Numerical Optimization for ML&DL (NOFML&DL)

Artificial Intelligence and Data Science

شرح بالعربي

Agenda

- Why Shall We Study Numerical Optimization?
- The Machine Learning Problem.
- Analytical, Algebraic and Numerical Solutions.
- Numerical Optimization.
- Single Variable Linear Regression.
- Vector Norms and Loss Function
- Gradient
- Gradient Descent (GD) Algorithm.

Agenda

Convex Function.

Gradient of Multivariable Function.

Contour Plot.

Local vs. Global Minimum.

GD Applied to LR (Single Variable).

The Machine Learning Problem

Data at hand is just a sample data.

Definite solution does not generalize the model.

We need to find an approximate solution to generalize the model.

Learning is no more than finding the optimal model parameters that minimizing the prediction error.

Analytical Solution

- An analytical solution involves framing the problem in a well-understood form and calculating the exact solution. **(can be done by hand)**.
- We prefer the analytical method in general because it is faster and because the solution is exact.
- If our data matrix is square, we can find model parameters by solving system of linear equations $\boldsymbol{X\theta = y \Longrightarrow \theta = X^{-1}y}$
- **However, this solution is not general. i.e., it is only available for the data at hand (training data).**

Approximate Solution

- In general, data matrix ($\boldsymbol{X_{m\times n}}$) is not square (we need it to be like that).i.e., $\boldsymbol{m > n}$ ($\boldsymbol{no\ sol.}$) $\boldsymbol{or\ m < n}$($\boldsymbol{infinite\ sol.}$).
- In this case we will not have a definite solution.
- In order to solve this problem, we will use approximation. i.e., find an approximate solution.

Algebraic Solution

- In case of linear models (e.g., linear regression), we can obtain the approximate solution algebraically.
- For $\boldsymbol{m} > \boldsymbol{n}$ (overdetermined system) we can use least square method

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\mathbf{1}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$$

- However, if $\boldsymbol{n}$ is large $\left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\mathbf{1}}$ will be computationally expensive.

Algebraic Solution

- For $\boldsymbol{m} < \boldsymbol{n}$ (underdetermined system) we can use least norm method

$$\boldsymbol{\theta} = \boldsymbol{X}^{\boldsymbol{T}}\left(\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}\right)^{-\mathbf{1}}\boldsymbol{y}$$

- However, if $\boldsymbol{m}$ is large $\left(\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}\right)^{-\mathbf{1}}$ will be computationally expensive.
- In addition, if the relation between input and output $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ is nonlinear, analytical and algebraic solutions do not exist. E.g., in case of high order polynomial functions and of course in ANN.

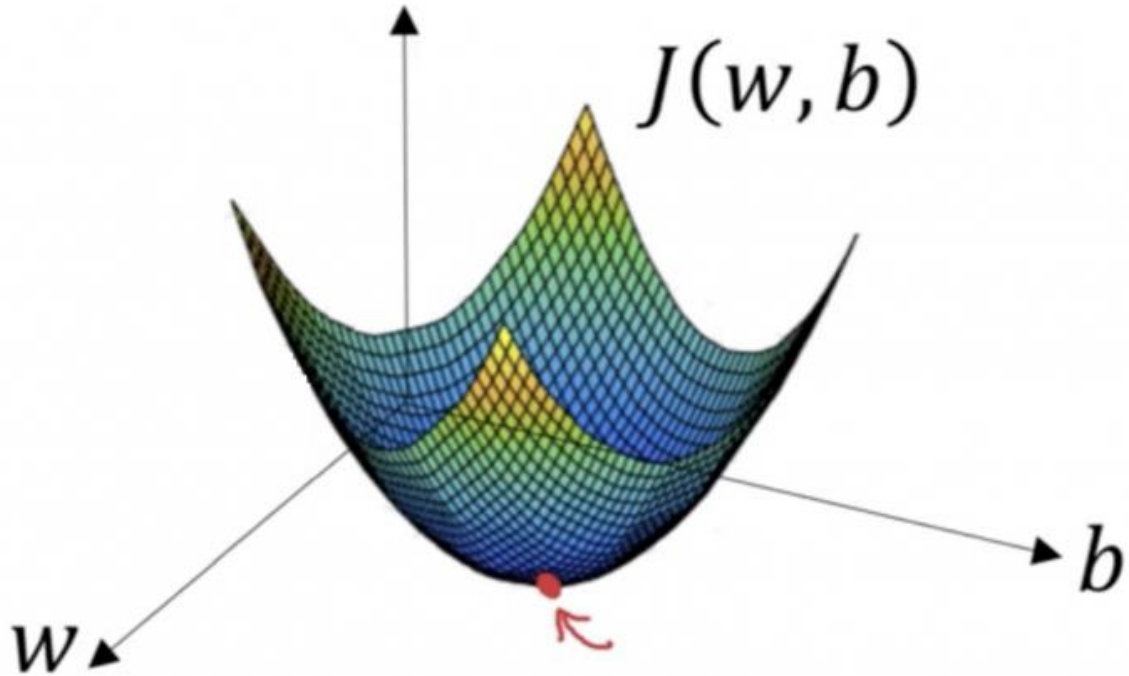
Numerical Solution

- For linear model $\hat{\boldsymbol{y}} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x}$ we need to find the optimum values of the parameters $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$ that minimizes the prediction error $|\hat{\boldsymbol{y}} - \mathbf{y}|$
- Loosely speaking, numerical solution involves the following steps:

1. Parameters initialization (assume $\boldsymbol{\theta_0} = \boldsymbol{\theta_1} = \mathbf{0}$).
2. Predict the output using $\hat{\boldsymbol{y}} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x}$.
3. Evaluate the prediction error using any error function e.g., $|\hat{\boldsymbol{y}} - \mathbf{y}|$.
4. Find the direction and magnitude of changing the model parameters $(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$ to decrease the error.
5. Update model parameters.
6. Go to step 2.

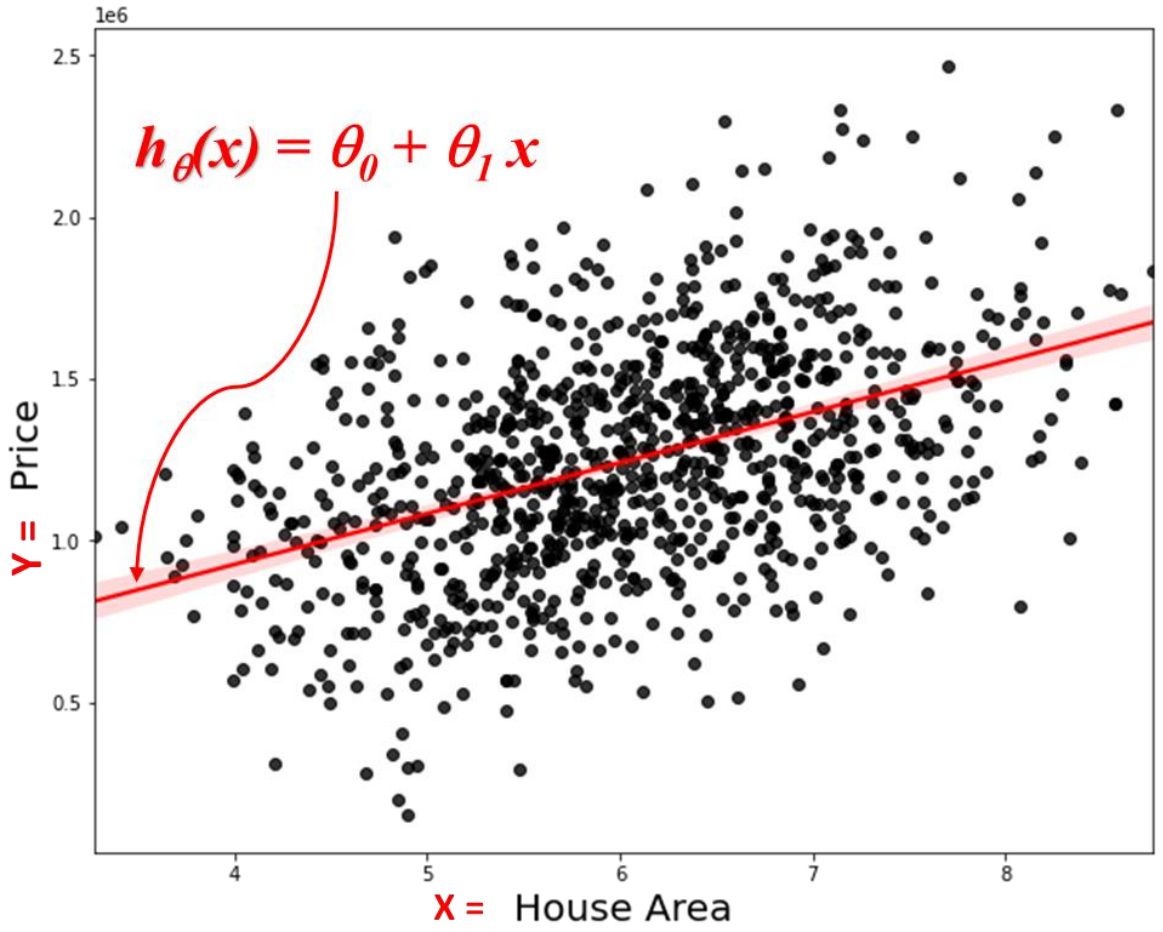
Numerical Optimization

- Keep iterating until we reach the minimum error. i.e., finding the values of model parameters that minimizes/maximizes the loss/cost/error function.
- These parameters range from simple linear regression model to weights and biases for deep neural networks. The same concept is applied.

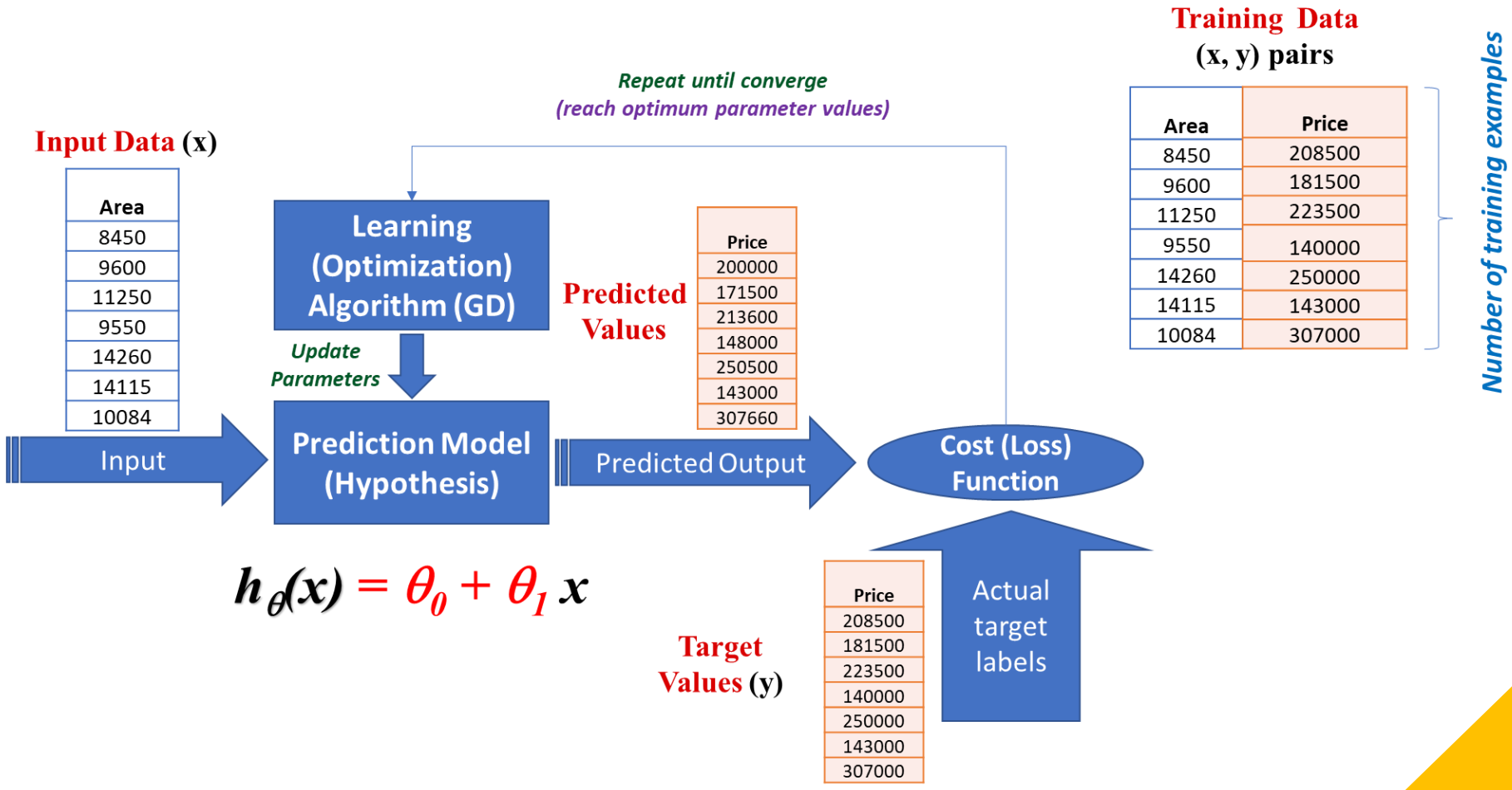


Single Variable Linear Regression (LR)

- $h_\theta(x) = \theta_0 + \theta_1 x$ *(hypothesis)*



Single Variable Linear Regression (LR)



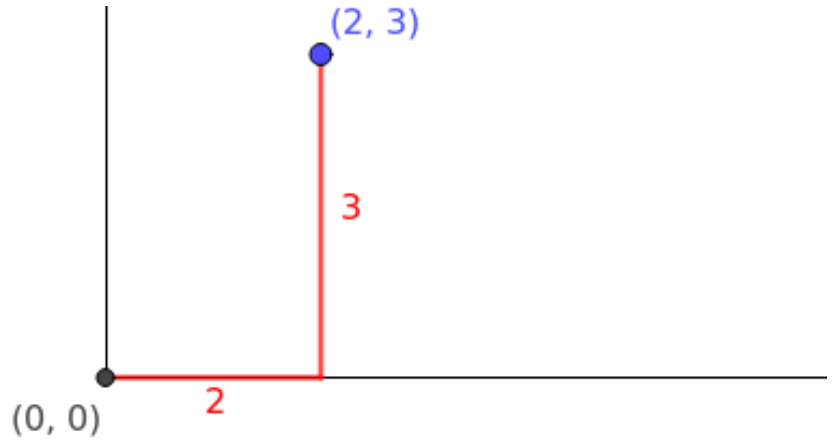
Numerical Solution

- For linear model $\hat{y} = \theta_0 + \theta_1 x$ we need to find the optimum values of the parameters θ_0 and θ_1 that minimizes the prediction error $|\hat{y} - y|$
- Loosely speaking, numerical solution involves the following steps:

1. Parameters initialization (assume $\theta_0 = \theta_1 = 0$).
2. Predict the output using $\hat{y} = \theta_0 + \theta_1 x$.
3. **Evaluate the prediction error using any error function e.g., $|\hat{y} - y|$.**
4. **Find the direction and magnitude of changing the model parameters (θ_0, θ_1) to decrease the error.**
5. **Update model parameters.**
6. Go to step 2.

