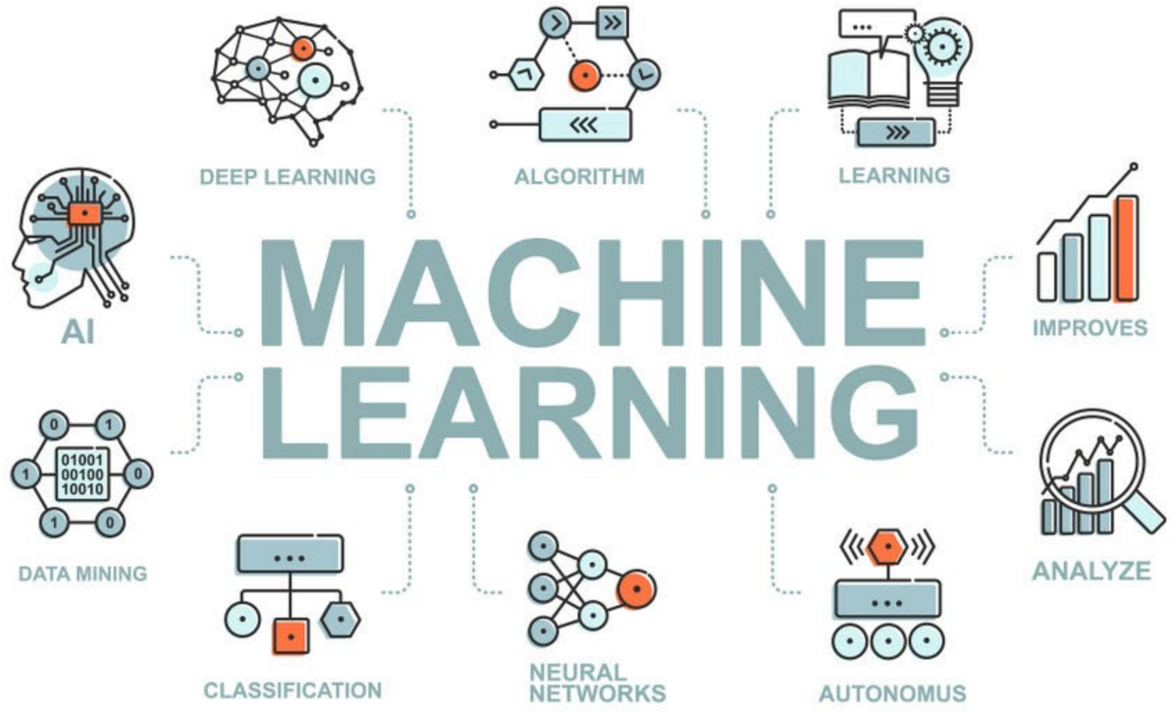
From statistics to machine learning. Procedures for training, optimizing, and validating training tools in ML

Machine Learning

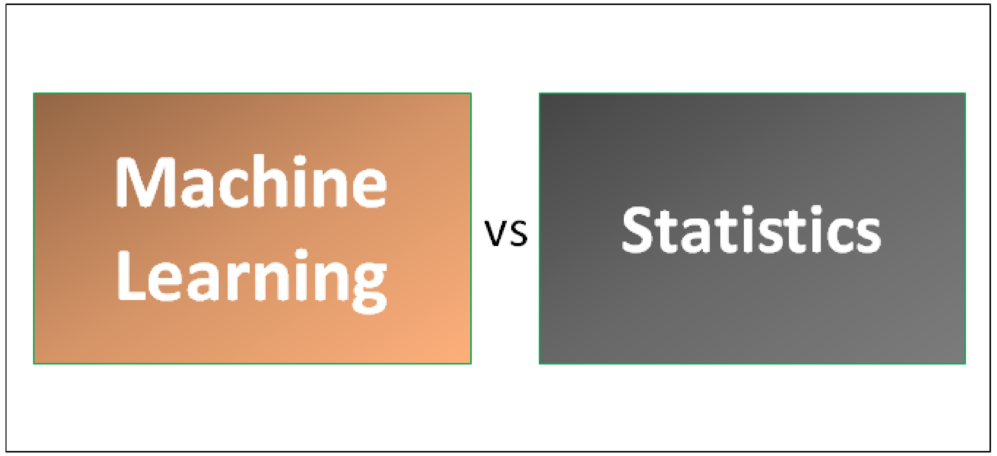


Toni Monleón-Getino, amonleong@ub.edu
04/25/2023

Index

From statistics to machine learning

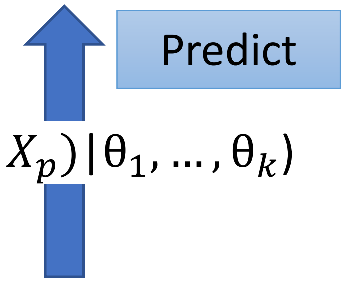
- The major difference between machine learning and statistics is their purpose.
- Machine learning models are designed to make the most accurate predictions possible.
- Statistical models are designed for inference about the relationships between variables.



https://towardsdatascience.com/the-actual-difference-between-statistics-and-machine-learning-64b49f07ea3

Data Matrix in statistics

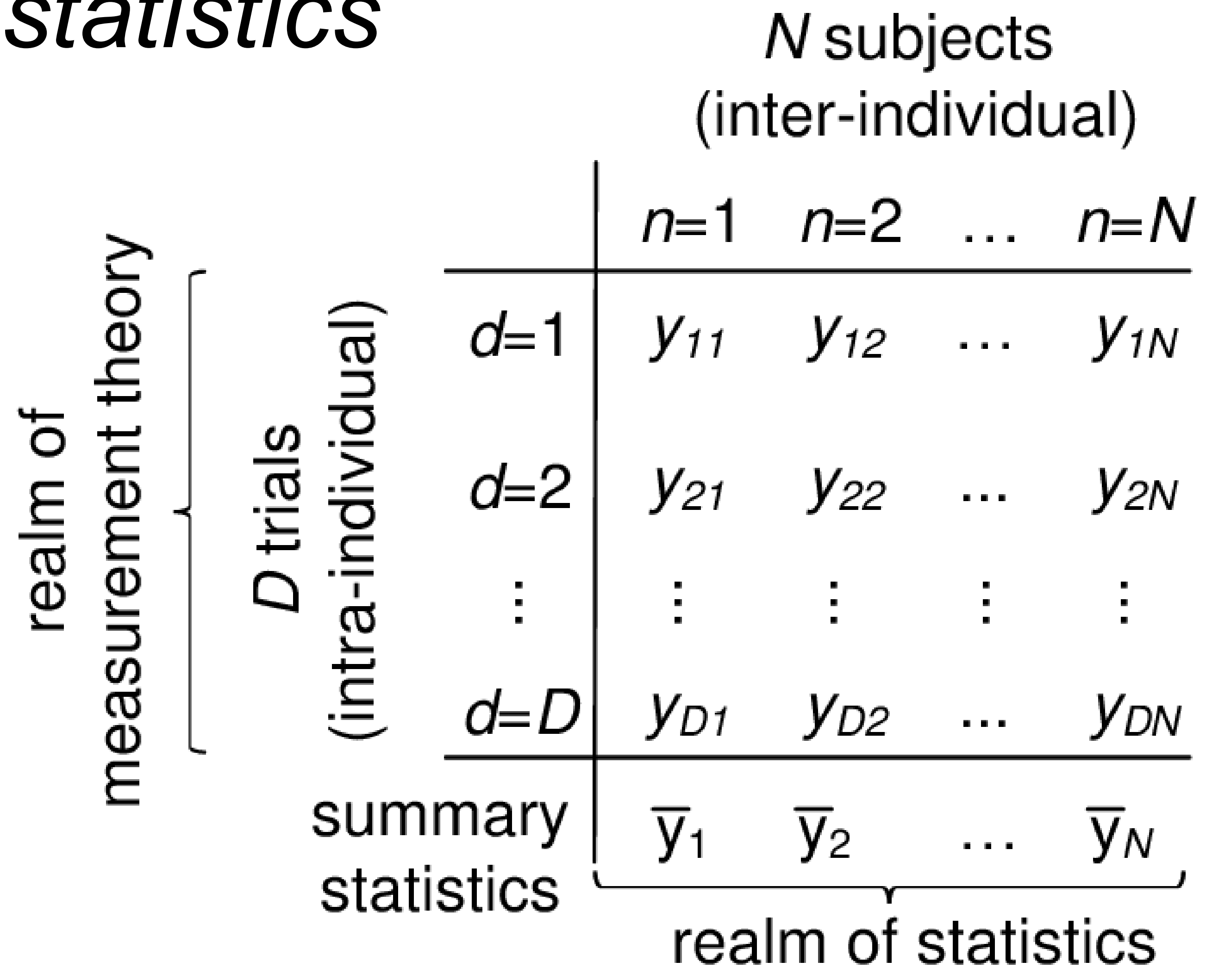
Y = Dependent variable



$f(X_1, \dots, X_p) \mid \theta_1, \dots, \theta_k)$

X = Independent variable

Y =



N subjects (inter-individual)

realm of measurement theory

D trials (intra-individual)

	n=1	*n*=2	…	*n*=*N*
d=1	y_{11}	y_{12}	…	y_{1N}
d=2	y_{21}	y_{22}	…	y_{2N}
⋮	⋮	⋮	⋮	⋮
d=*D*	y_{D1}	y_{D2}	…	y_{DN}
summary statistics	$\bar{y}_1$	$\bar{y}_2$	…	$\bar{y}_N$

realm of statistics

From statistics to machine learning

- What is Statistical Learning?
- Old wine on new bottles? Is it not just plain statistical inference and regression theory?
- New(ish) field on how to use statistics to make the computer ’learn’?
- A merger of classical disciplines in statistics with methodology from areas known as machine learning, pattern recognition and artificial neural networks
- Major purpose: Prediction -- as opposed to .... truth!?
- Major point of view: Function approximation, solution of a mathematically formulated estimation problem as opposed to algorithms.

From statistics to machine learning

- The areas mentioned above, machine learning, pattern recognition and artificial neural networks have lived their lifes mostly in the non-statistical literature.
- The theories for learning -- what would be called estimation in the statistical jargon -- have been developed mostly by computer scientists, engineers, physicists and others.
- The quite typical approach of statistics to the problem of inductive inference -- the learning from data -- is to formulate the problem as a mathematical problem.
- Then learning means that we want to find one mathematical model for data generation among a set of candidate models, and the one found is almost always found as a solution to an estimation equation or an optimization problems.
- A typical alternative approach to learning is algorithmic, and a lot of the algorithms are thought up with the behavior of human beings in mind. Hence the term ``learning'' -- and hence the widespread use of terminology such as ``training data'' and ``supervised learning'' in machine learning.

Simple Regression model in the statistical point of view

Statistical modeling: Regression models

Set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome variable') and one or more independent variables (often called 'predictors', 'covariates' or 'features').

