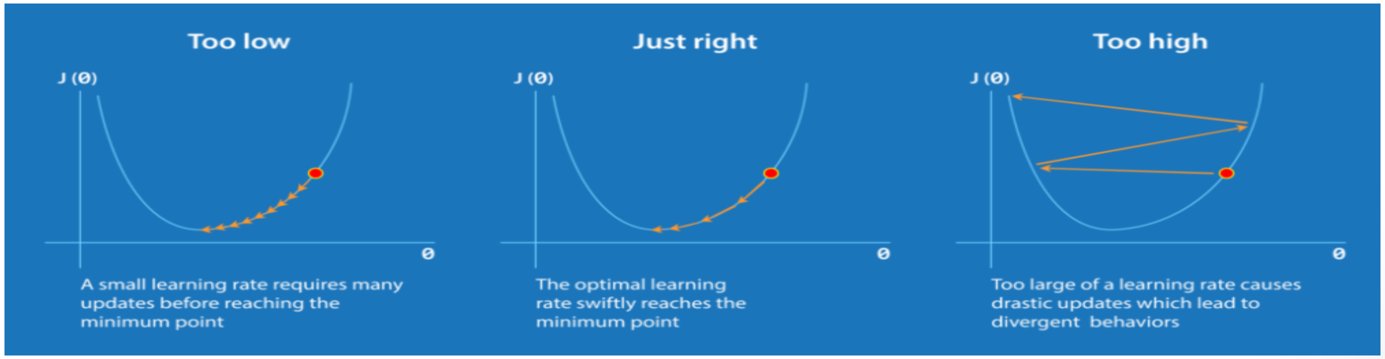
- The algorithm is based on the fact that at any given point $'x_k'$ in the domain of the function $f(x)$, the function decreases fast in the direction of negative gradient and increases in the opposite direction.
- If one goes from position a_k to position a_{k+1} by going in the direction of negative gradient with step size/length γ i.e. $a_{k+1} = a_k - \gamma \nabla f(a_k)$ then he will be going towards the minimum.
- It can be shown that if γ is small then $f(a_k) \geq f(a_{k+1})$ .So if x_0 is the starting point of the algorithm followed by the sequence $x_1, x_2, x_3 \ldots \ldots$ then $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$ $where\ f(x_0) \geq f(x_1) \geq f(x_2) \ldots.$

Algorithm in machine learning

- In machine learning code we normally use the following notations :
- Cost function is denoted by $J(\theta)$ where $= \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$, θ_i is the i^{th} parameter and learning rate is denoted by α
- so that the iterative formula becomes $\theta \coloneqq \theta - \alpha \nabla J(\theta)$. If we apply this formula individually to the components of θ then the formula for j^{th} component θ_j becomes
- $\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$

Note on step size:

- The value of step size γ can be changed at every iteration.(Hence the notation γ_k).
- In machine learning the value γ is called the learning rate(which can be varied).
- Usually, we take the value of the learning rate to be small such as 0.1, 0.01,0.001 etc..
- The value of the step should not be too big as it can skip the minimum point and thus the optimization can fail. It is a hyper-parameter and you need to experiment with its values.


Too low
J (θ)
θ
A small learning rate requires many updates before reaching the minimum point
Just right
J (θ)
θ
The optimal learning rate swiftly reaches the minimum point
Too high
J (θ)
θ
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Example:Single variable case

- Minimize $f(x) = x^2$

Solution:

We are given a function of one variable. Here cost function $J(\theta) = \theta^2$ and there is only one parameter so that $\theta = [\theta_1]$

From our cost function $J(\theta)$, we can clearly say that it will be minimum at $\theta = 0$, but it won't be so easy to derive such conclusions while working with some complex functions, so we will apply gradient descent here.

Step 1: Initialize θ by a random number say $\theta = 5$ and let the learning rate $\gamma = 0.1$

Step 2: Simplification of the iteration formula: $\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$

$$\theta := \theta - \alpha \frac{\partial J}{\partial \theta}$$
$$\theta := \theta - (0.1) \frac{\partial(\theta^2)}{\partial \theta}$$
$$\theta := \theta - (0.1) * (2\theta)$$
$$\theta := 0.8 * \theta$$

- (F) Table Generation:
- Here we are stating with θ = 5.keep in mind that here θ = 0.8*θ, for our learning rate and cost function.

θ	$J(\theta)$
5	25
4	16
3.2	10.24
2.56	6.55
2.04	4.19
⋮	⋮
0	0

θ	$J(\theta)$
-5	25
-4	16
-3.2	10.24
-2.56	6.55
-2.04	4.19
⋮	⋮
0	0

- We can see that, as we increase our number of iterations, our cost value goes down and the algorithm converges to the optimum value 0

Example:Two variable case

- Example 2(Two Variables case):
- Minimize $f(x, y) = x^2 + y^2$
- Solution:
- Our cost function is : $J(\theta) = {\theta_1}^2 + {\theta_2}^2$ where $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ & let the learning rate $\alpha = 0.1$

https://www.youtube.com/watch?v=IeRFW4_GLYA

- Increment function: $\theta_1 := \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$ & $\theta_2 := \theta_2 - \alpha \frac{\partial J}{\partial \theta_2}$
- $\theta_1 := \theta_1 - (0.1) * \frac{\partial({\theta_1}^2 + {\theta_2}^2)}{\partial \theta_1}$ & $\theta_2 := \theta_2 - (0.1) * \frac{\partial({\theta_1}^2 + {\theta_2}^2)}{\partial \theta_2}$
- $\theta_1 := \theta_1 - (0.1) * (2\theta_1)$ & $\theta_2 := \theta_2 - (0.1) * (2\theta_2)$
- $\theta_1 := (0.8) * \theta_1$ & $\theta_2 := (0.8) * \theta_2$

- Initialize $\theta_1 = 1$ & $\theta_2 = 1$ and iterate

θ_1	θ_1	$J(\theta)$
1	1	2
0.8	0.8	1.28
0.64	0.64	0.4096
0.512	0.512	0.2621
0.4096	0.4096	0.1677
$\vdots$	$\vdots$	$\vdots$
0	0	0

- We can see that, as we increase our number of iterations, our cost value goes down and the algorithm slowly converges to the optimum value $(0,0)$