Vector Norm

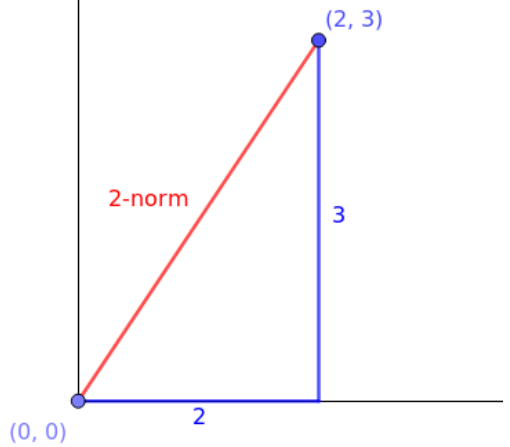
- ***L¹ Norm (Manhattan norm)***
- The L¹ norm is often referred to as the Manhattan/Taxicab Distance, the Mean Absolute Error (MAE), or the Least Absolute Shrinkage and Selection Operator (LASSO).
- The L1 norm that is calculated as the sum of the absolute values of the vector.



$$\|\mathbf{x}\|_1 = \sum_{i=1}^{N} |x_i| = |x_1| + |x_2| + \ldots + |x_N|$$

Vector Norm

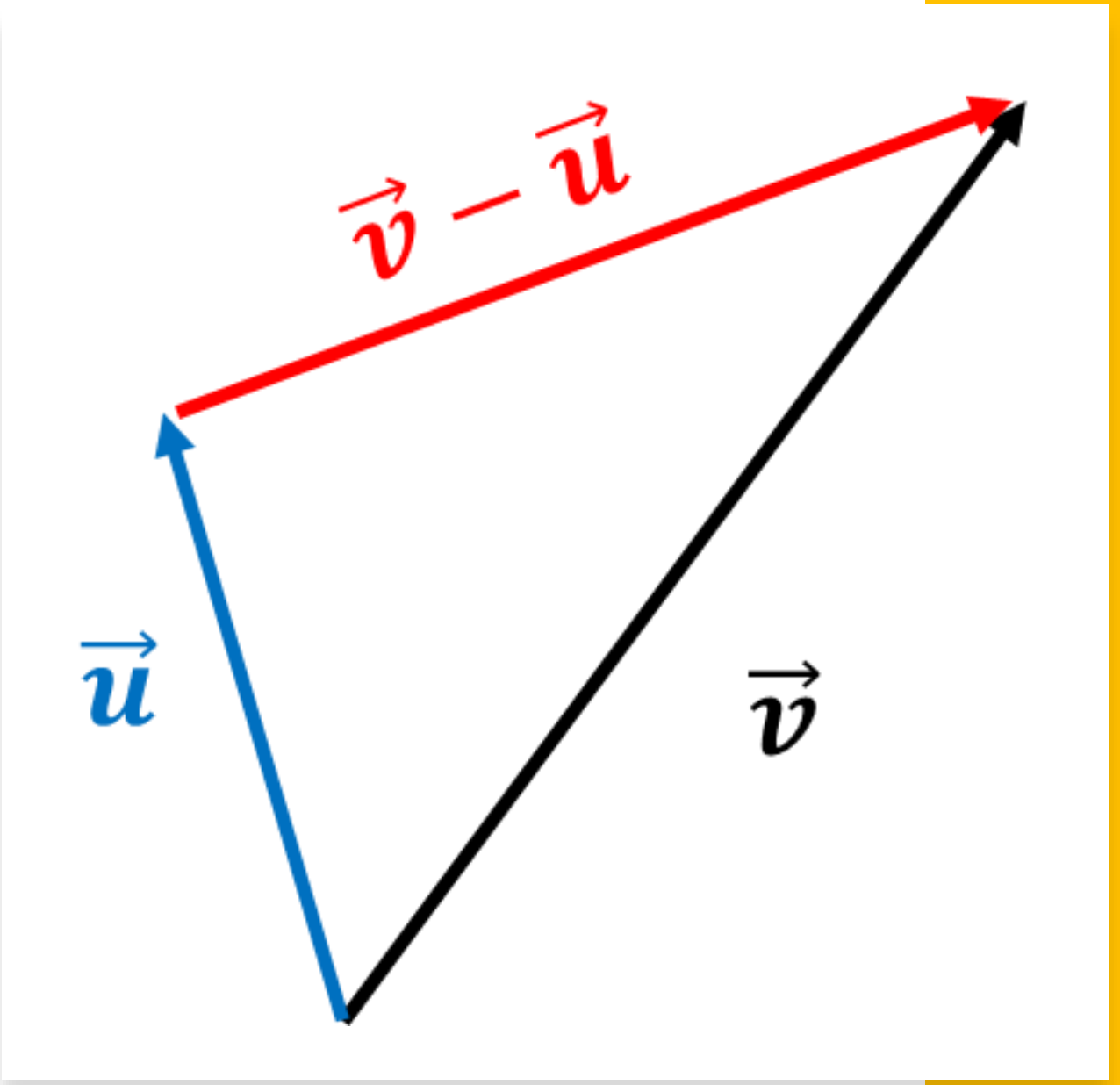
- ***L² Norm (Euclidean Norm)***
- The L² norm is often referred to as the Euclidean Distance, the Mean Squared Error (MSE)/ Least Squares Error, or the Ridge Operator.
- The L2 norm finds the shortest path from point A to point B.
- The L2 norm that is calculated as the square root of sum of squares of the vector components.

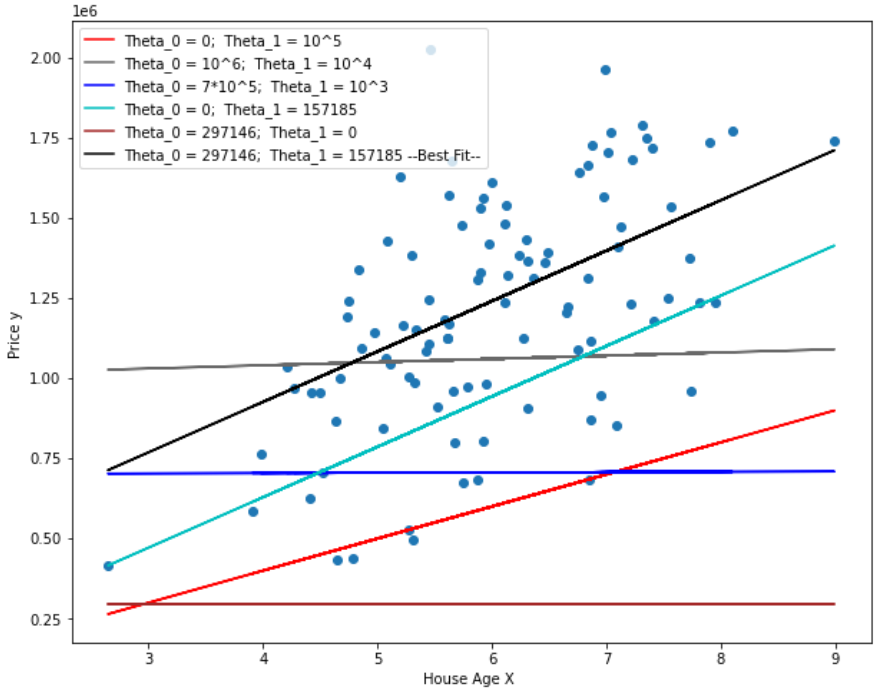


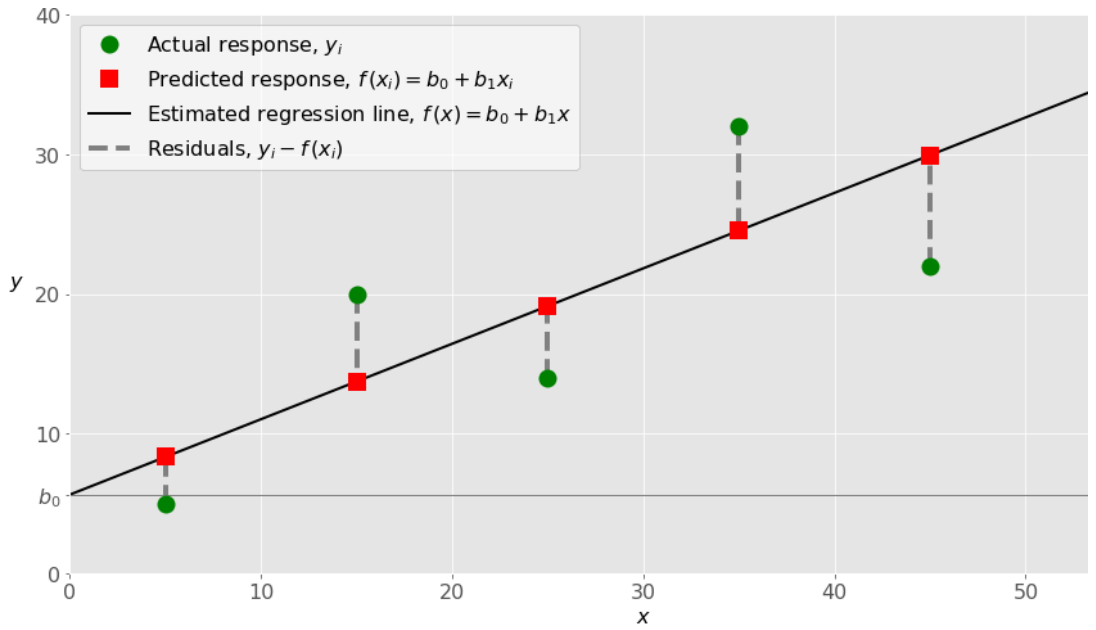
$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{N} |x_i|^2\right)^{1/2} = \sqrt{x_1^2 + x_2^2 + \ldots + x_N^2}$$

Vector Norm

- Distance between $\vec{v}$ and $\vec{u}$ is the norm of the resultant vector from their subtraction $\|\vec{v} - \vec{u}\|$.





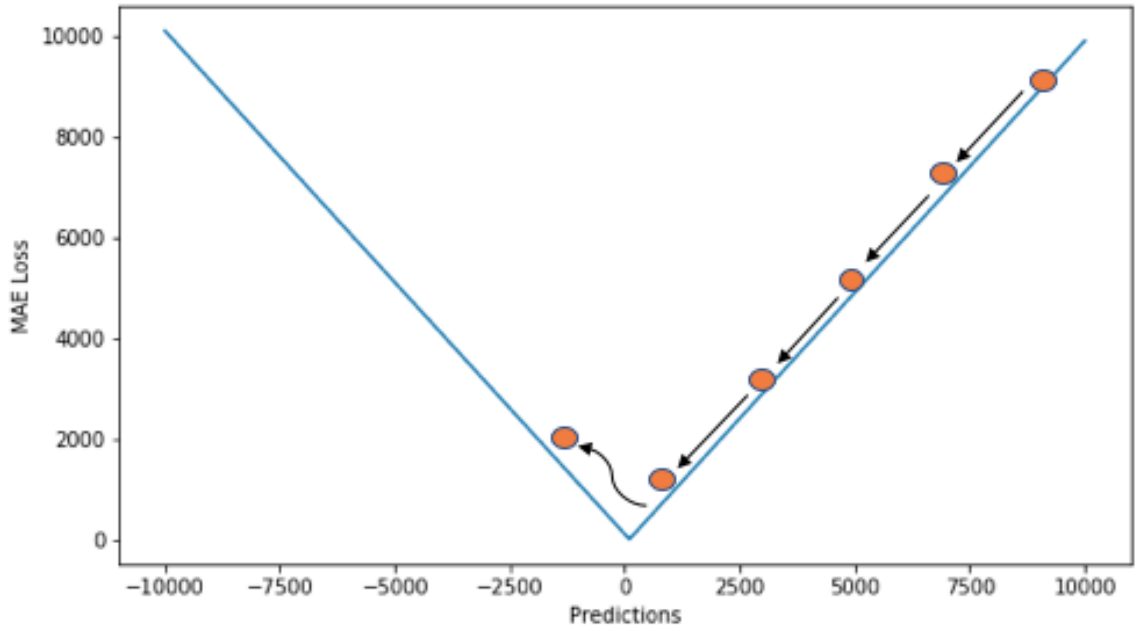


Loss (Cost) Function

- Residual is a norm of two vectors subtraction.
- Each parameter pairs give different fit.

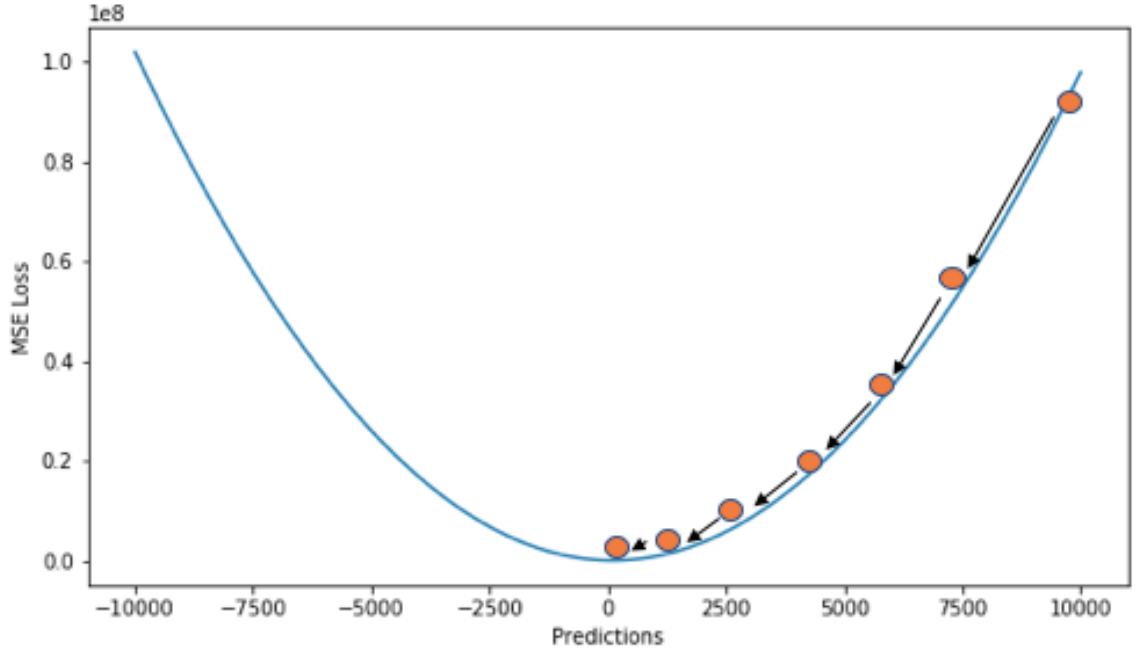
Loss (Cost) Function

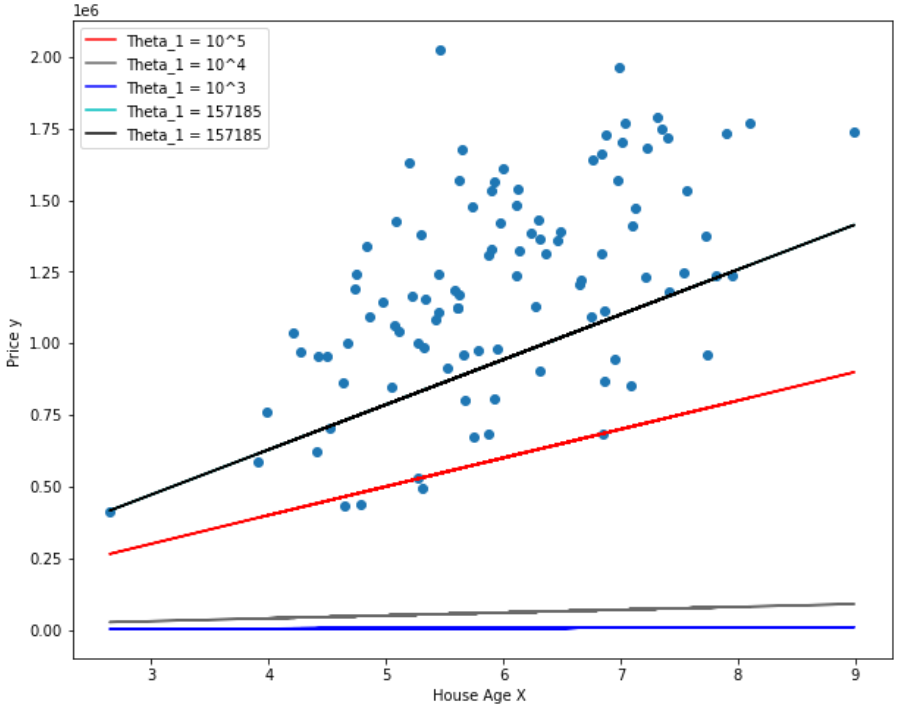
- **Mean Squared Error (MAE) (L1 Loss Function)** is used to minimize the error which is the sum of the all the **absolute** differences between the true value and the predicted value.
- $\boldsymbol{MAE = \frac{1}{m}\sum_{i=1}^{m}|\hat{y} - y|}$
- **MAE is robust to outlier. However, it has a constant gradient, and the gradient is not defined at the minimum.**