- Ordinary Least Squares (OLS) regression is a frequentist approach to modeling
- The idea is minimize a loss function (L^2), based only on training data
- Estimation of model parameters:

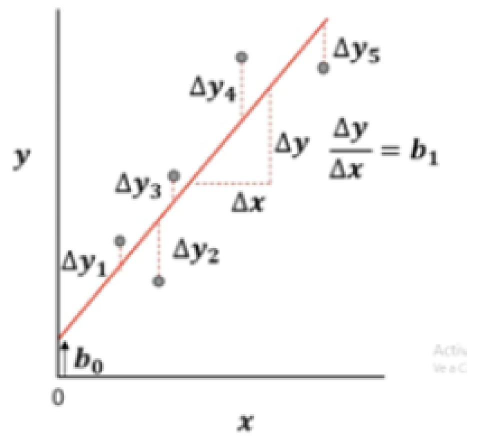
Loss function: ε, L^2

Model: $y = \beta_0 + \beta_1 x$
where y is the outcome and x the predictor, $\beta_0 and \beta_1$ are the model coefficients

Regression model (Supervised problema)

Model parameters set to minimize mismatch at with training data locations.
Model: $y = \beta_0 + \beta x$

- Objective: Find β_0, β_0, fit a linear function, to:
    - Minimize Δy_i over all the data with the L^2 Norm
    - Δy_i is the prediction error:
      $\Delta y_i = y_i - y_{est}$



- Minimize cost function:

$$\sum_{i=1}^{n} (\Delta y_i)^2 = \sum_{i=1}^{n} (y_i - (\beta_o + \beta_1 x))^2$$

Loss function: $\varepsilon = L^2$

Regression model (Supervised problem)

Simple Linear regression fits the function:

$y = \beta_0 + \beta_1 x_1$

or $y = \alpha + \beta x_1$ (in old times)

where x_1 is the predictor feature, y the response feature and β_0, β_1 the model parameters

Under the constraint:

$$RSS = \sum_{i=1}^{n} (y_i - (\beta_o + \beta_j x_{1i})^2$$

minimize the residual sum of squares (RSS) over the training data

The models is composed by:

Fixed numbers, x_i and Random variables: $Y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$ or:

$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$

with $\epsilon_i \sim N(0, \sigma^2)$ independent and identically distributed (i.i.d.)

So: y_i-values are outcomes of the random variable Y_i, but x_i-values are constants.

Regression model (Supervised problem)

Simple linear regression: Have n pairs $(x_1, y_1), \ldots, (x_n, y_n)$ and want "best"fitting line.
Estimation (OLS):

$$S(\beta_0, \beta_1) = \sum_i (y_i - (\beta_0 + \beta_1 x_i))^2$$

Loss function: $\varepsilon = L^2$

Minimize S over β, β_1 to get

$$\beta_0 = \bar{y} - b\bar{x} = intercept$$

$$\beta_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} = slope$$

Regression model (Supervised problema)

Models the distribution of a continuous-type/quantitative response variable Y_i of subject i in relation to one or more subject specific explanatory variables $X_{i1}, \ldots, X_{ip}$ as follows (additive model):

$$Y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip} + \epsilon_i .$$

- Intercept: β_0
- Regression coefficients: $\beta_1, \ldots, \beta_p$
- Error term ϵ_i has mean zero, it captures the residual variability

A typical aim is to study changes of the mean of Y_i under changes of $X_{i1}, \ldots, X_{ip}$.

Regression model (Supervised problema)

SLR in matrix form

Convert the SLR in a matrix form to generalize to Multiple regression
$\mathbf{y} \in R^n$ = vector of measurements

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

the "**X**-matrix"
min $\sum(y_i - (\beta_0 + \beta_1 x_i))^2$ is equivalent to finding

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

Point estimate as a projection

to mininmize the cost function $||\mathbf{y} - \mathbf{X}\beta||^2$
Number notation: $\mathbf{y} = \mathbf{X}\beta + \mathbf{e}$

Regression model (Supervised problema)

Multiple regression

The model can be written in matricial form:

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{e}$$

using,

$$X = \begin{pmatrix} 1 & X_{11} & X_{12} & \dots & X_{1p} \\ 1 & X_{21} & X_{22} & \dots & X_{2p} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & X_{n1} & X_{n2} & \dots & X_{np} \end{pmatrix} \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \cdots \\ \beta_p \end{pmatrix} \quad \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \cdots \\ \epsilon_n \end{pmatrix} \quad Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \cdots \\ Y_n \end{pmatrix}$$

The coefficients β_i are estimated using the OLS criteria (Ordirary Least Squares):

$$\min \sum_{i=1}^{n} (Y_i - (\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}))^2$$

to obtain: $\hat{\beta} = (X^T X)^{-1} X^T Y$

Regression model (Supervised problema)

The model:

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{e}$$

where $\mathbf{X}$ in an $n \times p$ matrix.

Example: multiple linear regression model: $y_i = \beta_0 + \beta_1 x_i + \beta_2 w_i + \beta_3 x_i$.

More examples: Estimate single intercept, many slopes; Test whether multiple lines are all parallel; ...

Used in all fields of biology (ecology, genetics, ...), medicine, etc, ...
Need to develop point estimates, methods of validation and testing.

Regression model (Supervised problema)

The point estimate is

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

Is named as Normal equations
which is always unbiased (if the mean of the Y-s is correct)

$$E\left[\hat{\beta}\right] = \beta$$

and has variance-covariance matrix

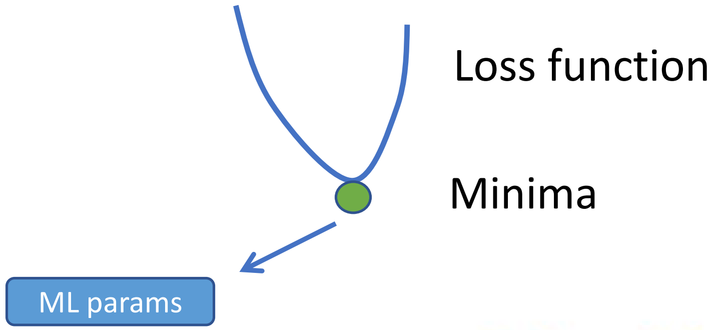
$$V\left[\hat{\beta}\right] = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}.$$

(if the variance assumptions are correct)
and is multivariate Gaussian (if the Y-values are Gaussian).

Machine learning terminology for model building and validation

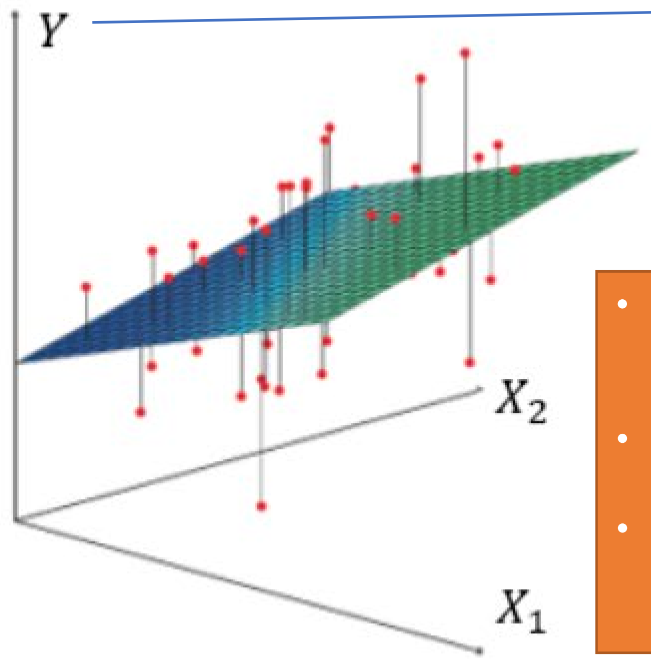
- It seems to be an analogy between statistical modeling and machine learning that we will cover in subsequent chapters in depth.
- However, a quick view has been provided as follows about linear regression:
  - Statistics: linear regression with two independent variables is trying to fit the best plane with the least errors using OLS
  - Machine learning: independent variables have been converted into the square of error terms (squaring ensures the function (Loss function) will become convex, which enhances faster convergence and also ensures a global optimum) and optimized based on coefficient values rather than independent variables.

Statistics vs Machine learning



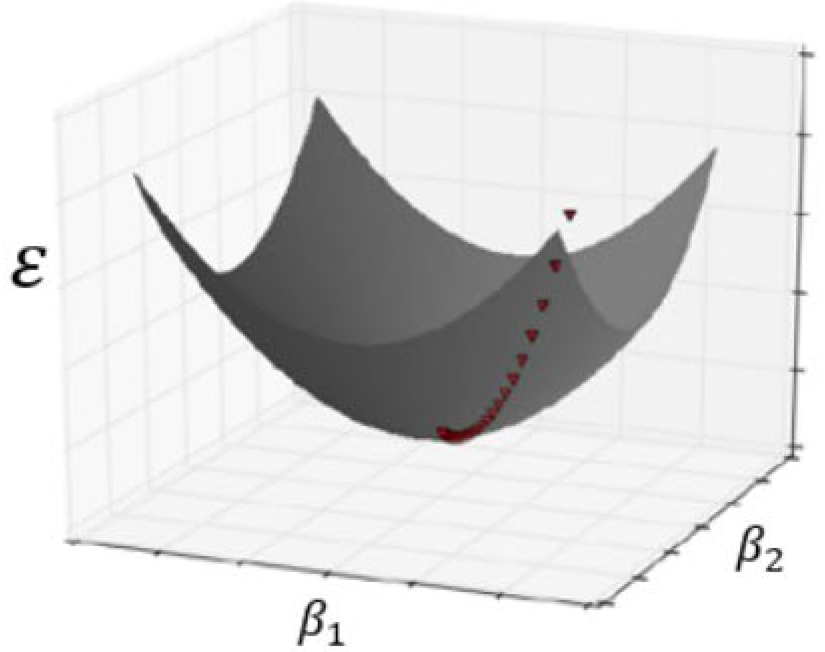
Statistical way

Machine learning way



$$\hat{Y} = \beta_1 * X_1 + \beta_2 * X_2$$

$$\varepsilon = (Y - (\beta_1 * X_1 + \beta_2 * X_2))^2$$



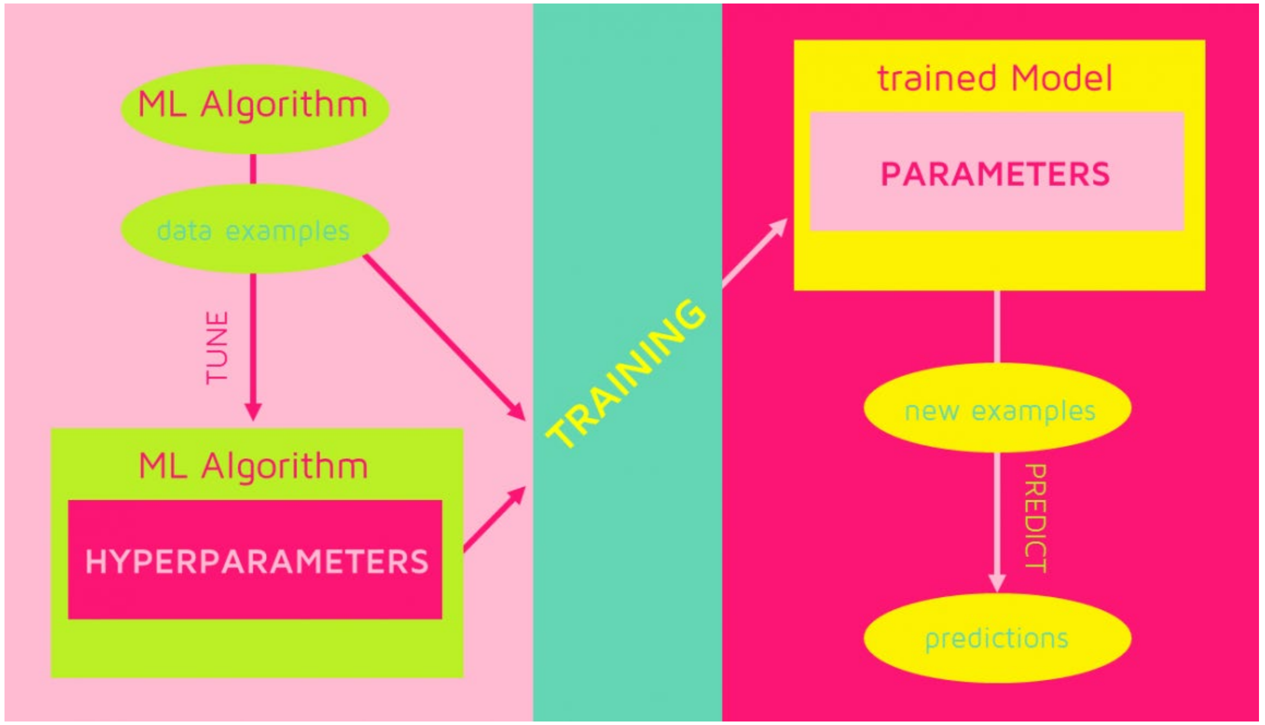
- We treat the parameter estimation problem as a problem of function optimization of Loss function (ε).
- There is lots of math, but it's very intuitive.
- The basic machine learning tasks: classification and prediction is formulated as a minimization task (optimization).

The slight difference between the loss function and the cost function is about the error within the training of machine learning models, as loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

https://www.javatpoint.com/gradient-descent-in-machine-learning

Parameters and hyperparameters in ML

- There is always a big confusion between Parameters and hyperparameters or model hyperparameters.
- **Model parameters:** are configuration variables that are internal to the model, and a model learns them on its own. For example, W (Weights) or Coefficients of independent variables (Beta) in the Linear regression model. Weights or Coefficients of independent variables in SVM, weight, and biases of a neural network, cluster centroid in clustering. Some key points for model parameters are as follows:
  - They are used by the model for making predictions.
  - They are learned by the model from the data itself
  - These are usually not set manually (needs optimization).
  - These are the part of the model and key to a machine learning Algorithm.
- **Hyperparameters:** These are adjustable parameters (explicitly defined by the user to control the learning process) that must be tuned in order to obtain a model with optimal performance:
  - These are usually defined manually by the machine learning engineer.
  - One cannot know the exact best value for hyperparameters for the given problem. The best value can be determined either by the rule of thumb or by trial and error.
  - Some examples of Hyperparameters are the learning rate for training a neural network, K in the KNN algorithm,
  - While hyperparameters are part of the input that we supply to the ML algorithm, parameters are the output as a result of fitting during training.
  - Two types of hyperparameters: Hyperparameter for Optimization, Hyperparameter for Specific Models

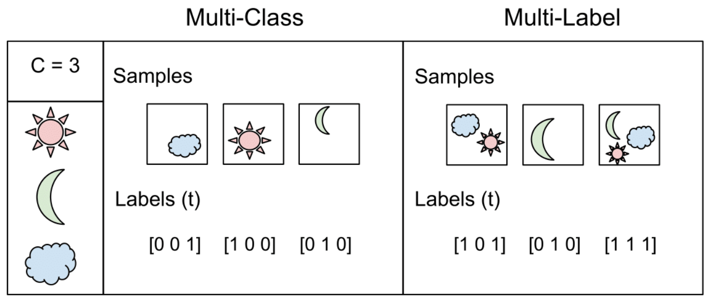


https://quantdare.com/what-is-the-difference-between-parameters-and-hyperparameters/
https://www.javatpoint.com/hyperparameters-in-machine-learning

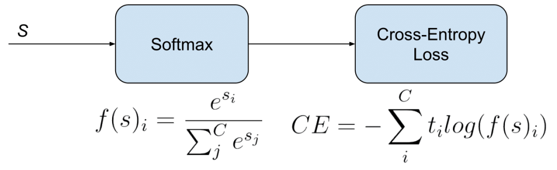
Loss function: L(w)

- Common Loss functions in machine learning: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23
- Machines learn by means of a loss function. It's a method of evaluating how well specific algorithm models the given data. If predictions deviates too much from actual results, loss function would cough up a very large number. Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction.
- Broadly, loss functions can be classified into two major categories depending upon the type of learning task we are dealing with: **Regression losses** and **Classification losses**.
- Loss functions for Regression: **Mean Square Error/ Mean Absolute Error/ Mean Bias Error**
- Loss functions for Classification: **Cross Entropy Loss/ SVM Loss** (see examples in https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e and https://gombru.github.io/2018/05/23/cross_entropy_loss/)

The classification problem



Activation function



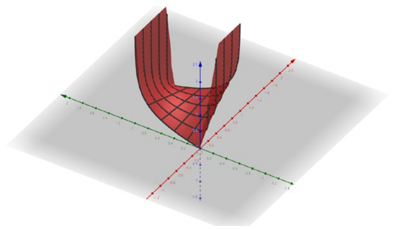
$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

Cross Entropy Loss

True probability distribution (one-hot)

$$H(p,q) = -\sum_{x \in \text{classes}} p(x) \log q(x)$$

Your model's predicted probability distribution



Parameters and hyperparameters in ML

PARAMETERS	HYPERPARAMETERS
They are required for making predictions	They are required for estimating the model parameters
They are estimated by optimization algorithms(Gradient Descent, Adam, Adagrad)	They are estimated by hyperparameter tuning
They are not set manually	They are set manually or tuning method
The final parameters found after training will decide how the model will perform on unseen data	The choice of hyperparameters decide how efficient the training is. In gradient descent the learning rate decide how efficient and accurate the optimization process is in estimating the parameters.

Example in a multiple linear regression model using an m-dimensional training data set :

$$\hat{y}_i = \sum_{j=0}^{m} X_{ij} w_j$$

Regression model

- where **X** is the predictor matrix, and **w** are the weights. Here w_0, w_1, w_2, …,w_m are the **model parameters**. If the model uses the gradient descent algorithm to minimize the objective function in order to determine the weights w_0, w_1, w_2, …,w_m, then we can have an optimizer such as GradientDescent(eta, n_iter).
- Here eta (learning rate) and n_iter (number of iterations) are the **hyperparameters** that would have to be adjusted in order to obtain the best values for the model parameters w_0, w_1, w_2, …,w_m.

$\downarrow$

Laboratory 2, 3

https://pub.towardsai.net/bad-and-good-regression-analysis-700ca9b506ff

https://www.geeksforgeeks.org/difference-between-model-parameters-vs-hyperparameters/

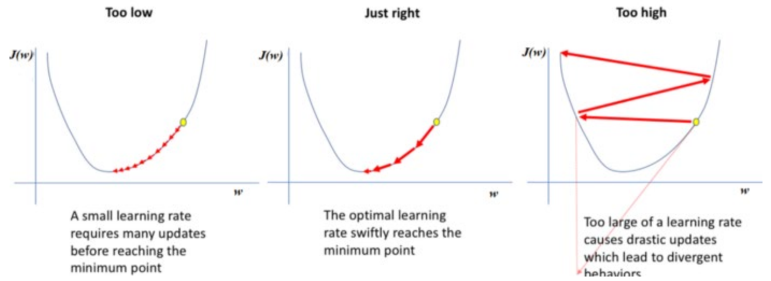
https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide

Hyperparameter tuning (tuning process)

- The process of selecting the best hyperparameters to use is known as hyperparameter tuning, and the tuning process is also known as hyperparameter optimization. Optimization parameters are used for optimizing the model.
- Some of the popular optimization Hyperparameters are given below:
    - **Learning Rate:** The learning rate is the hyperparameter in optimization algorithms that controls how much the model needs to change in response to the estimated error for each time when the model's weights are updated. It is one of the crucial parameters while building a neural network, and also it determines the frequency of cross-checking with model parameters.
    - **Batch Size:** To enhance the speed of the learning process, the training set is divided into different subsets, which are known as a batch.
    - **Number of Epochs:** An epoch can be defined as the complete cycle for training the machine learning model. Epoch represents an iterative learning process. The number of epochs varies from model to model, and various models are created with more than one epoch. To determine the right number of epochs, a validation error is taken into account. The number of epochs is increased until there is a reduction in a validation error. If there is no improvement in reduction error for the consecutive epochs, then it indicates to stop increasing the number of epochs.
- Choosing appropriate hyperparameters is an essential task when applying ML. Hyperparameters can affect the speed and also the accuracy of the final model. **Hyperparameter optimization** finds a tuple of hyperparameters that lead to the model which better solves the problem. Here, a list of the three most widespread algorithms to perform hyperparameters optimization:
    1. **Grid search**: It performs an exhaustive search by evaluating any candidates' combinations. Obviously, it could result in an unfeasible computing cost, so grid search is an option only when the number of candidates is limited enough.
    2. **Random search**: Providing a cheaper alternative, random search tests only as many tuples as you choose. The selection of the values to evaluate is completely random. Logically the required time decreases significantly. Apart from speed, Random search takes advantage of randomization in the case of continuous hyperparameters that must be discretized when optimized by Grid search.
    3. **Bayesian optimization**: Contrary to Grid and random search, Bayesian optimization uses previous iterations to guide the next ones. It consists of building a distribution of functions (Gaussian Process) that best describes the function to optimize. In this case, hyperparameter optimization, the function to optimize is those which, given the hyperparameters, returns the performance of the trained model they would lead to. After every step, this distribution of functions is updated and the algorithm detects which regions in the hyperparameter space are more interesting to explore and which are not. After a defined number of iterations, the algorithm stops and returns the optimum tuple. Bayesian optimization is a more efficient method for exploring the possibilities.

If we denote dw and db as gradients to update our parameters W and b for gradient descent algorithm as follows: W = W – learning rate * dW

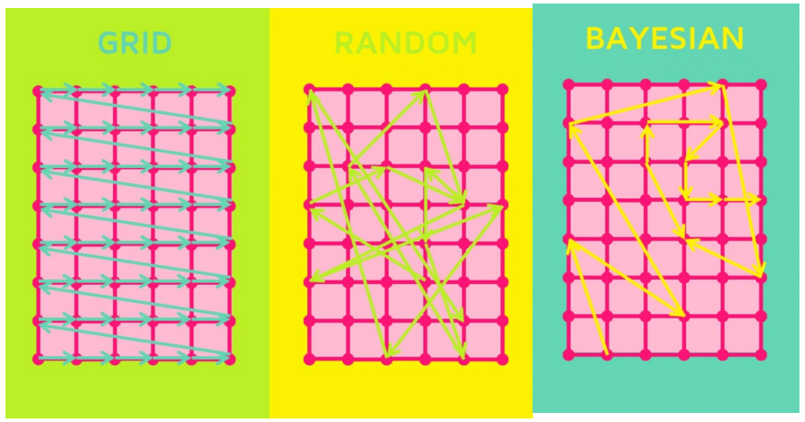
b = b – learning rate * db



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

Selecting the optimized learning rate is a challenging task because if the learning rate is very less, then it may slow down the training process. On the other hand, if the learning rate is too large, then it may not optimize the model properly.