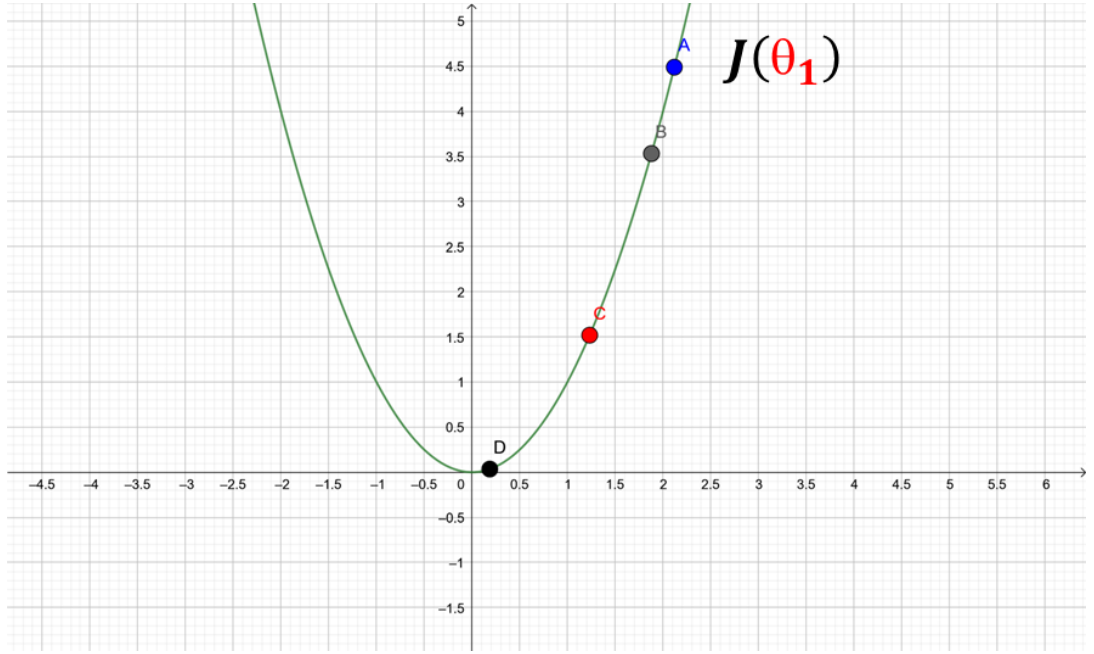


Loss (Cost) Function

- **Mean Absolute Error (MAE) (L2 Loss Function)** is used to minimize the error which is the sum of the all the **squared** differences between the true value and the predicted value.
- $\boldsymbol{MSE = \frac{1}{m}\sum_{i=1}^{m}(\hat{y} - y)^2}$
- **MSE is not robust to outlier. However, it has a changing gradient that enables better learning.**







Loss (Cost) Function

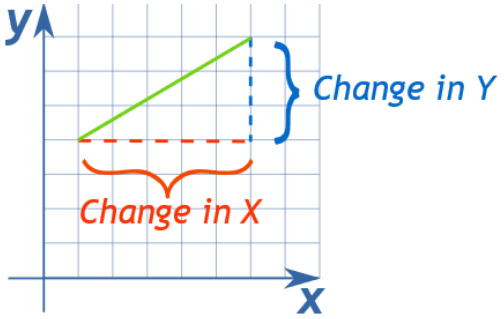
- For simplicity let $\theta_0 = \mathbf{0}$. *i.e.* $\boldsymbol{h}(\theta_1) = \theta_1 \boldsymbol{x}$
- We can plot loss as a function of θ_1 for different fit.

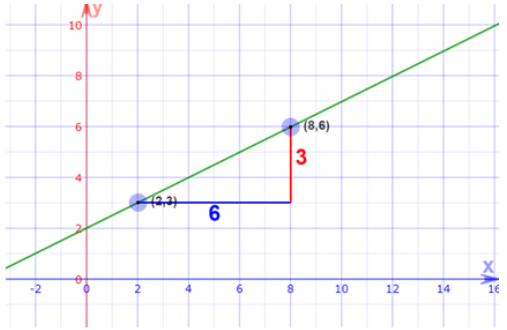
$$\boldsymbol{J}(\theta_1) = \boldsymbol{predicted\ value} - \boldsymbol{actual\ value}$$

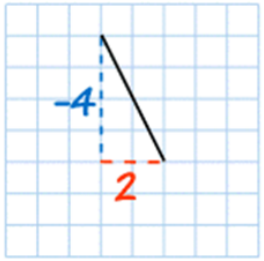
Numerical Solution

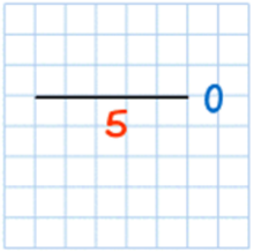
- For linear model $\hat{\boldsymbol{y}} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x}$ we need to find the optimum values of the parameters $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$ that minimizes the prediction error $|\hat{\boldsymbol{y}} - \mathbf{y}|$
- Loosely speaking, numerical solution involves the following steps:

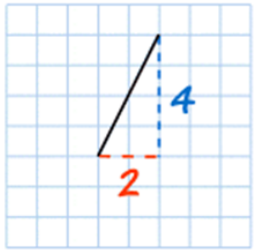
1. Parameters initialization (assume $\boldsymbol{\theta_0} = \boldsymbol{\theta_1} = \mathbf{0}$).
2. Predict the output using $\hat{\boldsymbol{y}} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x}$.
3. Evaluate the prediction error using any error function e.g., $|\hat{y} - y|$.
4. **Find the direction and magnitude of changing the model parameters $(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$ to decrease the error.**
5. Update model parameters.
6. Go to step 2.

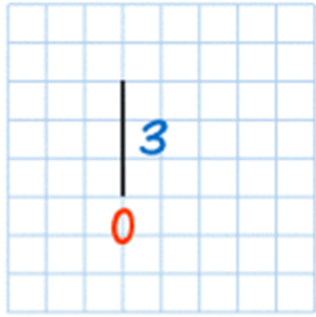


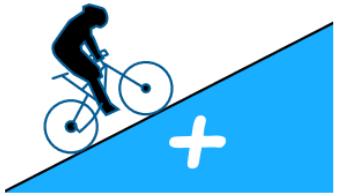












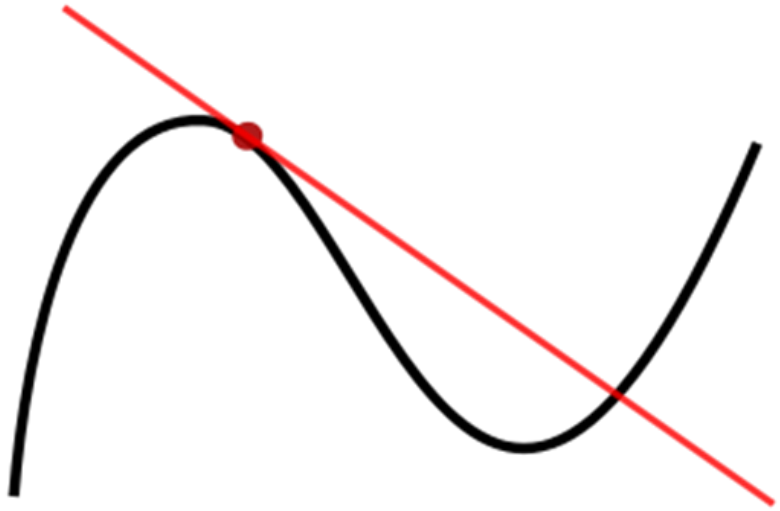
Undefined
Divide by zero

Gradient

- **What is the gradient?!**
- **Gradient is the slope**
- $\boldsymbol{Gradient = \frac{Change\ in\ Y}{Change\ in\ X} = \frac{Rise}{Run}}$

Gradient

- The **derivative** of a function of a real variable measures the sensitivity to change of the function value (output value) with respect to a change in its argument (input value).
- For example, the derivative of the position of a moving object with respect to time is the object's velocity: this measures how quickly the position of the object changes when time advances.

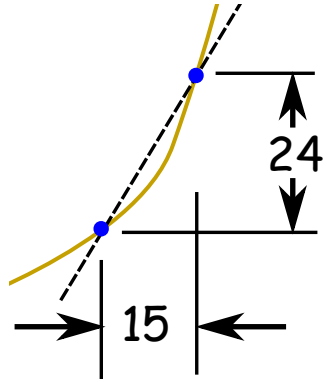


The graph of a function, drawn in black, and a tangent line to that function, drawn in red. The slope of the tangent line is equal to the derivative of the function at the marked point.

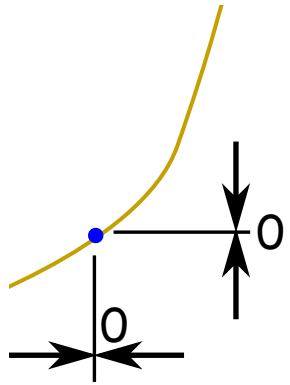
Gradient

- We can find an average slope between two points.
- But how do we find the slope at a point?

  There is nothing to measure!



average slope = $\frac{24}{15}$



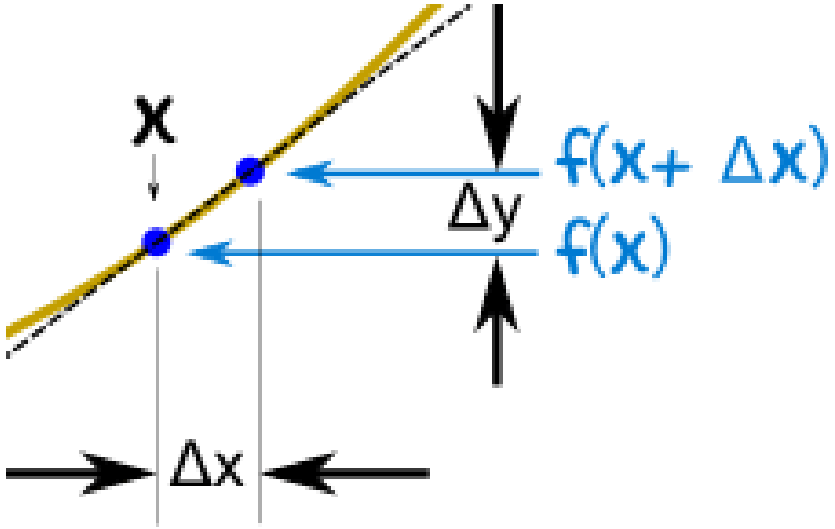
slope = $\frac{0}{0}$ = ???

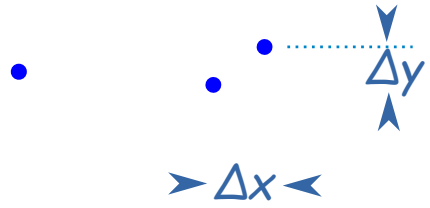
Gradient

- But with derivatives we use a small difference then have it shrink towards zero

- $\frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$
- **Gradient at x = derivative at x**

$$= \lim_{\Delta x \to 0} \left(\frac{f(x+\Delta x) - f(x)}{\Delta x} \right)$$





Gradient

Example: the function $f(x) = x^2$

We know $f(x) = x^2$, and we can calculate $f(x+\Delta x)$:

Start with: $f(x+\Delta x) = (x+\Delta x)^2$

Expand $(x + \Delta x)^2$: $f(x+\Delta x) = x^2 + 2x\,\Delta x + (\Delta x)^2$

The slope formula is: $\frac{f(x+\Delta x) - f(x)}{\Delta x}$

Put in $f(x+\Delta x)$ and $f(x)$: $\frac{x^2 + 2x\,\Delta x + (\Delta x)^2 - x^2}{\Delta x}$

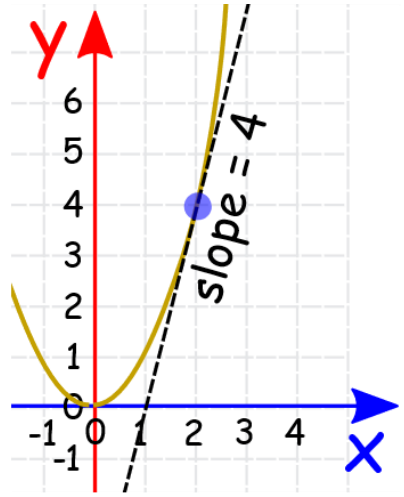
Simplify (x^2 and $-x^2$ cancel): $\frac{2x\,\Delta x + (\Delta x)^2}{\Delta x}$

Simplify more (divide through by Δx): $= 2x + \Delta x$

Then **as Δx heads towards 0** we get: $= 2x$

Result: the derivative of x^2 is $2x$

In other words, the slope at x is $2x$



$$f'(x) = \frac{d}{dx}(x^2) = 2x$$

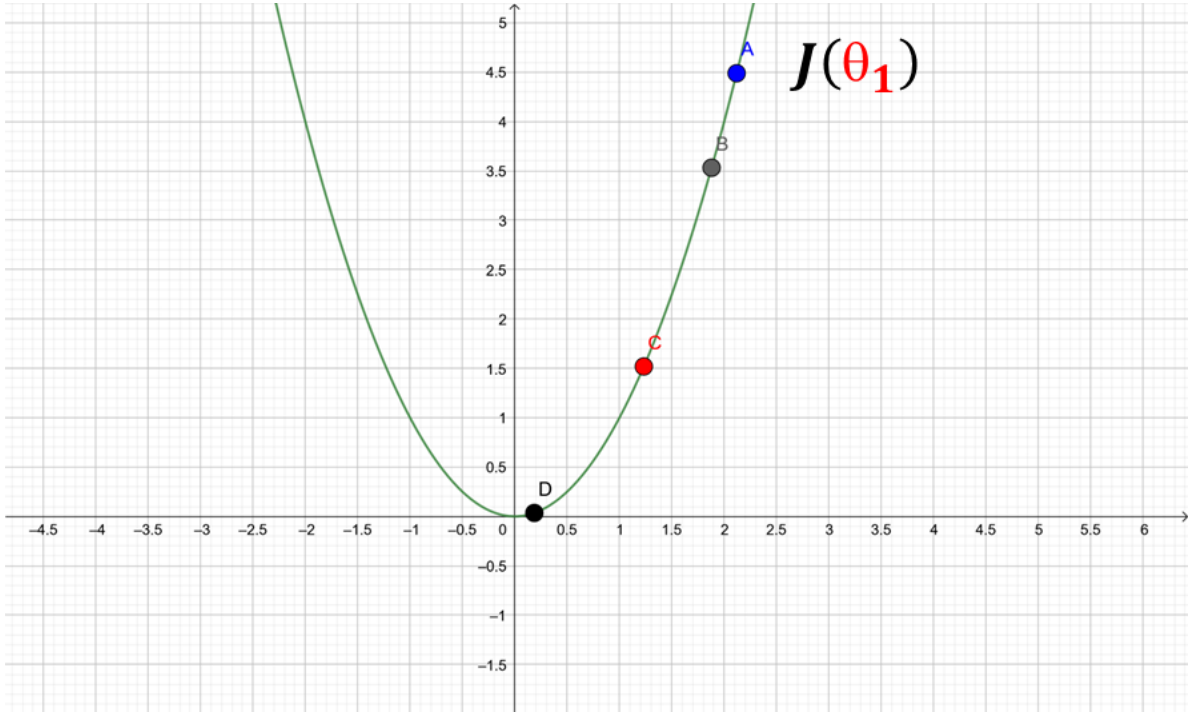
Numerical Solution

- For linear model $\hat{\boldsymbol{y}} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x}$ we need to find the optimum values of the parameters $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$ that minimizes the prediction error $|\hat{\boldsymbol{y}} - \mathbf{y}|$
- Loosely speaking, numerical solution involves the following steps:

1. Parameters initialization (assume $\boldsymbol{\theta_0} = \boldsymbol{\theta_1} = \mathbf{0}$).
2. Predict the output using $\hat{\boldsymbol{y}} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x}$.
3. Evaluate the prediction error using any error function e.g., $|\hat{y} - y|$.
4. Find the direction and magnitude of changing the model parameters (θ_0, θ_1) to decrease the error.
5. **Update model parameters.**
6. Go to step 2.

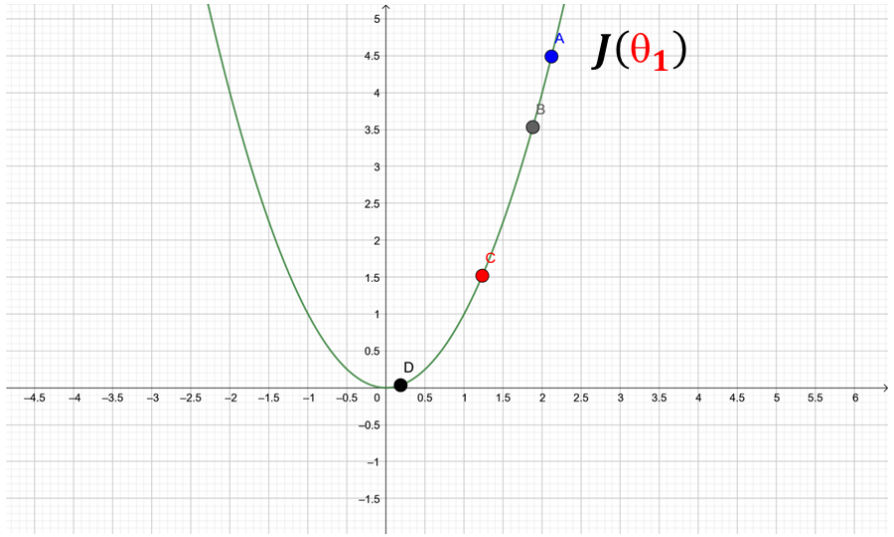
Update Model Parameters

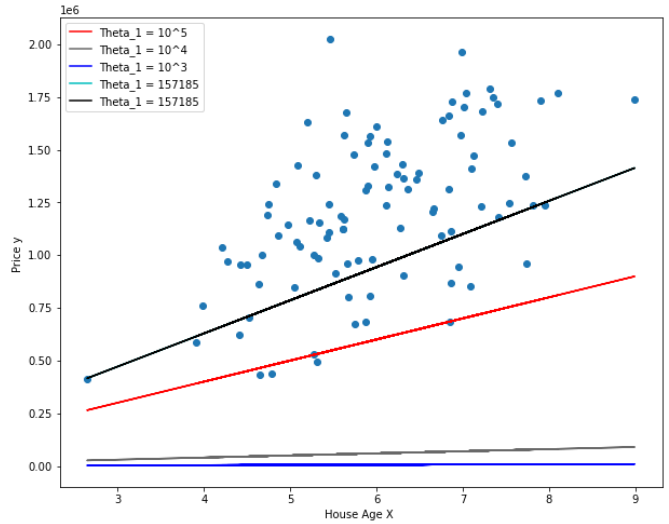
- $\boldsymbol{\theta_{t+1} = \theta_t - update\ term}$



Update Model Parameters

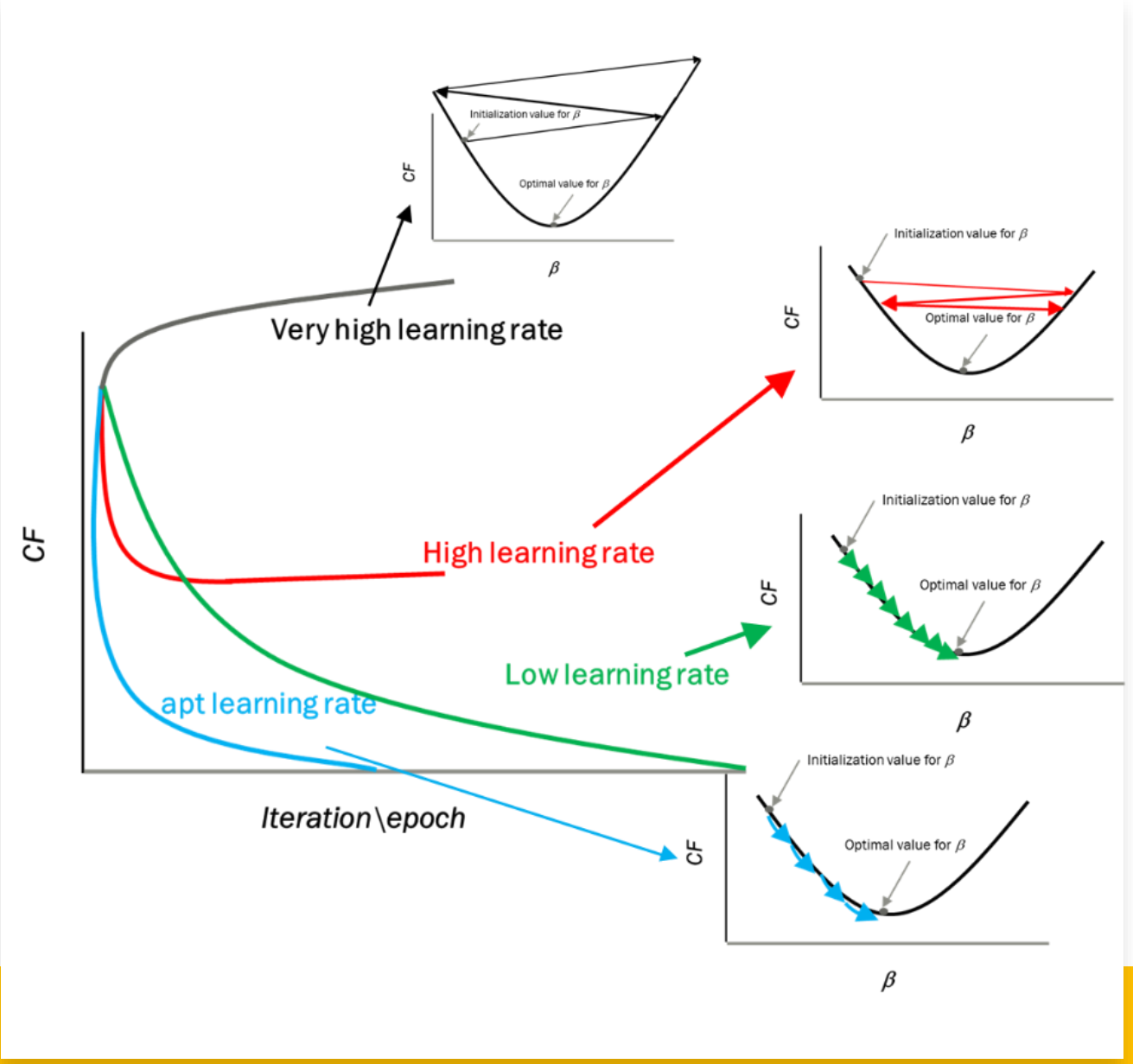
- $\boldsymbol{\theta_{t+1} = \theta_t - gradient}$
- If we have **+ve** gradient $\boldsymbol{\theta}$ will **increase**.
- If we have **–ve** gradient $\boldsymbol{\theta}$ will **decrease**.
- In both cases $\boldsymbol{\theta}$ approaches the value that **minimizes the cost function**.
- However, if we only use the gradient to update the parameter, $\boldsymbol{\theta}$ will **overshoot the minimum and diverge**.

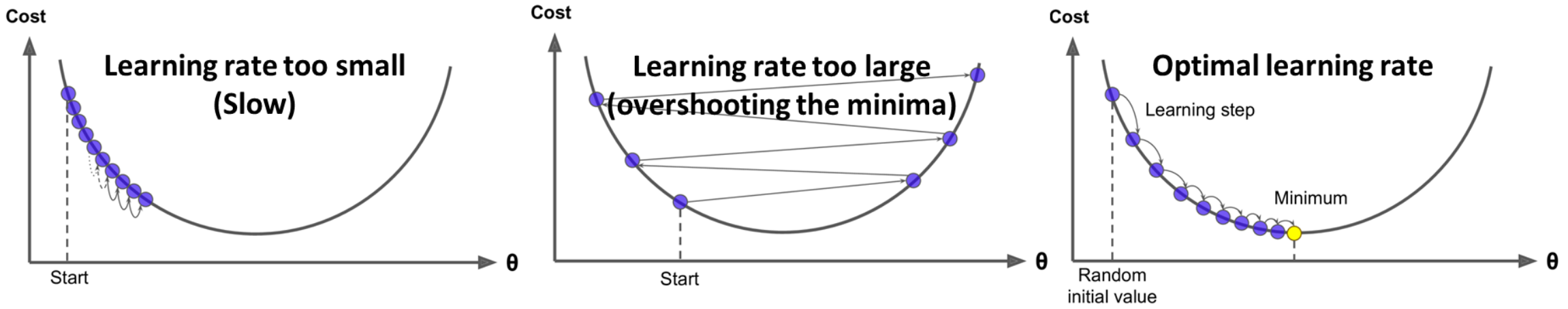




Learning Rate

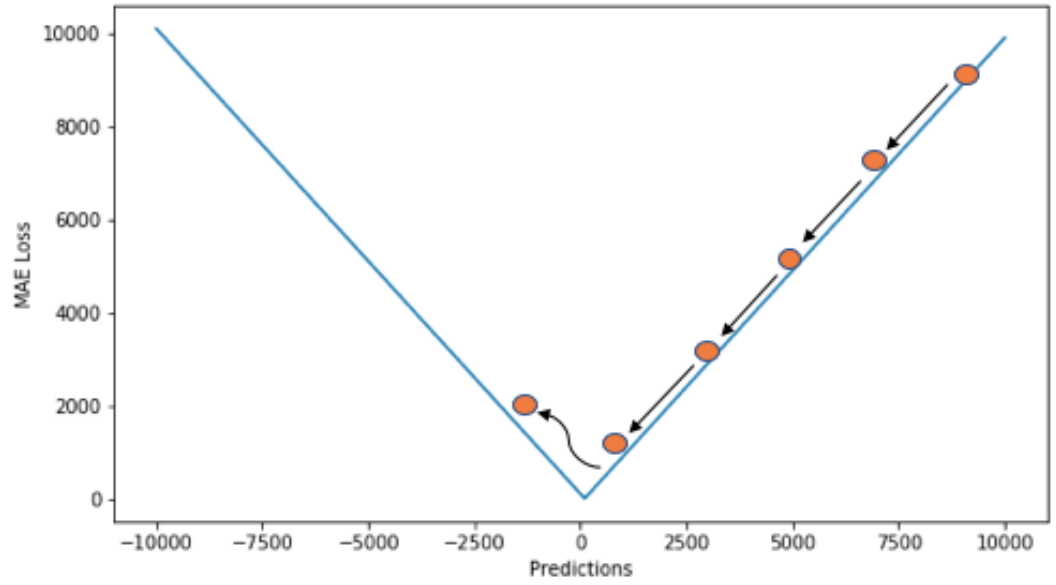
- $\boldsymbol{\theta_{t+1} = \theta_t - \alpha * gradient}$

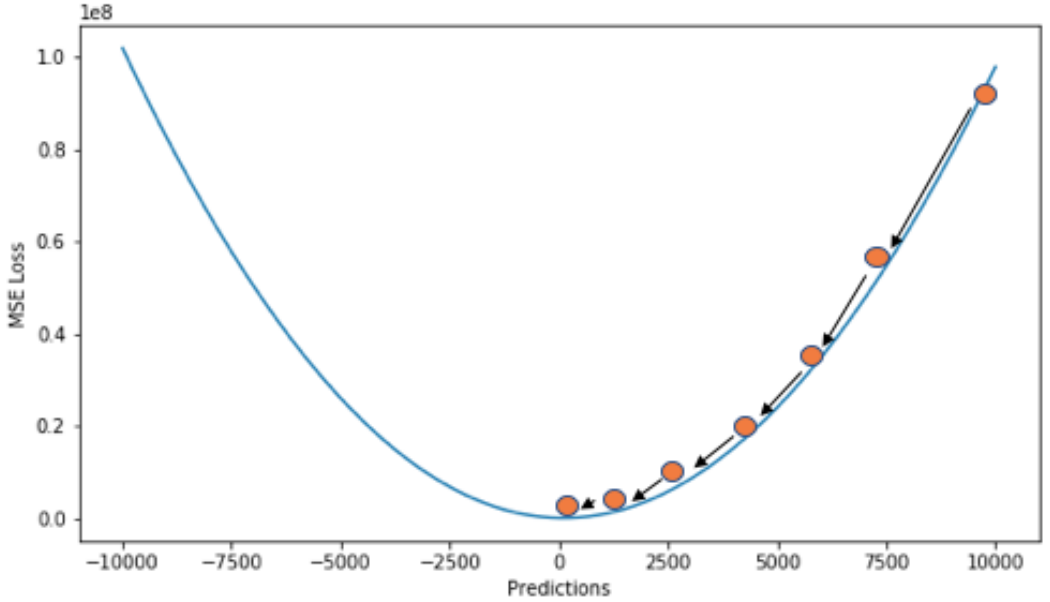




Learning Rate

- This size of steps taken to reach the minimum or bottom is called Learning Rate (***hyperparameter***). We can cover more area with larger steps/higher learning rate but are at the risk of overshooting the minima. On the other hand, small steps/smaller learning rates will consume a lot of time to reach the lowest point.
- **Example: Set a learning rate of (0.03), (1) , (0.1) on the slider.**
- https://developers.google.com/machine-learning/crash-course/fitter/graph