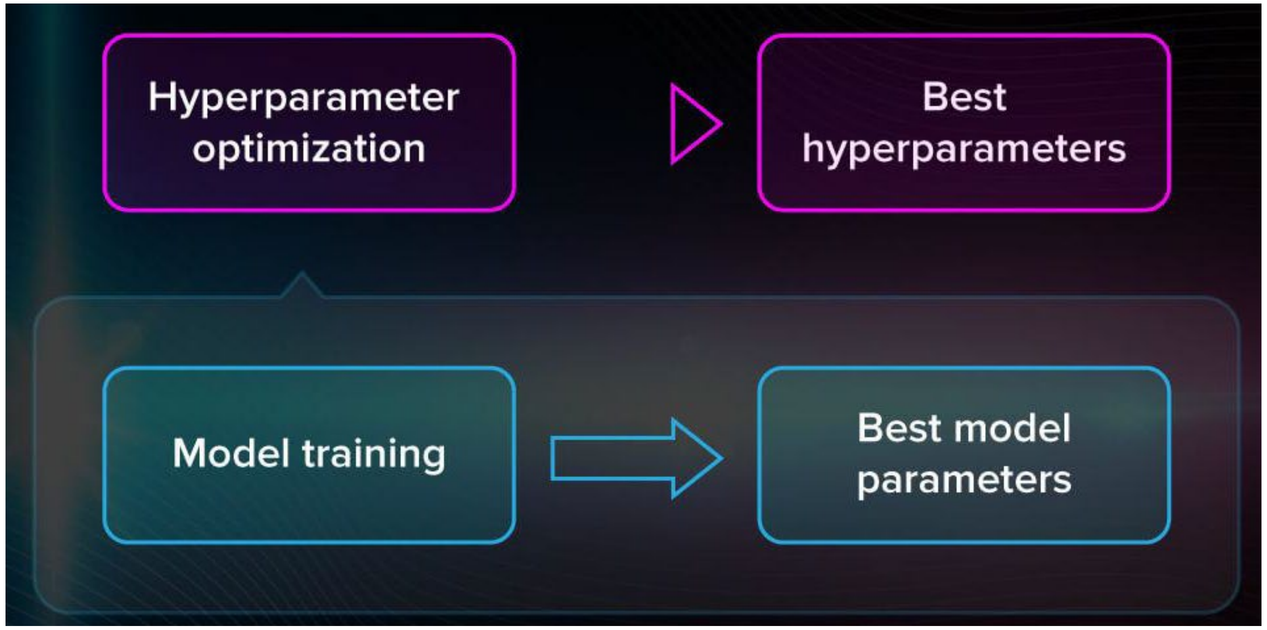
https://towardsdatascience.com/hyper-parameter-tuning-techniques-in-deep-learning-4dad592c63c8

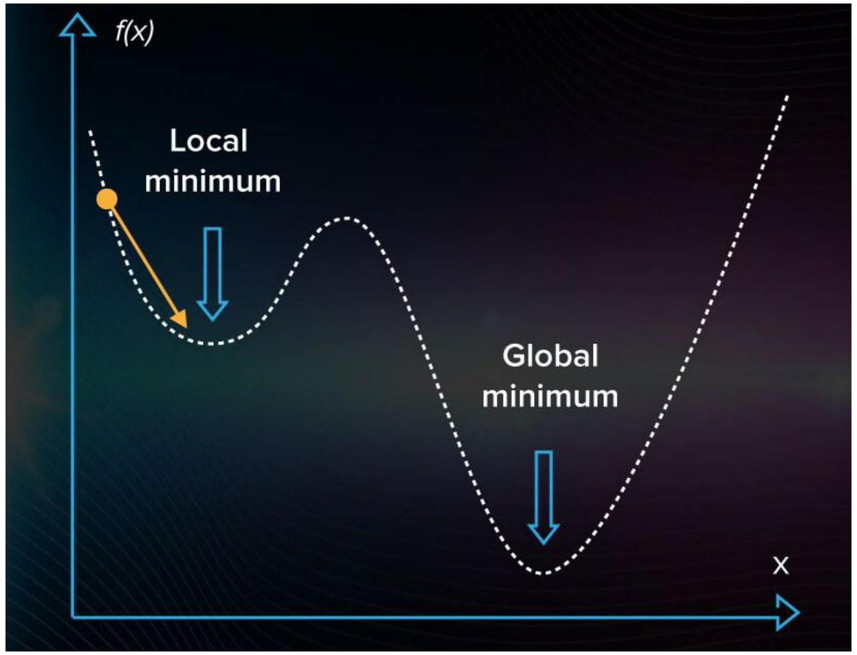


https://quantdare.com/what-is-the-difference-between-parameters-and-hyperparameters/
https://www.javatpoint.com/hyperparameters-in-machine-learning

Parameters and optimization in ML

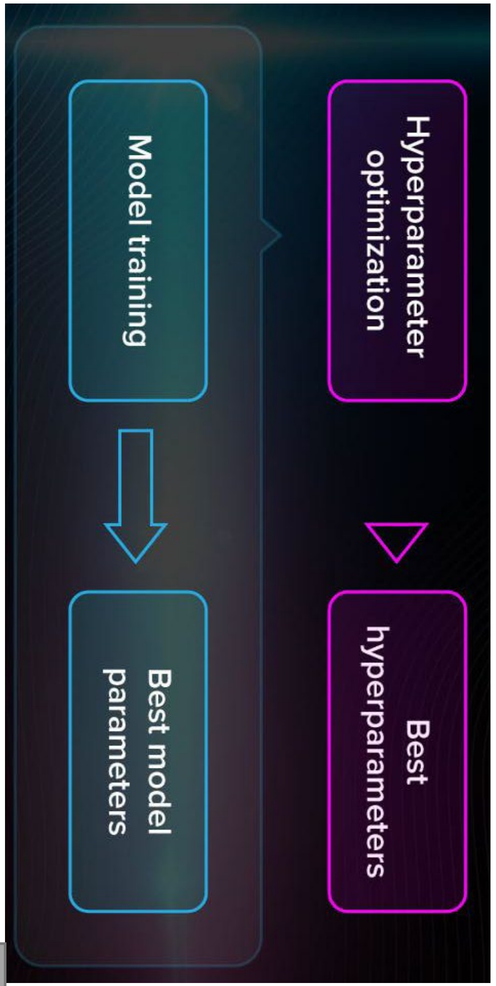
- Machine learning uses optimization for tuning all the parameters of various algorithms.
- Basics about optimization:
  - There are different algorithms (methods) to calculate parameters based on optimization (e.g. descendent gradient)
  - Before stepping into gradient descent, the introduction of mathematics and the concept of convex and non-convex functions is very helpful.





Parameters Optimization

- Optimization is the process where we train the model iteratively that results in a maximum and minimum function evaluation.
- It is one of the most important phenomena in Machine Learning to get better results.
- Why do we optimize our machine learning models?:
  - We compare the results in every iteration by changing the hyperparameters in each step until we reach the optimum results.
  - We create an accurate model with less error rate.
  - There are different ways using which we can optimize a model.
  - Two important Optimization algorithms:
    - Gradient Descent
    - Stochastic Gradient Descent Algorithms



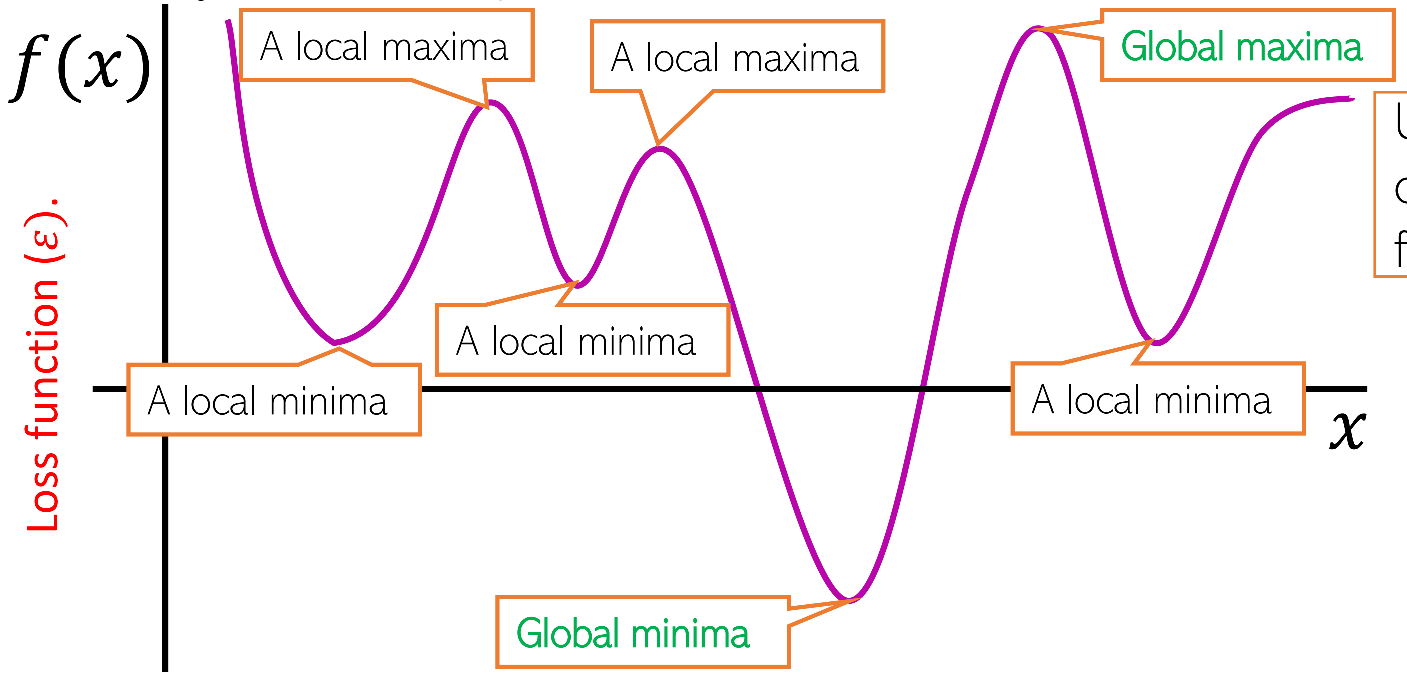
We are going to see a theoretical introduction to optimization in ML and some of the main methods used in practice, introduced by Dr **Srivastava** in https://www.cse.iitk.ac.in/users/nsrivast/

Optimization: Functions and their optima (review of theoretical concepts)

The objective function of the ML problem we are solving (e.g., squared loss for regression)

- Many ML problems require us to optimize a function f of some variable(s) x
- For simplicity, assume f is a scalar-valued function of a scalar x ($f: \mathbb{R} \to \mathbb{R}$)