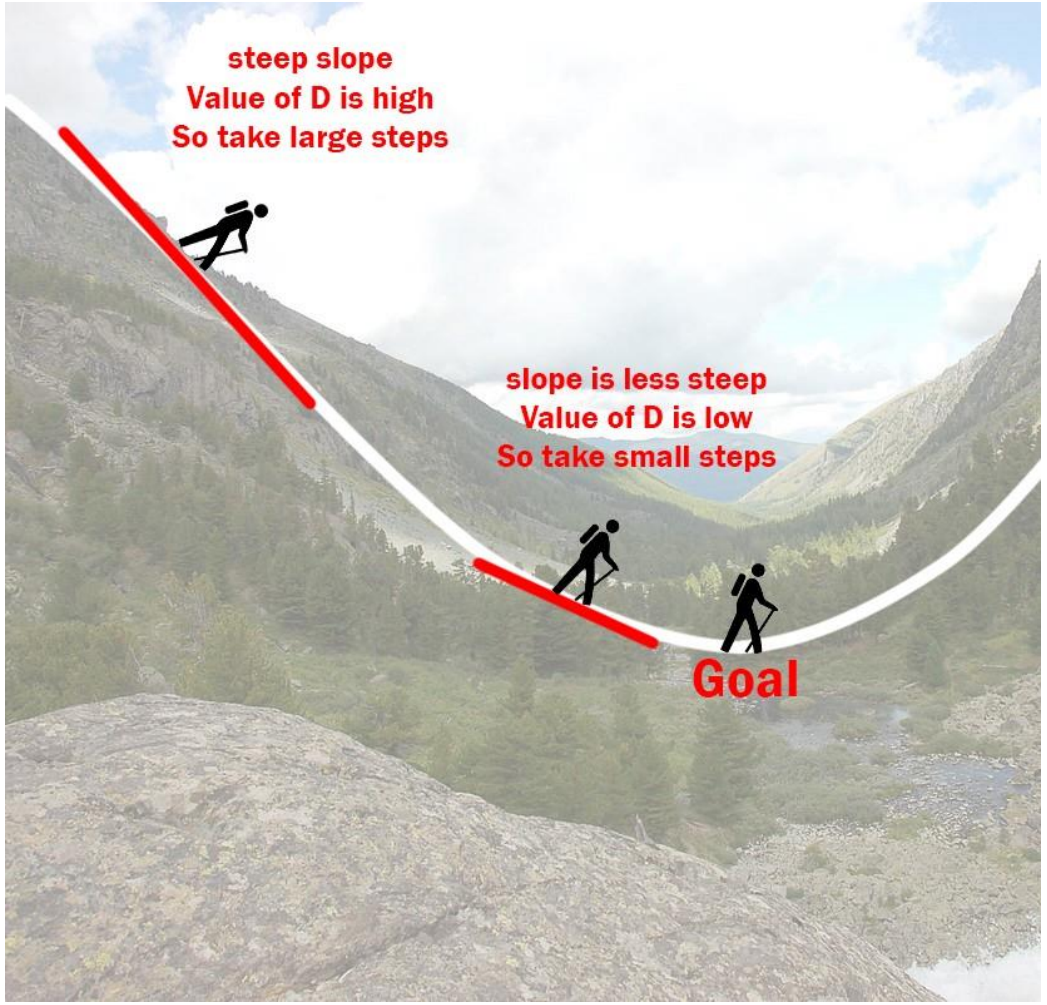


Loss Functions and Gradient

- **MAE** is robust to outlier. However, it has a constant gradient.
- **MSE** is not robust to outlier. However, it has a changing gradient that enables better learning.

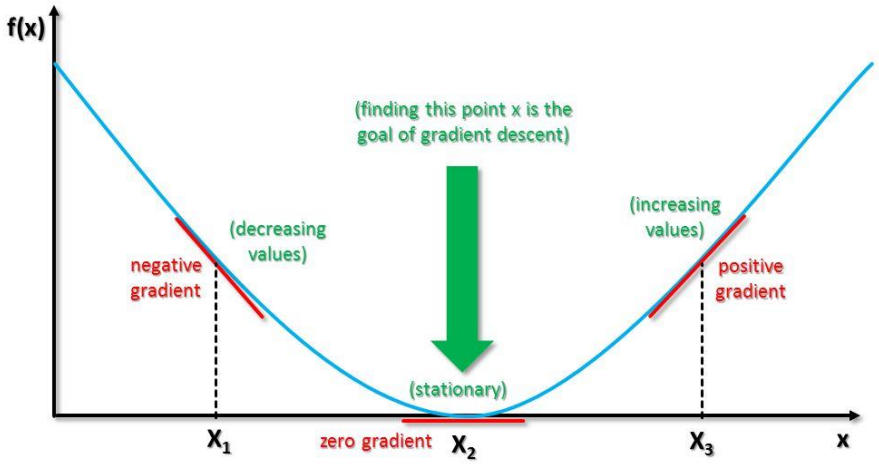
Gradient Descent Algorithm

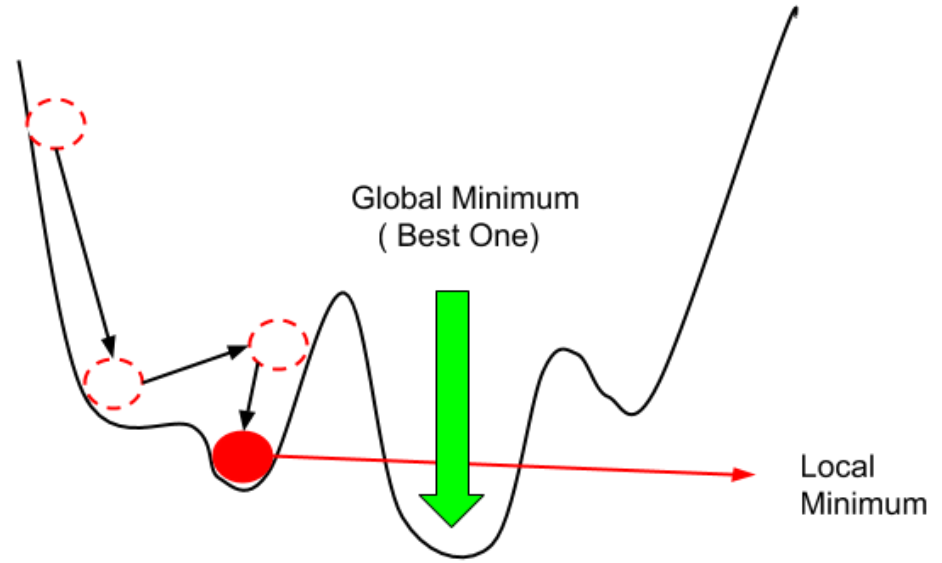
- **Intuition :** A person (blindly) trying to go down a hill when it is foggy. The idea is to make single step at a time, in the direction of the steepest descent.
- **GD** makes the same thing to find the min of a loss function.



Gradient Descent

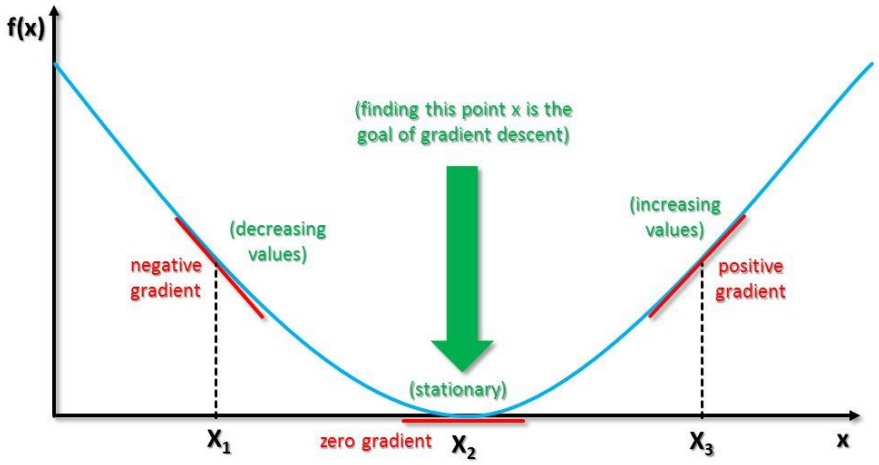
- **Gradient Descent** is a first-order iterative optimization algorithm for finding a **local** minimum of a **differentiable** function.
- The idea is to take repeated steps in the opposite direction of the gradient of the function at the current point, because this is the direction of steepest descent.

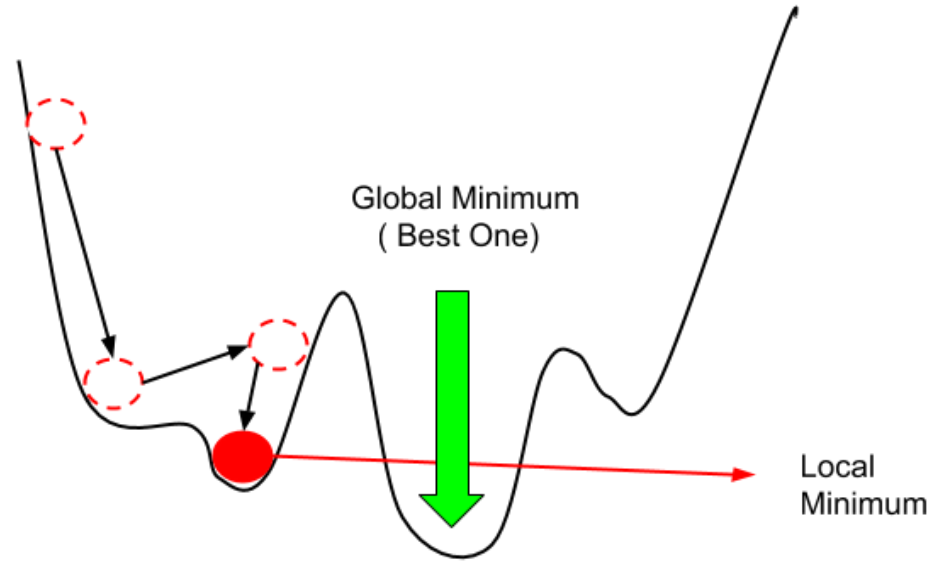


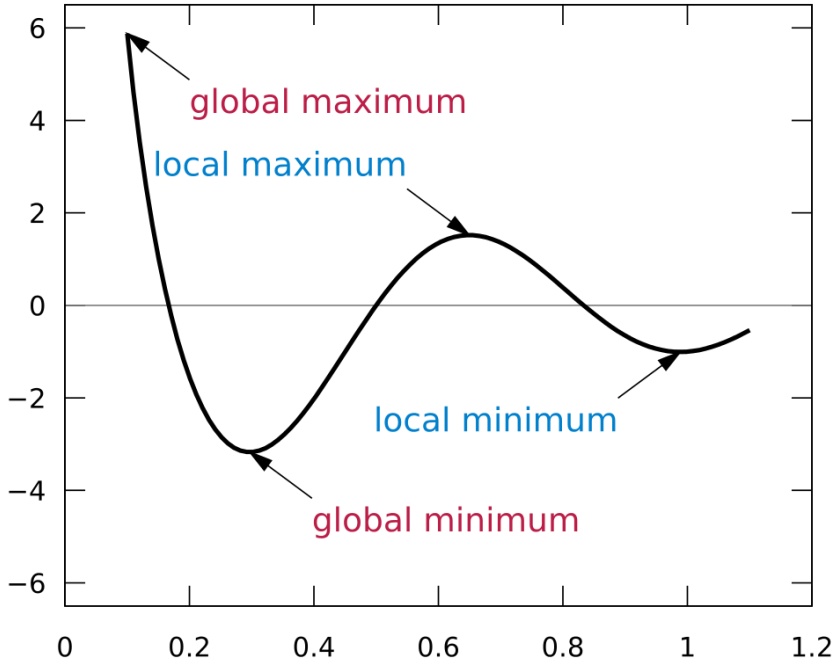


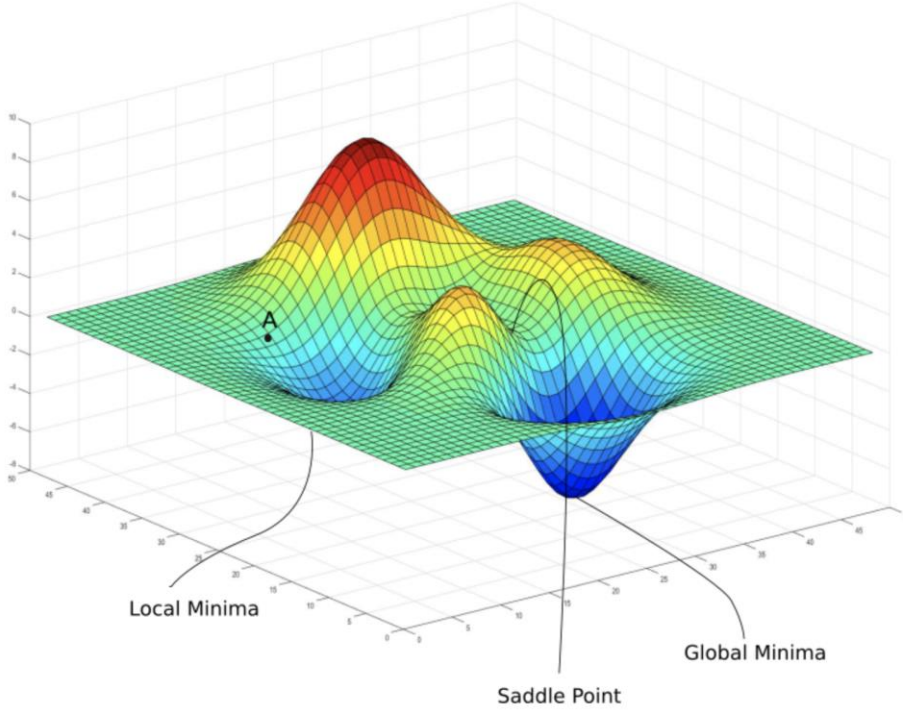
Gradient Descent

- This algorithm and its variants have been proven effective to solve data related problems, especially in the domain of neural networks. It's not the only algorithm or the best but it is seen as the « **hello world** » of machine learning.
- **Gradient descent (GD)** is an optimization algorithm that's used when training a machine learning model. It's based on a **convex** function and tweaks its parameters (coefficients) iteratively to **minimize** a given function as far as possible.



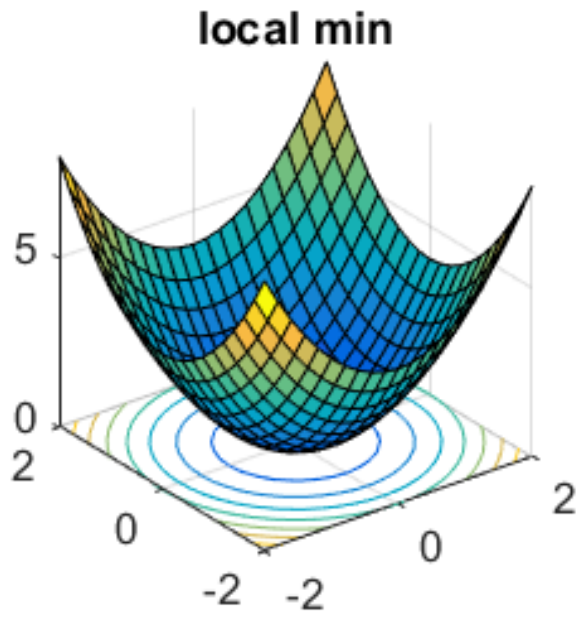


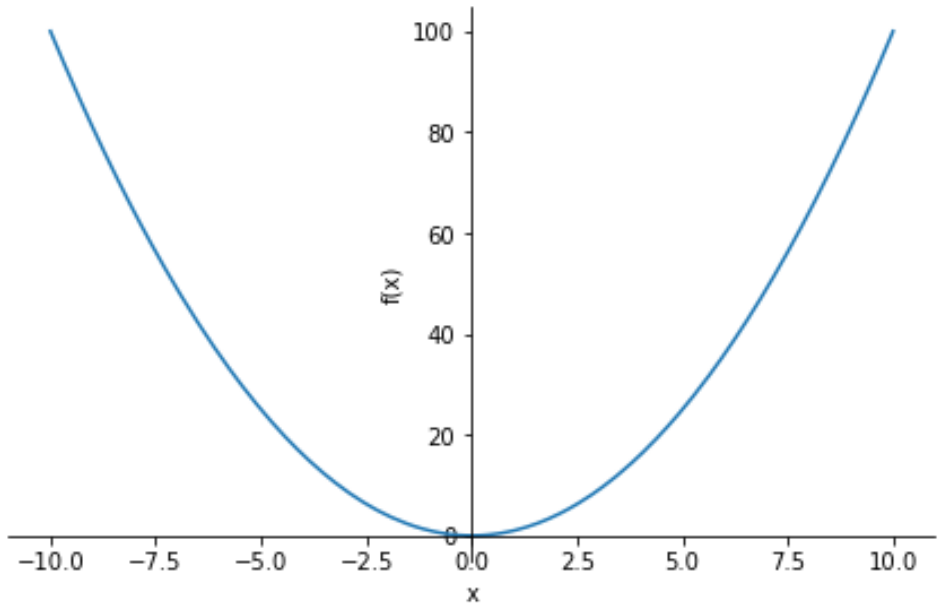




Local vs. Global Minimum

- A local minimum (or optimum) of a function is a point where the function value is smaller than at nearby points, but possibly greater than at a distant point.
- A global minimum (or optimum) is a point where the function value is smaller than at all other feasible points.