Assume unconstrained for now, i.e., just a real-valued number/vector



Usually interested in global optima but often want to find local optima, too

Will see what these are later

- Any function has one/more optima (maxima, minima), and maybe saddle points

- Finding the optima or saddles requires derivatives/gradients of the function

Optimization: Derivatives

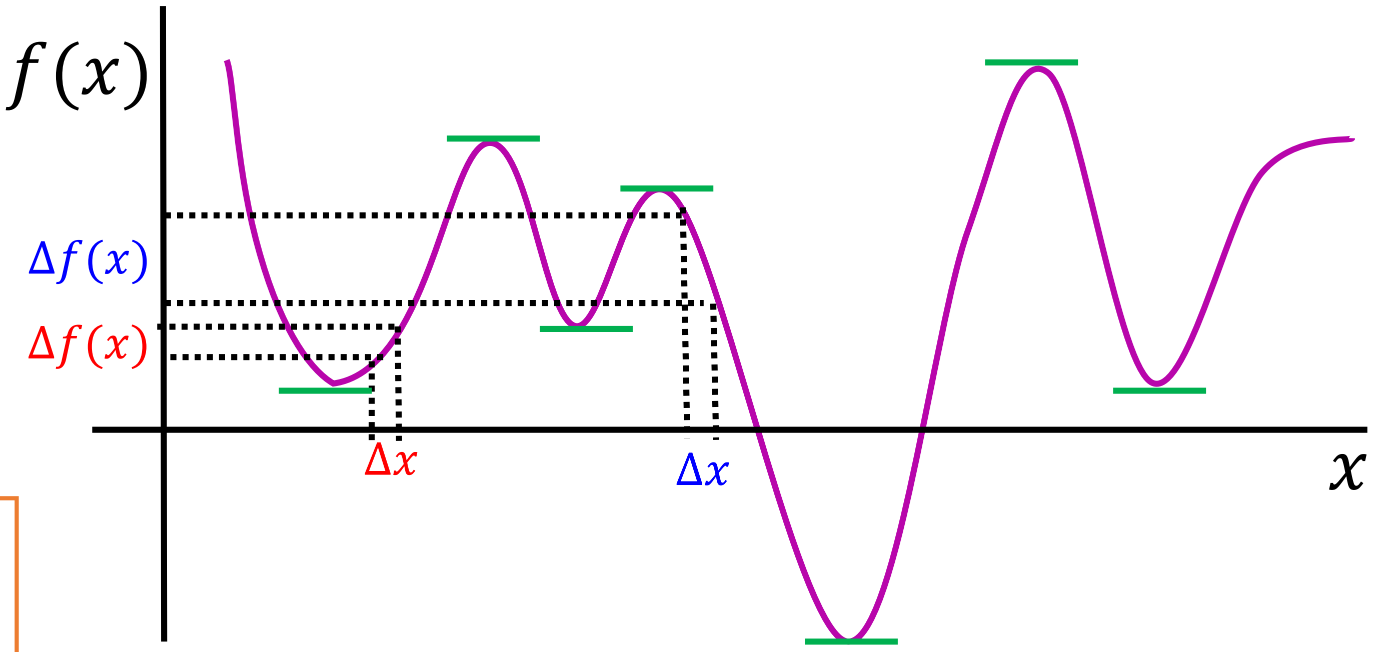
Will sometimes use $f'(x)$ to denote the derivative

- Magnitude of derivative at a point is the rate of change of the func at that point

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{\Delta f(x)}{\Delta x}$$

Sign is also important: Positive derivative means f is increasing at x if we increase the value of x by a very small amount; negative derivative means it is decreasing

Understanding how f changes its value as we change x is helpful to understand optimization (minimization/maximization) algorithms



- Derivative becomes zero at stationary points (optima or saddle points)
  - The function becomes "flat" ($\Delta f(x) = 0$ if we change x by a very little at such points)
  - These are the points where the function has its maxima/minima (unless they are saddles)

Optimization: Rules of Derivatives

Some basic rules of taking derivatives

- Sum Rule: $\left(f(x) + g(x)\right)' = f'(x) + g'(x)$
- Scaling Rule: $\left(a \cdot f(x)\right)' = a \cdot f'(x)$ if a is not a function of x
- Product Rule: $\left(f(x) \cdot g(x)\right)' = f'(x) \cdot g(x) + g'(x) \cdot f(x)$
- Quotient Rule: $\left(f(x)/g(x)\right)' = \left(f'(x) \cdot g(x) - g'(x)f(x)\right)/\left(g(x)\right)^2$
- Chain Rule: $\left(f\left(g(x)\right)\right)' \overset{\text{def}}{=} (f \circ g)'(x) = f'\left(g(x)\right) \cdot g'(x)$

- We already used some of these (sum, scaling and chain) when calculating the derivative for the linear regression model

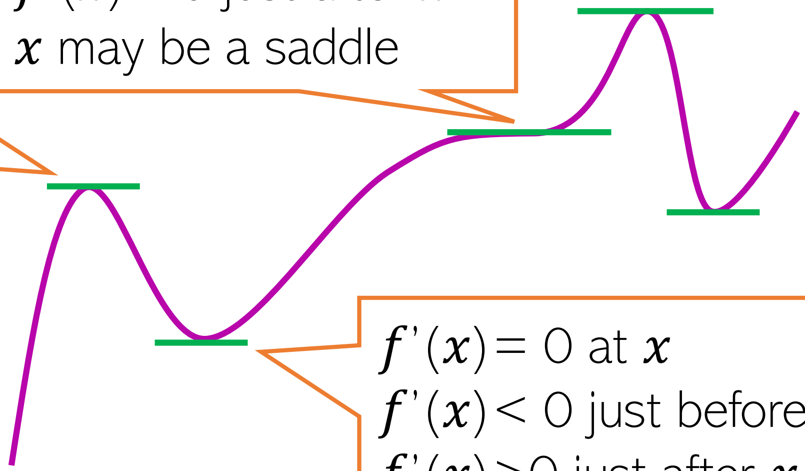
https://www.cse.iitk.ac.in/users/nsrivast/

Optimization: Derivatives

- How the derivative itself changes tells us about the function’s optima

$f'(x) = 0$ at x,
$f'(x) > 0$ just before x
$f'(x) < 0$ just after x
x is a maxima

$f'(x) = 0$ at x
$f'(x) = 0$ just before x
$f'(x) = 0$ just after x
x may be a saddle



$f'(x) = 0$ at x
$f'(x) < 0$ just before x
$f'(x) > 0$ just after x
x is a minima

$f'(x) = 0$ and $f''(x) < 0$
x is a maxima

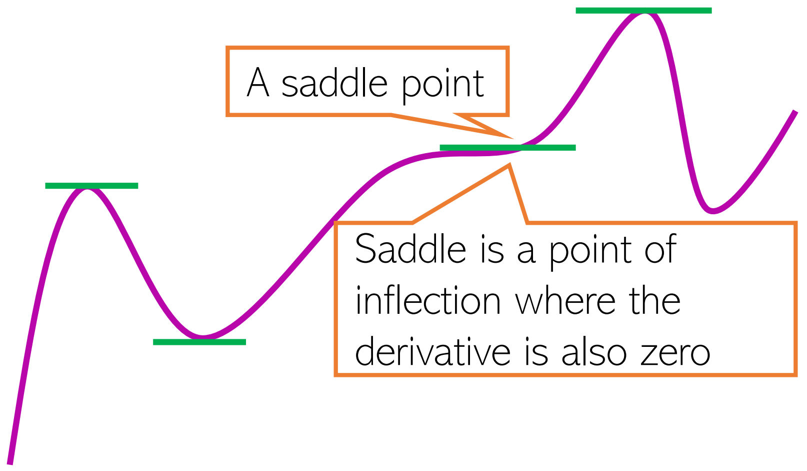
$f'(x) = 0$ and $f''(x) > 0$
x is a minima

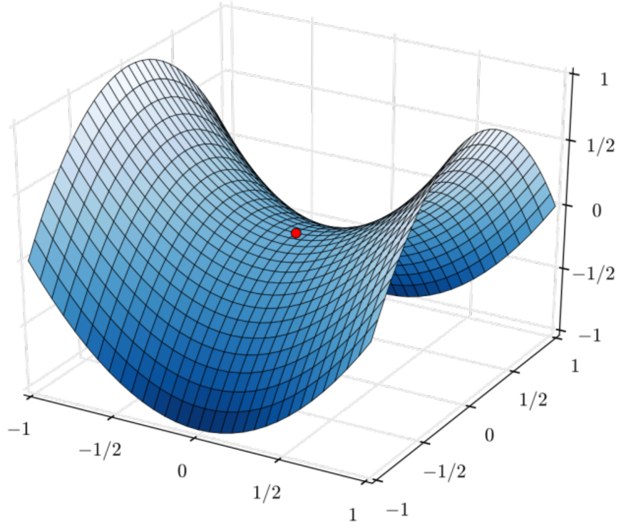
$f'(x) = 0$ and $f''(x) = 0$
x may be a saddle. May need higher derivatives

- The second derivative $f''(x)$ can provide this information

Optimization: Saddle Points

- Points where derivative is zero but are neither minima nor maxima


A saddle point
Saddle is a point of inflection where the derivative is also zero



- Saddle points are very common for loss functions of ML models
  - Need to be handled carefully during optimization

- Second or higher derivative may help identify if a stationary point is a saddle

https://www.cse.iitk.ac.in/users/nsrivast/

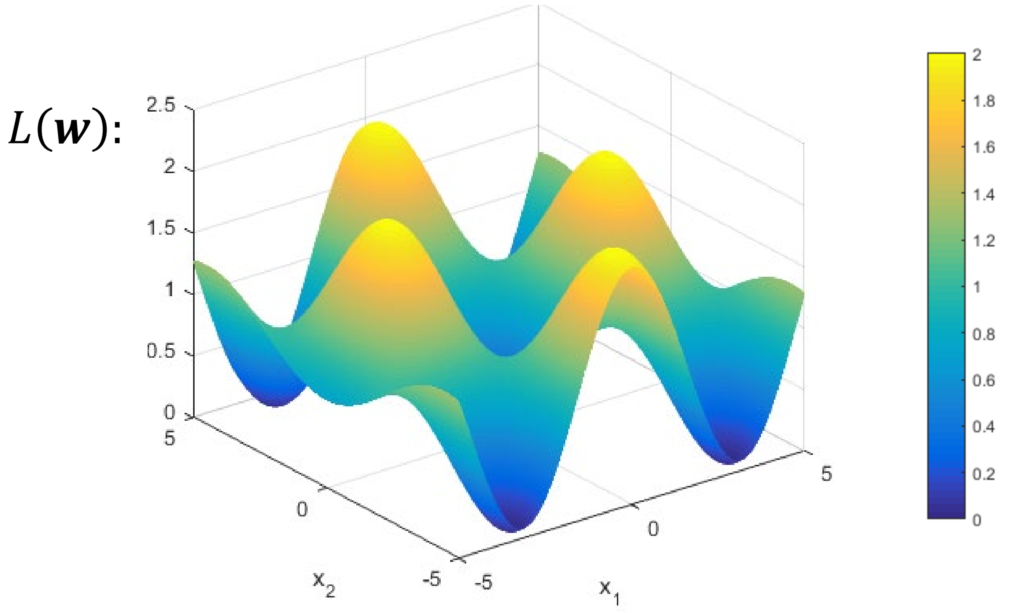
Optimization: Multivariate Functions

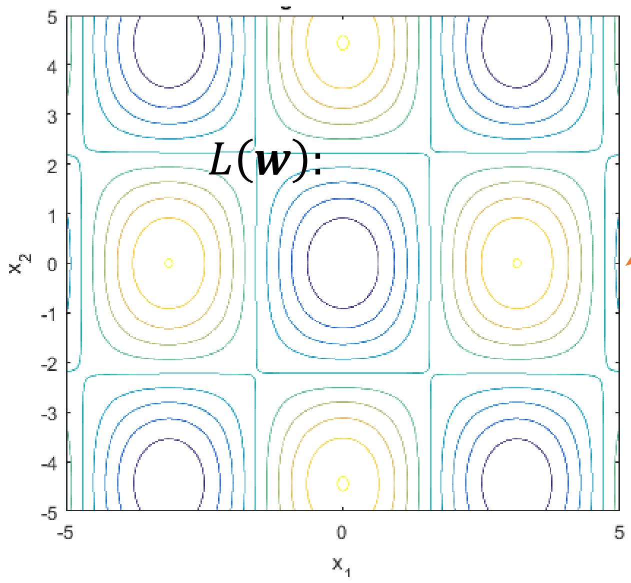
- Most functions that we see in ML are multivariate function (great difficulty, many times with 0, redundant information)