Convex Function

Has one local minimum which also is its global minimum.

Linear Regression

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Logistic Regression

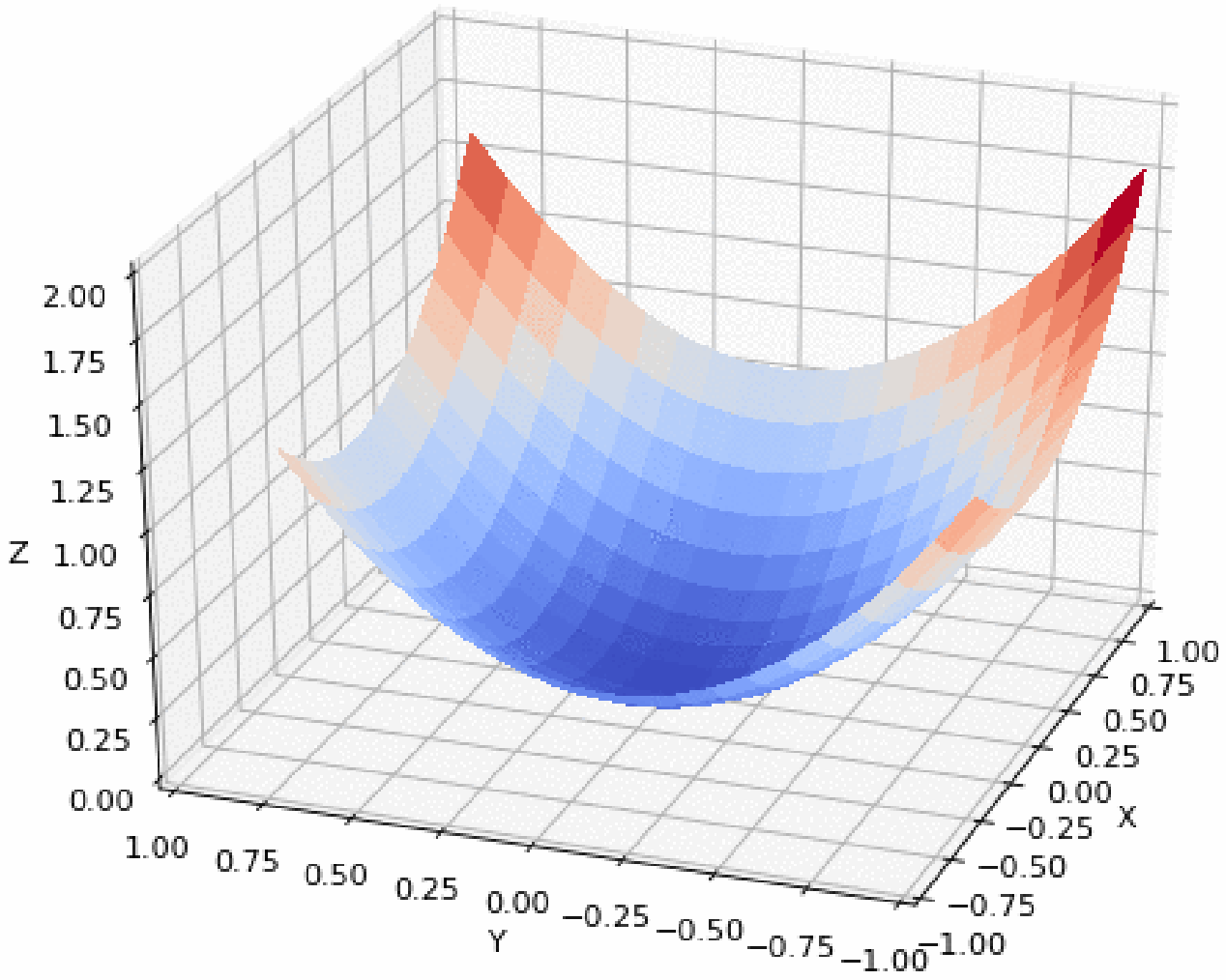
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$J(\theta) = \frac{1}{m}[\sum_{i=1}^{m} -y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

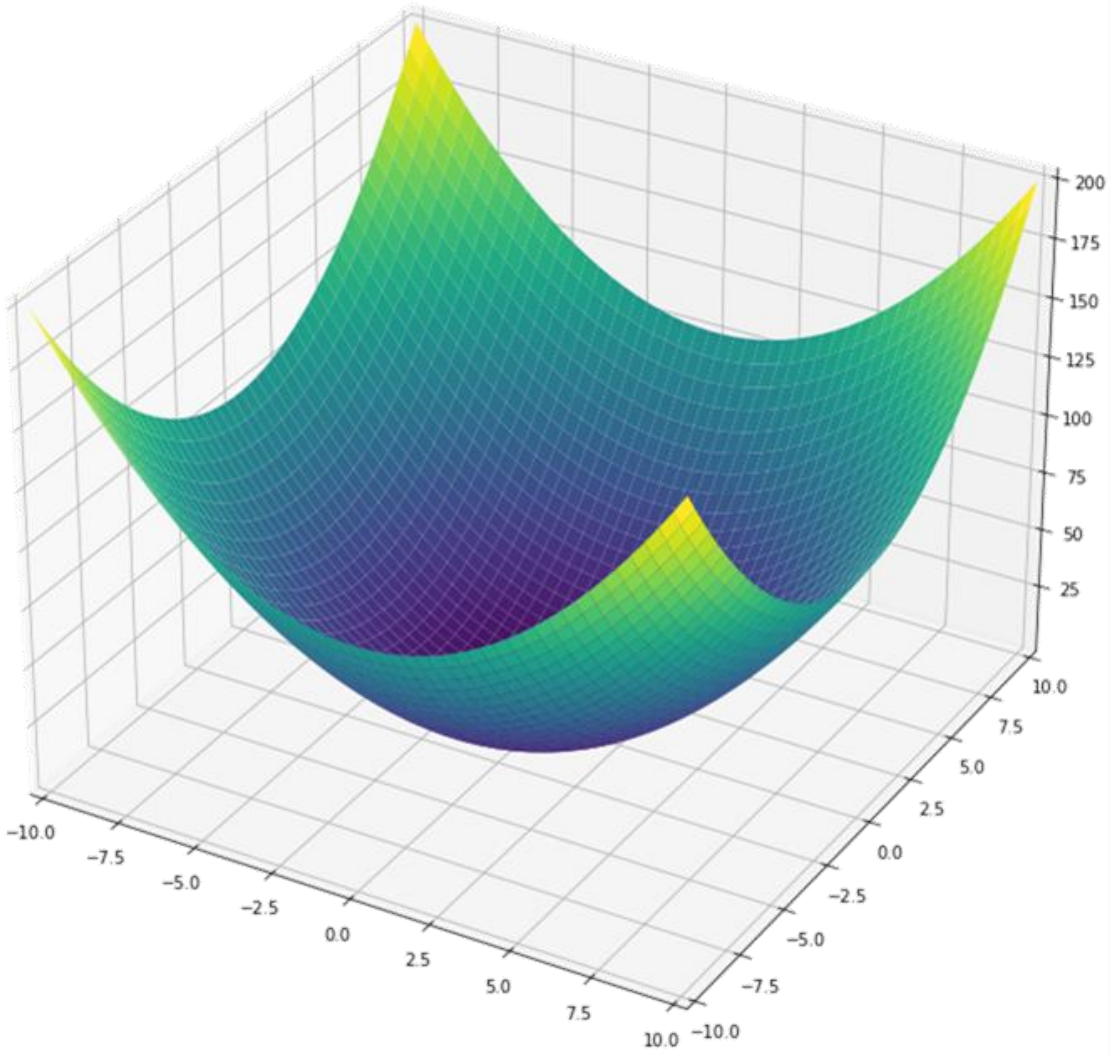
$m = number\ of\ samples$

Convex Problem

We try to choose the formula for the cost function that makes it convex.



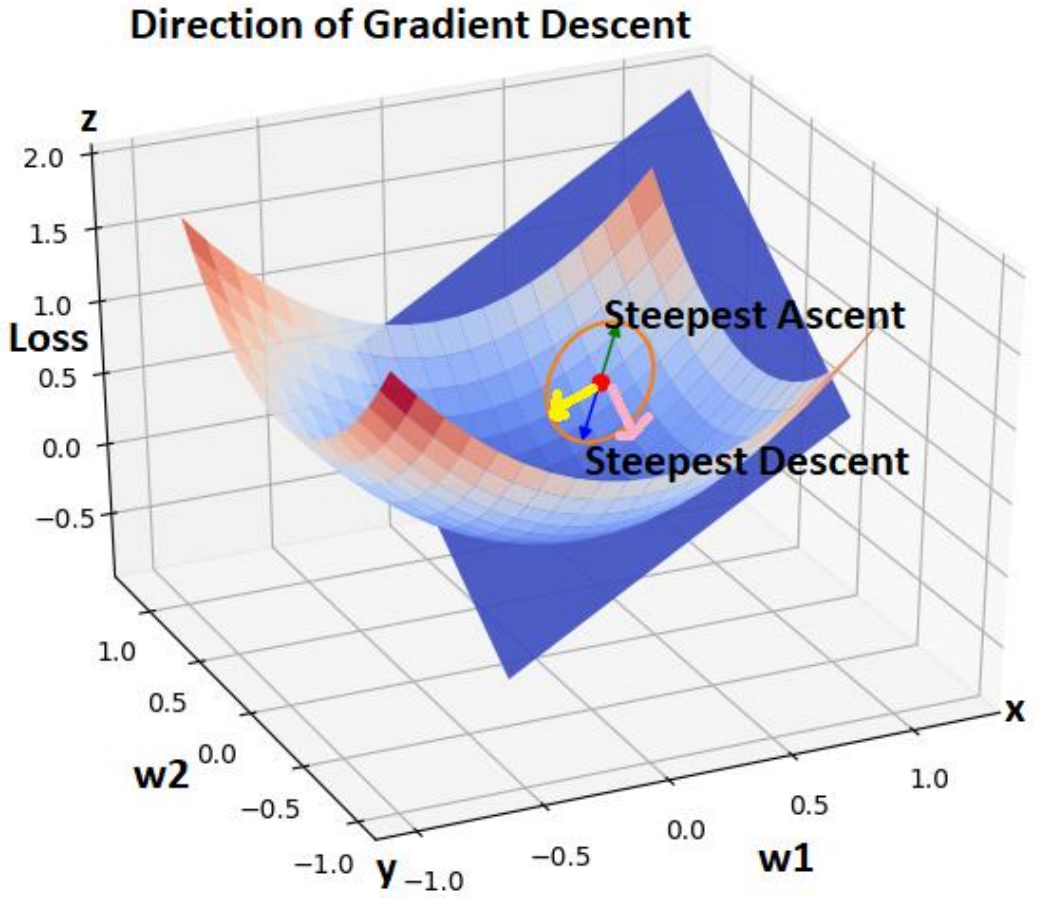
Gradient of Multivariable Function



Gradient of Multivariable Function

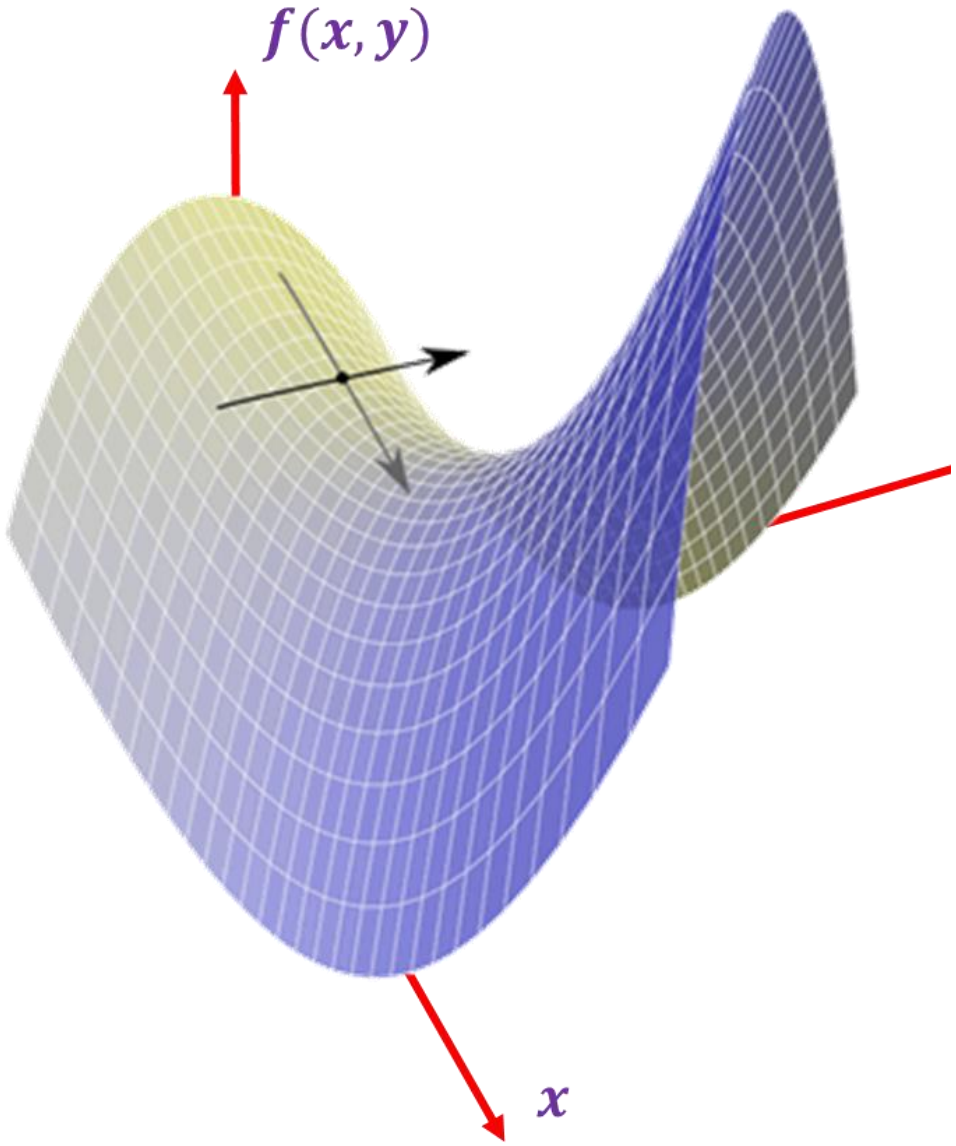
Gradient of Multivariable Function

- **Gradient Vector:**
- $\nabla\theta = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \end{bmatrix}$