- Example: Loss fn $L(\boldsymbol{w})$ in lin-reg was a multivar. function of D-dim vector $\boldsymbol{w}$

$$L(\boldsymbol{w}): \mathbb{R}^D \to \mathbb{R}$$

- Here is an illustration of a function of 2 variables (4 maxima and 5 minima)





Two-dim contour plot of the function (i.e., what it looks like from the above)

Plot courtesy: http://benchmarkfcns.xyz/benchmarkfcns/griewankfcn.html

https://www.cse.iitk.ac.in/users/nsrivast/

Optimization: Derivatives of Multivariate Functions

- Can define derivative for a multivariate functions as well via the gradient
- Gradient of a function $f(\boldsymbol{x}): \mathbb{R}^D \rightarrow \mathbb{R}$ is a $D \times 1$ vector of partial derivatives

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_D}\right)$$

Each element in this gradient vector tells us how much f will change if we move a little along the corresponding (similar to one-dim case)

- Optima and saddle points defined similar to one-dim case
  - Required properties that we saw for one-dim case must be satisfied along all the directions
- The second derivative in this case is known as the **Hessian**

Optimization: The Hessian

- For a multivar scalar valued function $\boldsymbol{f(x)}: \mathbb{R}^D \to \mathbb{R}$, Hessian is a $D \times D$ matrix

$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_D} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_D x_1} & \frac{\partial^2 f}{\partial x_D x_2} & \cdots & \frac{\partial^2 f}{\partial x_D^2} \end{bmatrix}$$

Gives information about the curvature of the function at point $\boldsymbol{x}$

Note: If the function itself is vector valued, e.g., $\boldsymbol{f(x)}: \mathbb{R}^D \to \mathbb{R}^K$ then we will have K such $D \times D$ Hessian matrices, one for each output dimension of $\boldsymbol{f}$

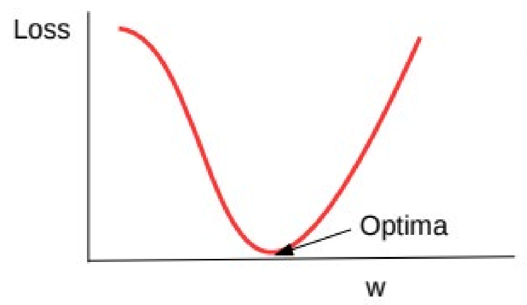
A square, symmetric $D \times D$ matrix M is PSD if $\boldsymbol{x}^\top \boldsymbol{M x} \geq \boldsymbol{0}\ \forall\ \boldsymbol{x} \in \mathbb{R}^D$
Will be NSD if $\boldsymbol{x}^\top \boldsymbol{M x} \leq \boldsymbol{0}\ \forall\ \boldsymbol{x} \in \mathbb{R}^D$

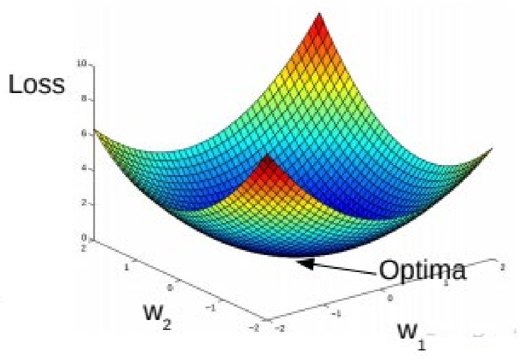
PSD if all eigenvalues are non-negative

- The Hessian matrix can be used to assess the optima/saddle points
  - $\nabla f(\boldsymbol{x}) = 0$ and $\nabla^2 f(\boldsymbol{x})$ is a positive semi-definite (PSD) matrix then $\boldsymbol{x}$ is a minima
  - $\nabla f(\boldsymbol{x}) = 0$, and $\nabla^2 f(\boldsymbol{x})$ is a negative semi-definite (NSD) matrix then $\boldsymbol{x}$ is a maxima

Optimization: Convex and Non-Convex Functions

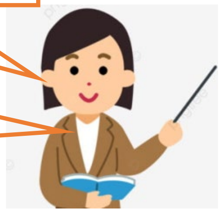
- A function being optimized can be either convex or non-convex
- Here are a couple of examples of convex functions





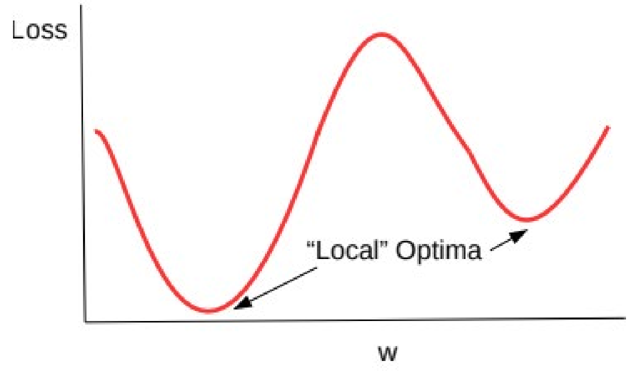
Convex functions are bowl-shaped. They have a unique optima (minima)

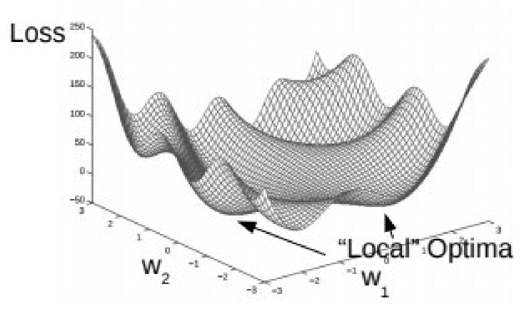
Negative of a convex function is called a concave function, which also has a unique optima (maxima)



- Here are a couple of examples of non-convex functions

Non-convex functions have multiple minima. Usually harder to optimize as compared to convex functions



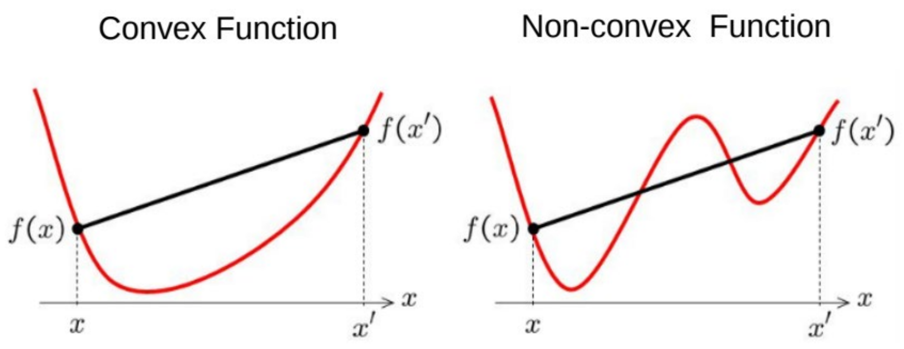


Loss functions of most deep learning models are non-convex

https://www.cse.iitk.ac.in/users/nsrivast/

Optimization: Convex Functions

- Informally, $f(x)$ is convex if all of its chords lie above the function everywhere



Note: "Chord lies above function" more formally means

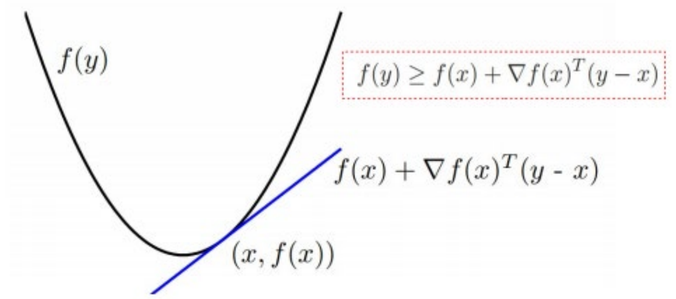
If *f* is convex then given

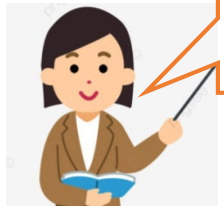
$\alpha_1, \ldots, \alpha_n$ s.t $\sum_{i=1}^{n} \alpha_i = 1$

$$f\left(\sum_{i=1}^{n} \alpha_i x_i\right) \leq \sum_{i=1}^{n} \alpha_i f(x_i)$$

Jensen's Inequality

- Formally, (assuming differentiable function), some tests for convexity:
  - First-order convexity (graph of f must be above all the tangents)


$f(y)$
$f(y) \geq f(x) + \nabla f(x)^T (y - x)$
$f(x) + \nabla f(x)^T (y - x)$
$(x, f(x))$

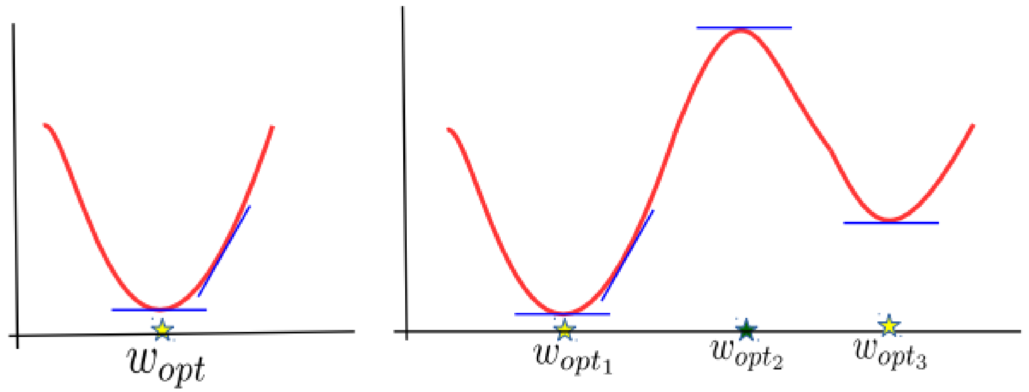


Exercise: Show that ridge regression objective is convex

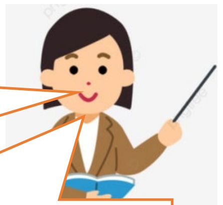
  - Second derivative a.k.a. Hessian (if exists) must be positive semi-definite