Gradient of Multivariable Function

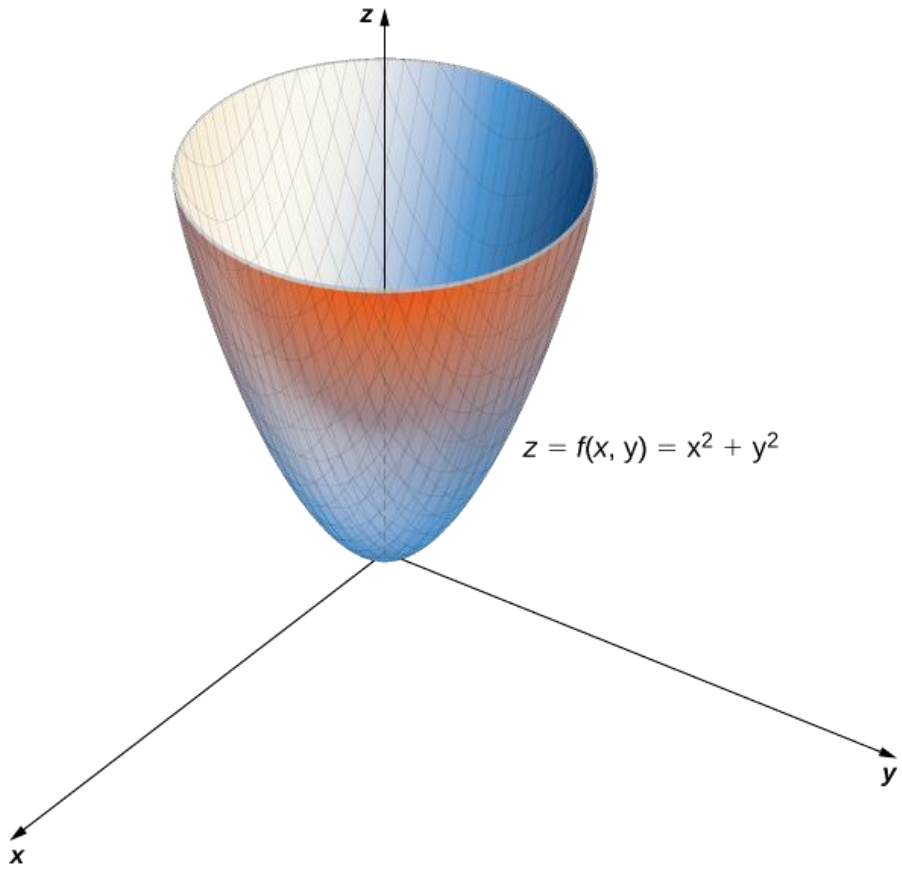
- Partial Derivative is the rate of change of a multi-variable function when all, but one variable is held fixed.
- **Example:**
  - a function for a surface that depends on two variables x and y.
  - When we find the slope in the x direction (while keeping y fixed) we have found a partial derivative.



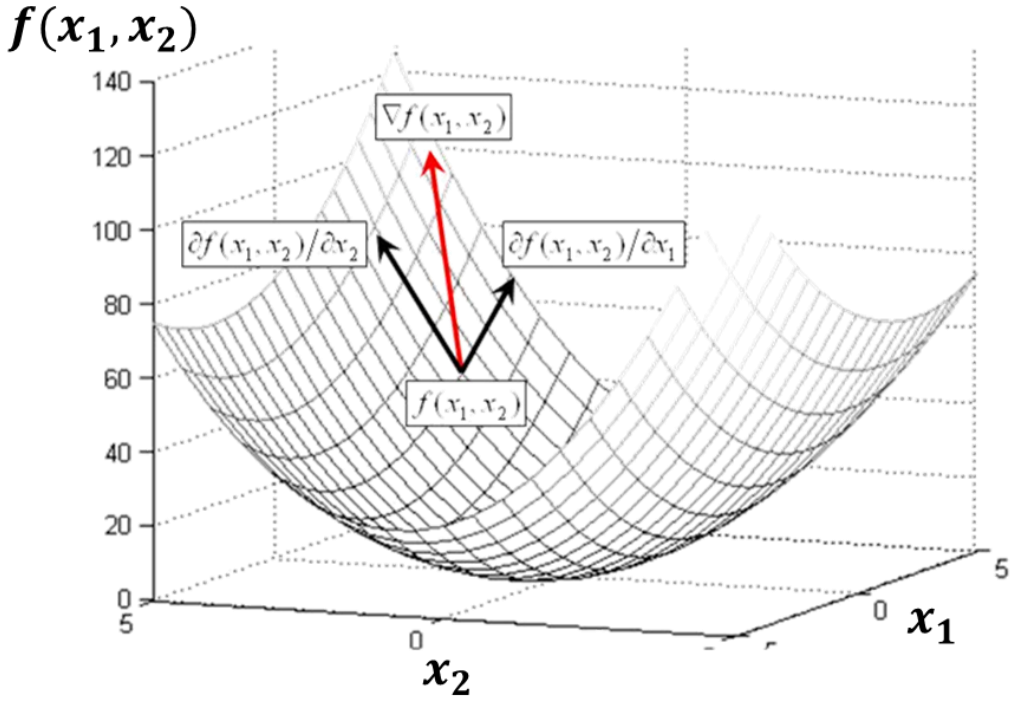
Gradient of Multivariable Function

- $\frac{\partial f}{\partial x} = 2x \; ; \frac{\partial f}{\partial y} = 2y$
- $\boldsymbol{Gradient} = \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$

- Sometimes called Jacobian vector (It is a special case from Jacobian matrix)
- Gradient is a vector



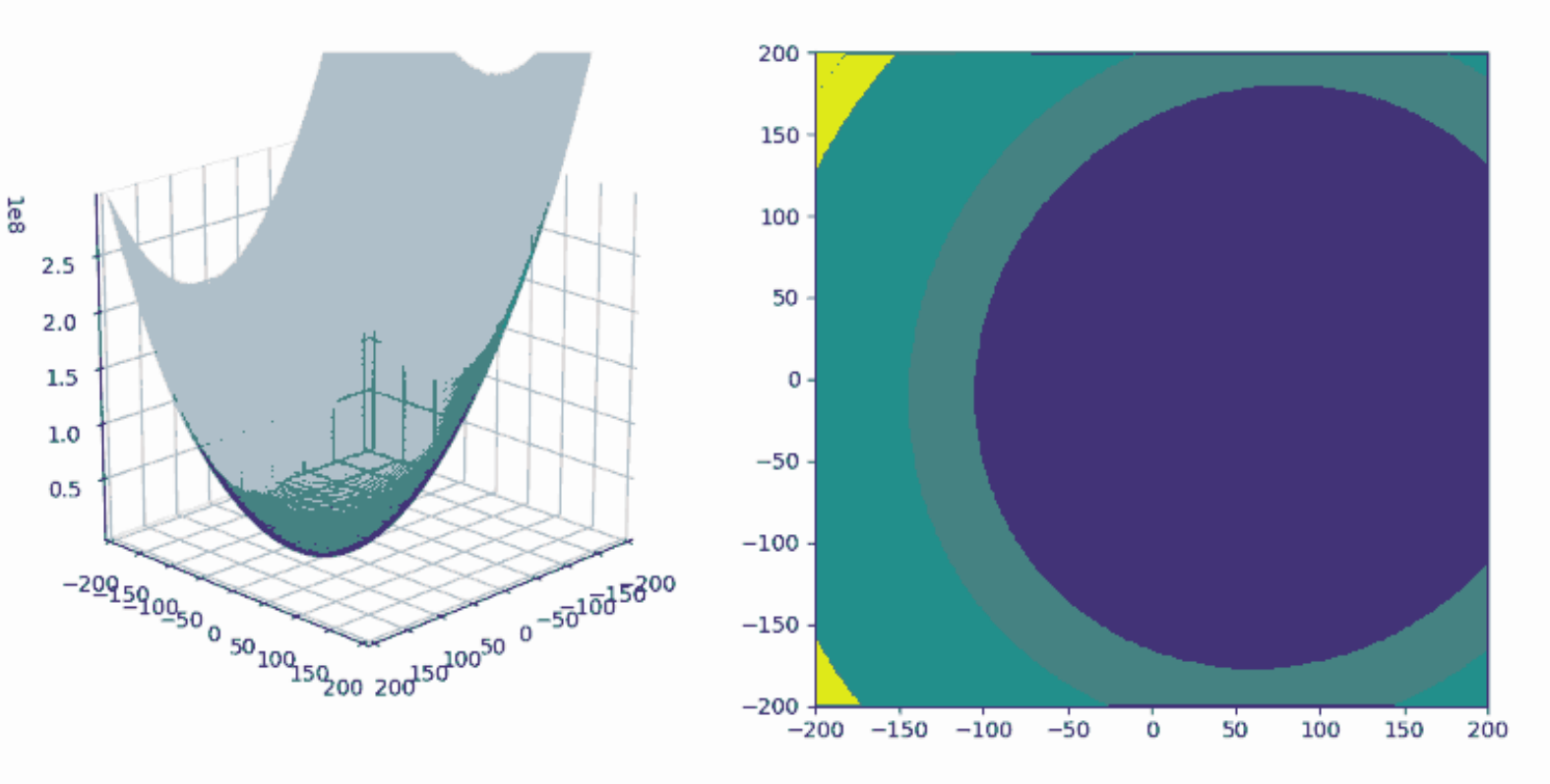
Gradient vector points in direction of steepest ascent of function



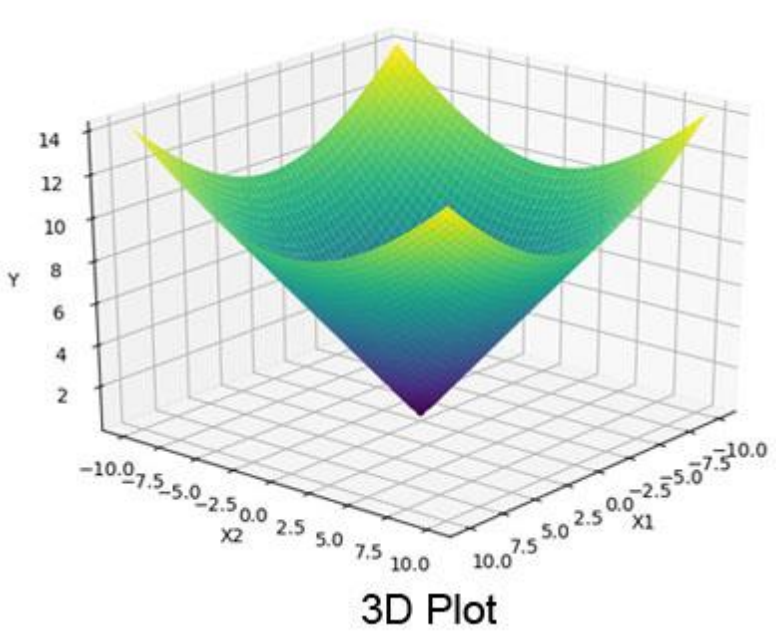
$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}.$$

Gradient of Multivariable Function

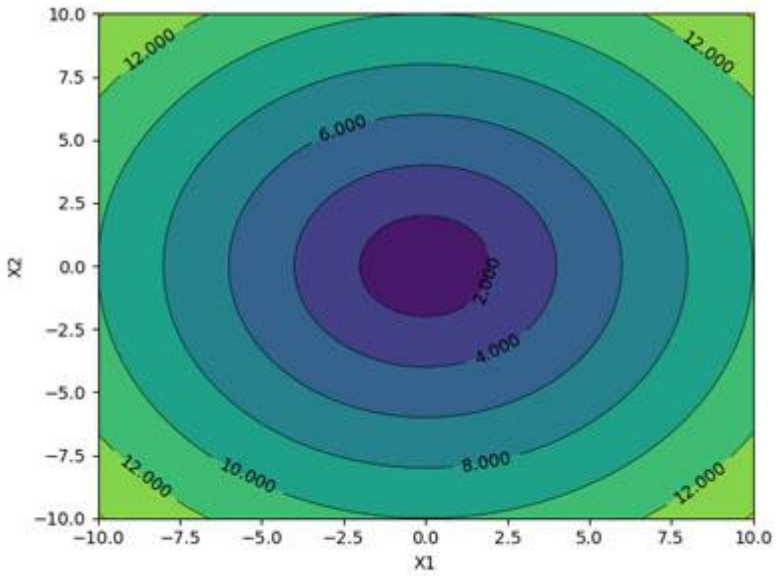
- Gradient is a vector points to the opposite of steepest descent direction.
- Can be generalized to **n** dimensions
- $\boldsymbol{p}$ is any point at where the gradient is calculated.