Optimization: Optimization Using First-Order Optimality

- Very simple. Already used this approach for linear and ridge regression



Called "first order" since only gradient is used and gradient provides the first order info about the function being optimized

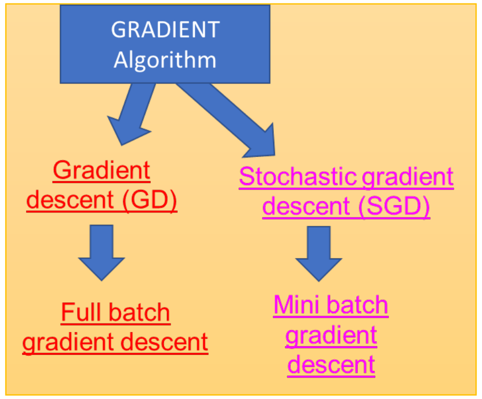


The approach works only for very simple problems where the objective is convex and there are no constraints on the values $\boldsymbol{w}$ can take

- First order optimality: The gradient $\boldsymbol{g}$ must be equal to zero at the optima

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}}[L(\boldsymbol{w})] = \mathbf{0}$$

- Sometimes, setting $\boldsymbol{g} = \mathbf{0}$ and solving for $\boldsymbol{w}$ gives a closed form
- solution
- If closed form solution is not available, the gradient vector $\boldsymbol{g}$ can still be used in iterative optimization algos, like **gradient descent**



GRADIENT Algorithm
Gradient descent (GD)
Stochastic gradient descent (SGD)
Full batch gradient descent
Mini batch gradient descent

https://www.cse.iitk.ac.in/users/nsrivast/

Optimization algorithms in ML

$$\begin{tikzcd} & \text{GRADIENT Algorithm} \arrow[dl] \arrow[dr] & \\ \text{Gradient descent (GD)} \arrow[d] & & \text{Stochastic gradient descent (SGD)} \arrow[d] \\ \text{Full batch gradient descent} & & \text{Mini batch gradient descent} \end{tikzcd}$$

- Gradient descent (GD): This is a way to minimize the objective function L(w) parameterized by the model's parameter w by updating the parameters in the opposite direction to the gradient of the objective function L(w) with respect to the parameters (w). The learning rate determines the size of steps taken to reach the minimum. See example in https://en.wikipedia.org/wiki/Gradient_descent
- Full batch gradient descent (all training observations considered in each and every iteration): In full batch gradient descent, all the observations are considered for each and every iteration; this methodology takes a lot of memory and will be slow as well. Also, in practice, we do not need to have all the observations to update the weights. Nonetheless, this method provides the best way of updating parameters with less noise at the expense of huge computation.
- Stochastic gradient descent (SGD) (one observation per iteration): This method updates weights by taking one observation at each stage of iteration. This method provides the quickest way of traversing weights; however, a lot of noise is involved while converging.
- Mini batch gradient descent (about 30 training observations or more for each and every iteration): This is a trade-off between huge computational costs and a quick method of updating weights. In this method, at each iteration, about 30 observations will be selected at random and gradients calculated to update the model weights. Here, a question many can ask is, why the minimum 30 and not any other number? If we look into statistical basics, 30 observations required to be considering in order approximating sample as a population. However, even 40, 50, and so on will also do well in batch size selection. Nonetheless, a practitioner needs to change the batch size and verify the results, to determine at what value the model is producing the optimum results.
- See more in: https://towardsdatascience.com/understanding-optimization-algorithms-in-machine-learning-edfdb4df766b
- https://towardsdatascience.com/gradient-descent-clearly-explained-in-python-part-1-the-troubling-theory-49a7fa2c4c06
- https://keepcoding.io/blog/stochastic-gradient-descent-deep-learning/
- https://www.cse.iitk.ac.in/users/nsrivast/ https://datascience.stackexchange.com/questions/36450/what-is-the-difference-between-gradient-descent-and-stochastic-gradient-descent

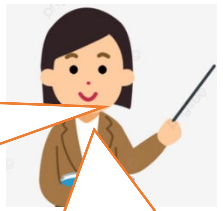
The red path is the one followed by the gradient descent, which, when calculating the gradient using all the samples of the dataset, always achieves consistent updates in the direction that allows minimizing the error. On the other hand, the magenta path is the one followed by the SGD. What is happening? In both gradient descent (GD) and stochastic gradient descent (SGD), you update a set of parameters in an iterative manner to minimize an error function. While in GD, you have to run through ALL the samples in your training set to do a single update for a parameter in a particular iteration, in SGD, on the other hand, you use ONLY ONE or SUBSET of training sample from your training set to do the update for a parameter in a particular iteration. If you use SUBSET, it is called Minibatch Stochastic gradient Descent. Thus, if the number of training samples are large, in fact very large, then using gradient descent may take too long because in every iteration when you are updating the values of the parameters, you are running through the complete training set. On the other hand, using SGD will be faster because you use only one training sample and it starts improving itself right away from the first sample. SGD often converges much faster compared to GD but the error function is not as well minimized as in the case of GD. Often in most cases, the close approximation that you get in SGD for the parameter values are enough because they reach the optimal values and keep oscillating there.

Optimization: Optimization via Gradient Descent

Can I used this approach to solve maximization problems?

For max. problems we can use gradient ascent $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta_t \mathbf{g}^{(t)}$

Iterative since it requires several steps/iterations to find the optimal solution



Fact: Gradient gives the direction of steepest change in function's value

Will move in the direction of the gradient

For convex functions, GD will converge to the global minima

Good initialization needed for non-convex functions

Gradient Descent

- Initialize $\mathbf{w}$ as $\mathbf{w}^{(0)}$
- For iteration $t = 0,1,2,\ldots$ (or until convergence)
  - Calculate the gradient $\mathbf{g}^{(t)}$ using the current iterates $\mathbf{w}^{(t)}$
  - Set the learning rate η_t
  - Move in the opposite direction of gradient

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \mathbf{g}^{(t)}$$

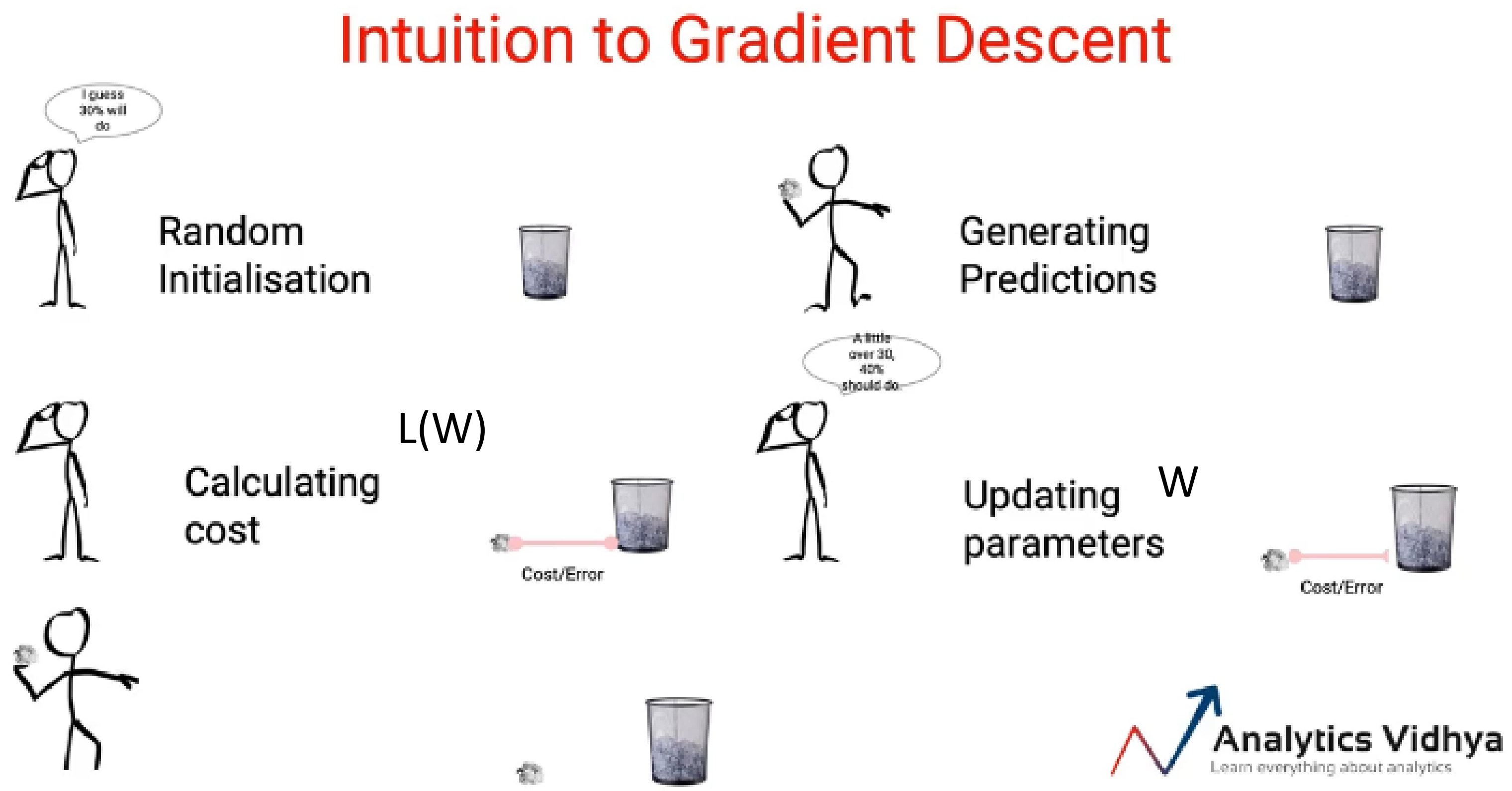
Will see the justification shortly

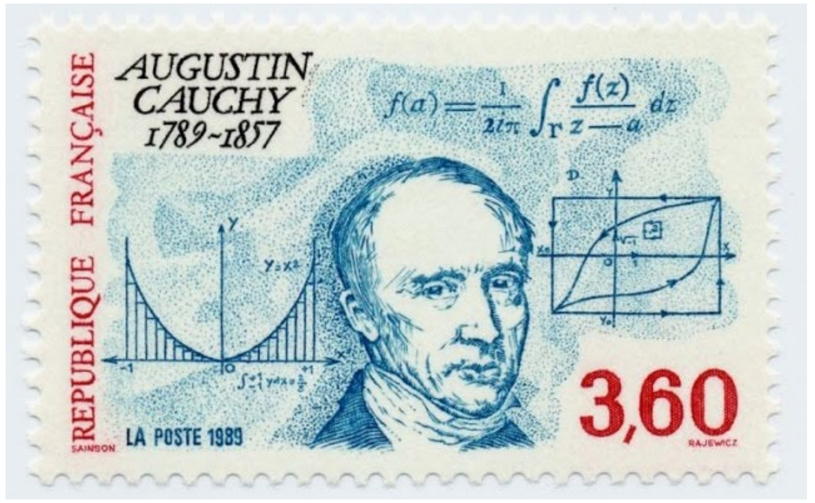
The learning rate very imp. Should be set carefully (fixed or chosen adaptively). Will discuss some strategies later