Contour Plot

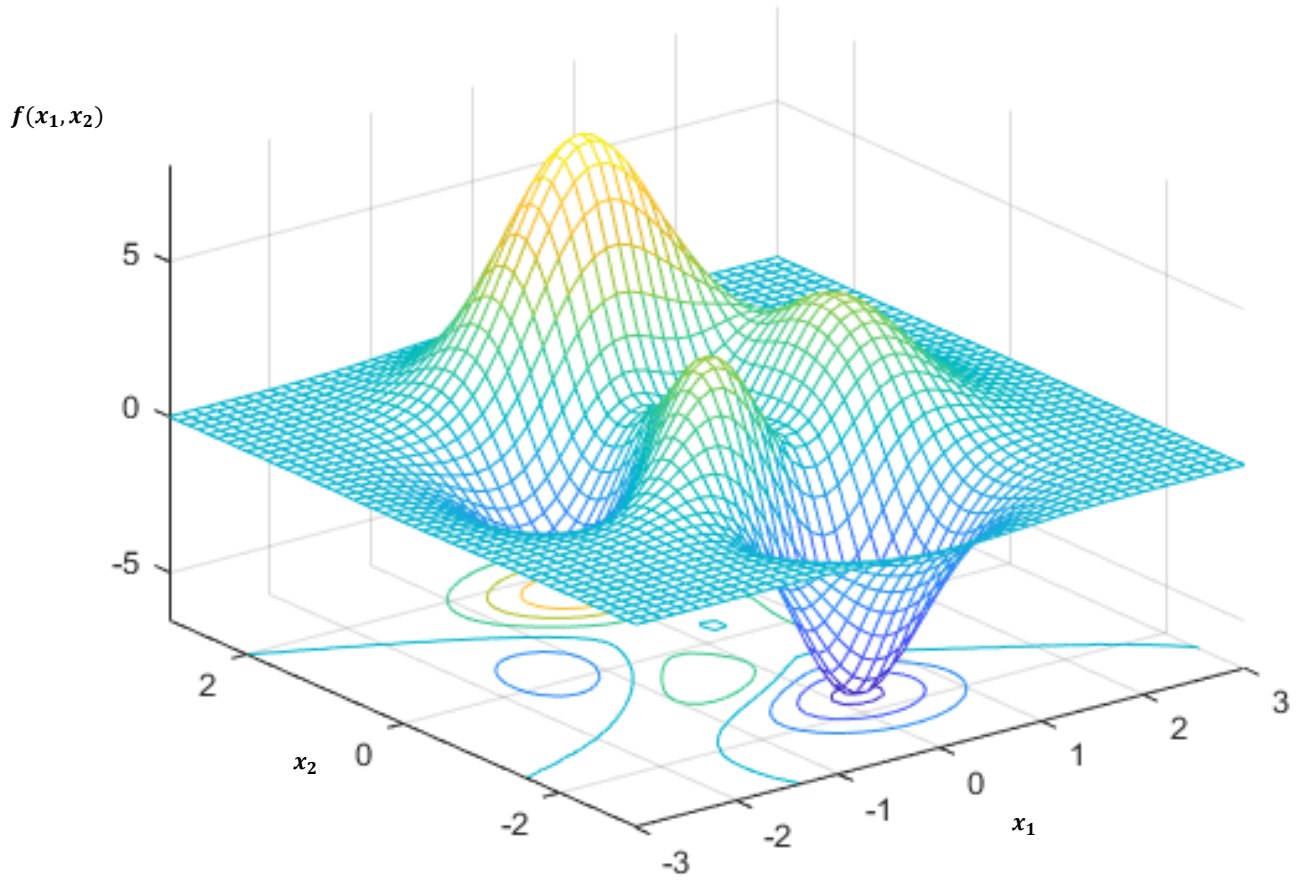


3D Plot

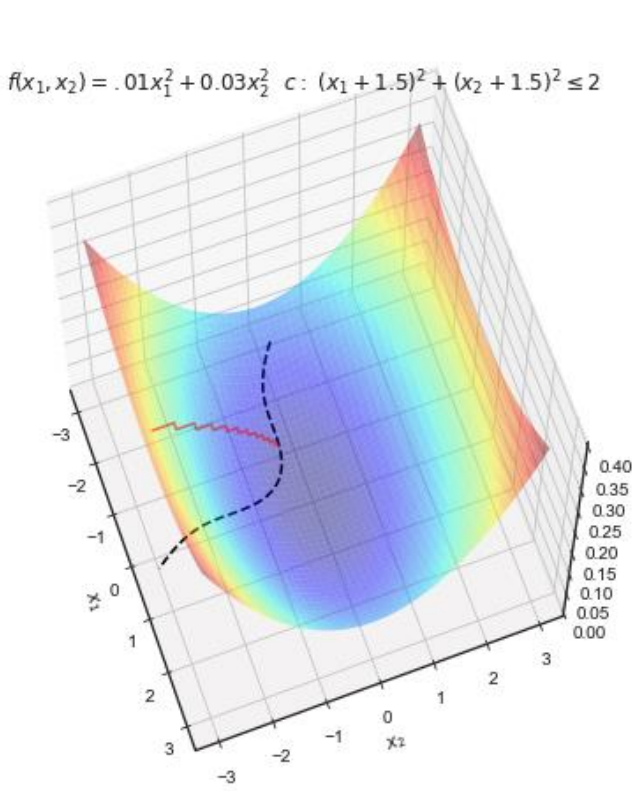


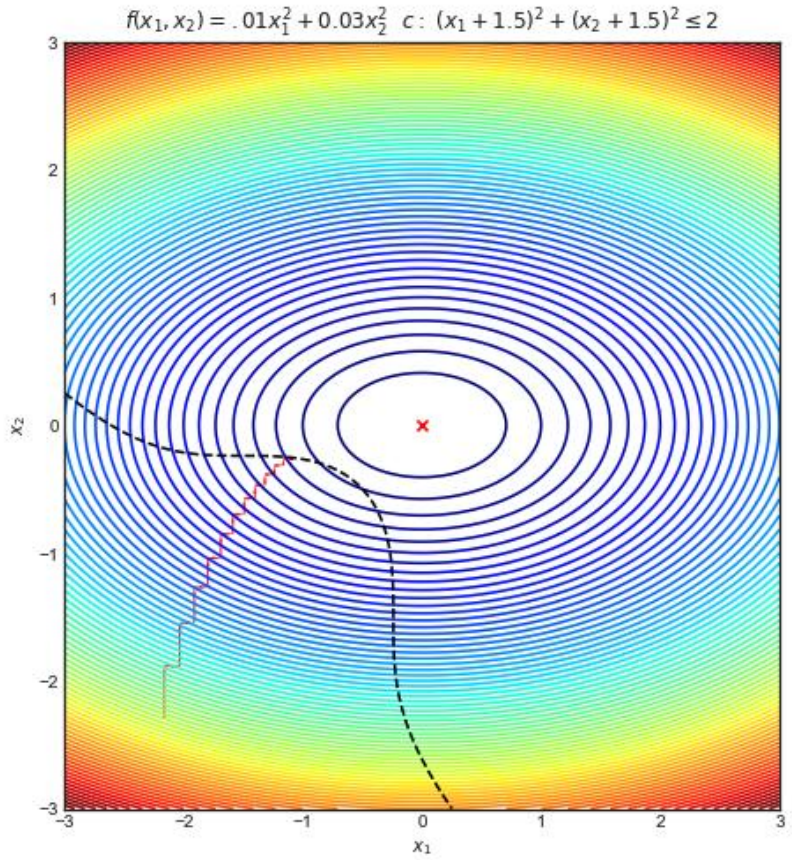
Contour Plot

Contour Plot

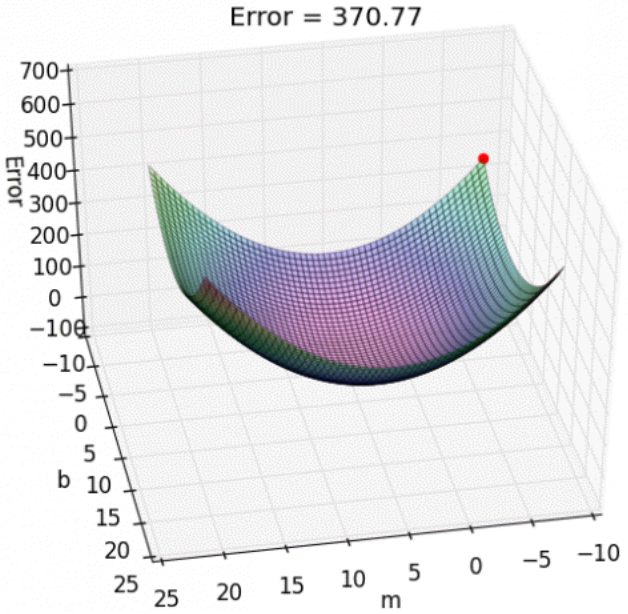


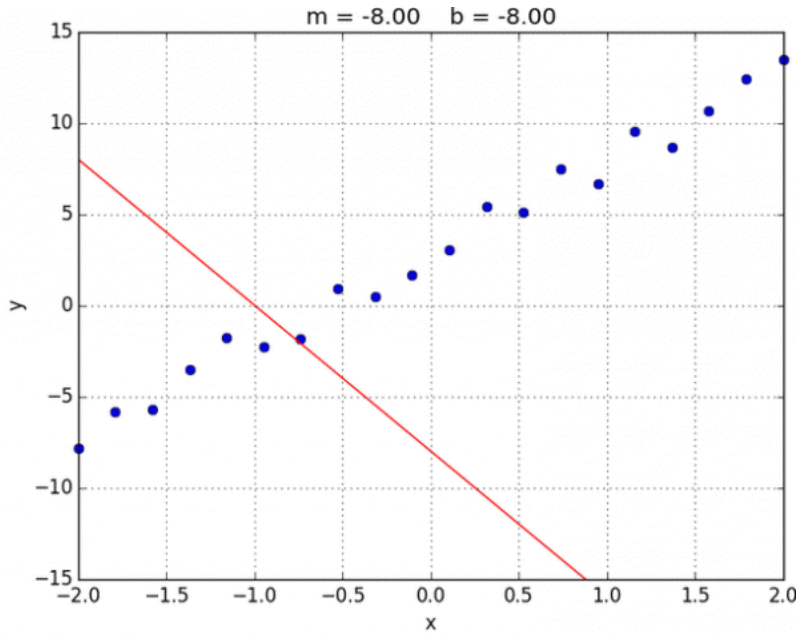
Contour Plot



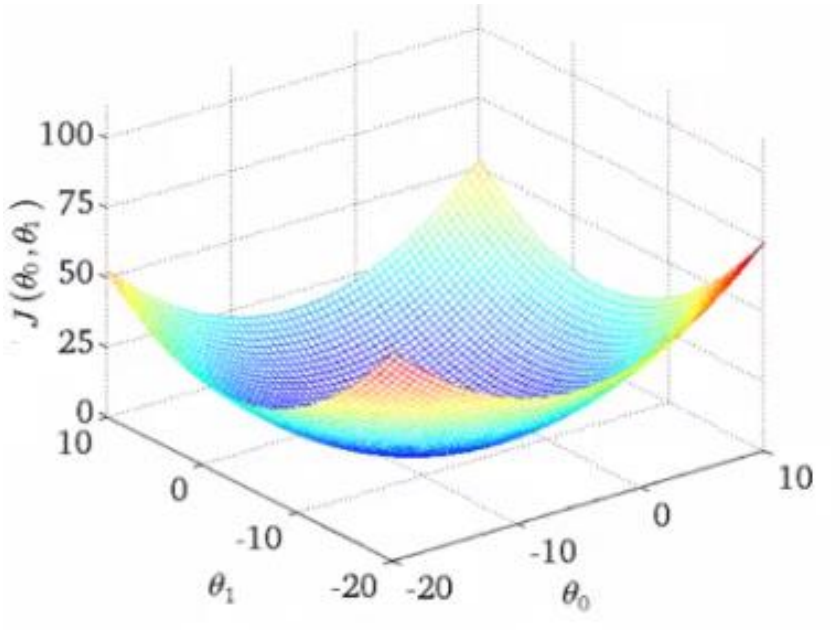


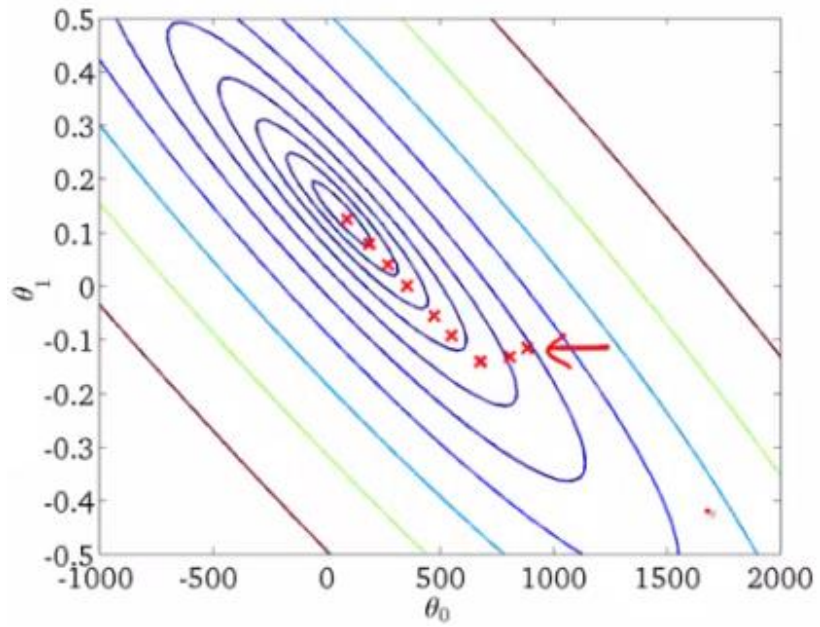
Contour Plot





LR & Loss Function





LR & Loss Function

LR Loss Function

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

GD Applied to LR (Single Variable)

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

GD Applied to LR (Single Variable)

GD Applied to LR (Single Variable)

- **Implementation Steps:**
- ***Step1:*** Initialize parameters **(θ_0 & θ_1)** with random value or simply zero. Also choose the **Learning rate**.
- ***Step2:*** Use (θ_0 & θ_1) to predict the output $h_\theta(x) = \theta_0 + \theta_1 x$.
- ***Step3:*** Calculate Cost function $J(\theta_0, \theta_1)$.
- ***Step4:*** Calculate the gradient.
- ***Step5:*** Update the parameters (simultaneously).
- ***Step6:*** Repeat from 2 to 5 until converge to the minimum or achieve maximum iterations.

GD Applied to LR (Single Variable)

- **Implementation Notes:**
  - Parameters should be updated simultaneously.
  - Learning step will decrease as you become closer to the minimum. Even with fixed learning rate.
  - Do not use very large learning rate in order not to overshoot.
  - Do not use very small learning rate in order not to go very slowly.

Resources

- https://www.coursera.org/learn/machine-learning
- https://machinelearningmastery.com/analytical-vs-numerical-solutions-in-machine-learning/
- https://www.youtube.com/watch?v=e6kf6DDQVYA&ab_channel=TreeSoftMatterTheory
- https://en.wikipedia.org/wiki/Mathematical_optimization
- https://builtin.com/data-science/gradient-descent
- https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
- https://math.stackexchange.com/questions/2202545/why-using-squared-distances-in-the-cost-function-linear-regression
- https://towardsdatascience.com/optimization-loss-function-under-the-hood-part-ii-d20a239cde11
- https://www.mathsisfun.com/gradient.html
- https://en.wikipedia.org/wiki/Derivative
- https://www.mathsisfun.com/calculus/derivatives-introduction.html
- https://math.libretexts.org/Bookshelves/Calculus/Map%3A_Calculus__Early_Transcendentals_(Stewart)/14%3A_Partial_Derivatives/14.01%3A_Functions_of_Several_Variables
- https://slideplayer.com/slide/4753135/
- https://en.wikipedia.org/wiki/Gradient
- https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/partial-derivative-and-gradient-articles/a/the-gradient
- https://la.mathworks.com/help/matlab/ref/meshc.html
- http://www.adeveloperdiary.com/data-science/how-to-visualize-gradient-descent-using-contour-plot-in-python/
- https://rpubs.com/mgswiss15/M6C_7Multivariate
- https://stats.stackexchange.com/questions/354046/coordinate-descent-with-constraints
- https://www.mathworks.com/help/optim/ug/local-vs-global-optima.html#:~:text=A%20local%20minimum%20of%20a,at%20all%20other%20feasible%20points.
- https://en.wikipedia.org/wiki/Maxima_and_minima
- https://wngaw.github.io/linear-regression/
- http://www.cheerml.com/saddle-points
- https://towardsdatascience.com/understand-convexity-in-optimization-db87653bf920
- https://towardsdatascience.com/understand-convexity-in-optimization-db87653bf920
- https://www.sciencedirect.com/topics/engineering/convex-function
- https://www.math24.net/convex-functions#example2
- https://tutorial.math.lamar.edu/Classes/CalcI/NewtonsMethod.aspx
- https://en.wikipedia.org/wiki/Newton's_method
- https://tutorial.math.lamar.edu/Classes/CalcI/NewtonsMethod.aspx

Resources

- https://realpython.com/linear-regression-in-python/
- https://towardsdatascience.com/linear-regression-using-python-b136c91bf0a2
- https://towardsdatascience.com/why-norms-matters-machine-learning-3f08120af429
- https://towardsdatascience.com/why-norms-matters-machine-learning-3f08120af429
- https://machinelearningmastery.com/vector-norms-machine-learning/
- https://medium.com/linear-algebra/part-18-norms-30a8b3739bb
- https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0
- Andrew Ng, Machine Learning, Stanford University, Coursera
- https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0
- https://medium.com/data-science-365/linear-regression-with-gradient-descent-895bb7d18d52
- https://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html
- https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220
- https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220
- https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/
- https://builtin.com/data-science/gradient-descent
- https://www.mltut.com/stochastic-gradient-descent-a-super-easy-complete-guide/
- https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931
- https://kaigangi72.medium.com/stochastic-gradient-descent-demystified-part-1-8e4b897079b7
- https://medium.datadriveninvestor.com/gradient-descent-algorithm-b4c5afb4eb98
- https://medium.com/mindorks/an-introduction-to-gradient-descent-7b0c6d9e49f6
- https://medium.com/@venkatavinay222/at-the-end-machine-learning-is-all-about-optimization-ft-gradient-descent-e1588b7d95d2
- “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow” by Aurélien Géron
- https://laptrinhx.com/feature-scaling-why-and-how-3308094292/
- https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3