Sometimes may be tricky to to assess convergence? Will see some methods later

Optimization: Optimization via Gradient Descent

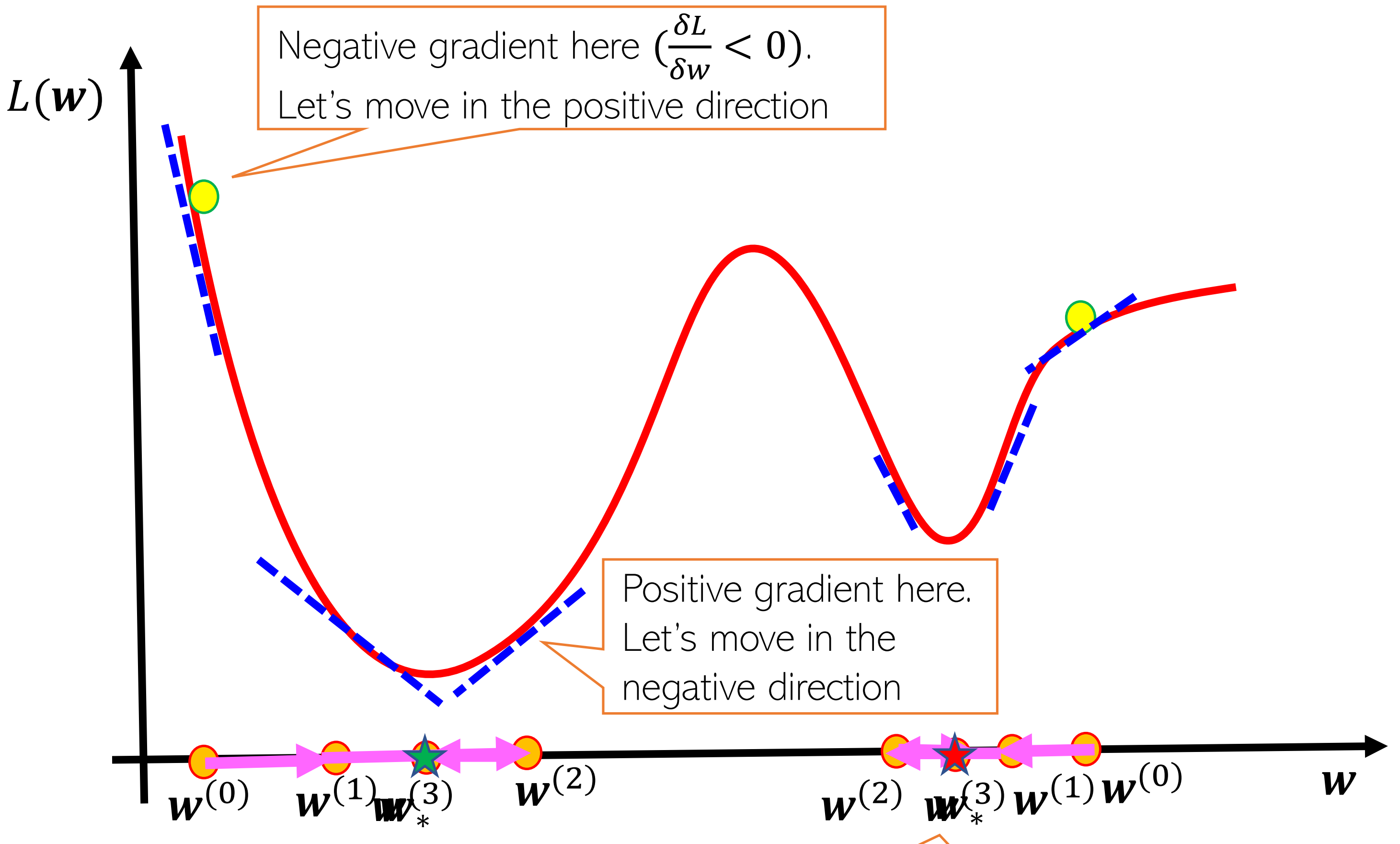
Gradient descent was initially discovered by "Augustin-Louis Cauchy" in mid of 18th century. Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function





https://www.analyticsvidhya.com/blog/2021/03/understanding-gradient-descent-algorithm/

Gradient Descent: An Illustration



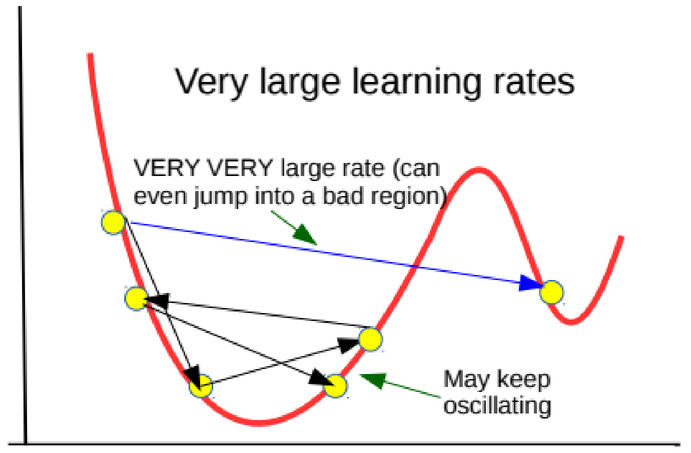
Negative gradient here ($\frac{\delta L}{\delta w} < 0$). Let's move in the positive direction

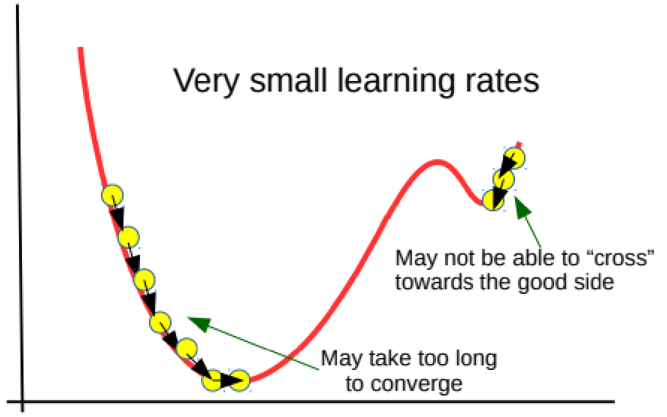
Positive gradient here. Let's move in the negative direction

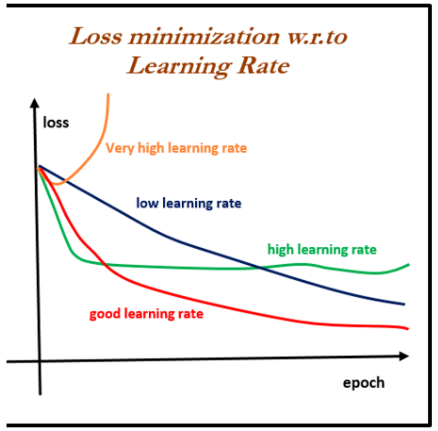
Stuck at a local minima

Good initialization is very important

Learning rate is very important


Very large learning rates
VERY VERY large rate (can even jump into a bad region)
May keep oscillating


Very small learning rates
May not be able to "cross" towards the good side
May take too long to converge



https://www.cse.iitk.ac.in/users/nsrivast/

GD: An Example LEAST SQUARES LINEAR REGRESSION

- Let's apply GD for least squares linear regression

$$\boldsymbol{w}_{ridge} = \arg\min_{\boldsymbol{w}} L_{reg}(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- The gradient: $\boldsymbol{g} = -\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n$

- Each GD update will be of the form

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \sum_{n=1}^{N} 2 \left(y_n - {\boldsymbol{w}^{(t)}}^\top \boldsymbol{x}_n \right) \boldsymbol{x}_n$$

Prediction error of current model $\boldsymbol{w}^{(t)}$ on the $\boldsymbol{n}^{th}$ training example

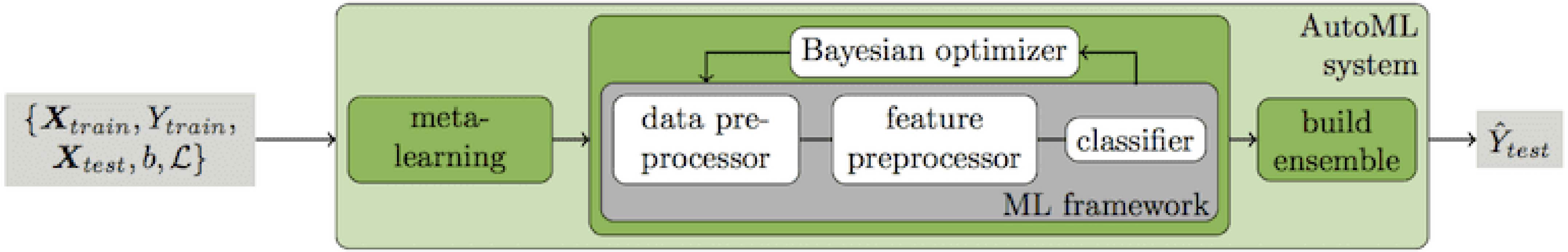
Training examples on which the current model's error is large contribute more to the update

- If you need an example of GD with a practical case, check Andrew NG's notes here where he clearly shows you the steps involved the use of GD: https://web.archive.org/web/20180618211933/http://cs229.stanford.edu/notes/cs229-notes1.pdf or https://medium.com/swlh/the-math-of-machine-learning-i-gradient-descent-with-univariate-linear-regression-2afbfb556131

AUTOML

- AUTOML allows the generation of models in a simple way aimed at people without knowledge of ML, for this AutoML enables the automation of manual and repetitive tasks in the development process of ML models, which allows speeding up their development, reducing errors and costs, as well as generate inference results more accurately
- It reduces the time and, depending on the use case, it can mean a reduction in the cost of developing ML models.
- It democratizes ML, enabling the development of ML models for people who don't have deep knowledge of data science.
- Promotes the use of ML models in companies and institutions where there are no ML experts.
- It allows you to speed up experimentation with ML models and thus lay the foundations for later refining the model that offers the best results.

Auto-Sklearn for Automated Machine Learning in Python
https://machinelearningmastery.com/auto-sklearn-for-automated-machine-learning-in-python/



- **Auto-Sklearn** is an open-source library for performing AutoML in Python. It makes use of the popular Scikit-Learn machine learning library for data transforms and machine learning algorithms and uses a Bayesian Optimization search procedure to efficiently discover a top-performing model pipeline for a given dataset.
- Other is ***Pycaret:*** ***https://pycaret.gitbook.io/docs/***

References

- https://github.com/mackelab/machine-learning-I
- https://www.cs.ubc.ca/~murphyk/MLbook/pml-intro-22may12.pdf
- https://www.cse.iitk.ac.in/users/nsrivast/
- https://web.archive.org/web/20180618211933/http://cs229.stanford.edu/notes/cs229-notes1.pdf
- https://medium.com/swlh/the-math-of-machine-learning-i-gradient-descent-with-univariate-linear-regression-2afbfb556131
- https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23