:: Now let's study some popular algorithms

It's too difficult to learn the algorithms because I have already forgotten the math

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j}\ell(\theta) &= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)\frac{\partial}{\partial \theta_j}g(\theta^T x) \\
&= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)g(\theta^T x)(1-g(\theta^T x)\frac{\partial}{\partial \theta_j}\theta^T x \\
&= \left(y(1-g(\theta^T x)) - (1-y)g(\theta^T x)\right)x_j \\
&= \left(y - h_\theta(x)\right)x_j
\end{aligned}
$$

:: Now let’s study some popular algorithms

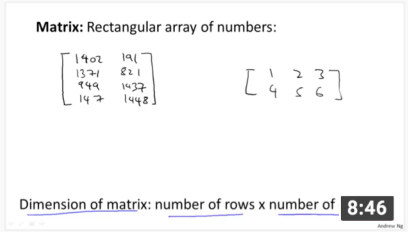
Don’t Worry

The formula in every algorithm will be explained in details.

:: Linear Algebra Basic (Optional)

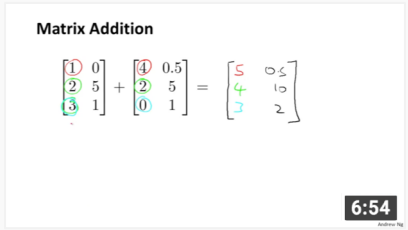
Excellent videos tutorial by Andrew Ng

Just take about 1 hour to learn the basic concept of Linear Algebra. It's strongly recommended to watches these videos. Don't worry, It's really really easy !!!



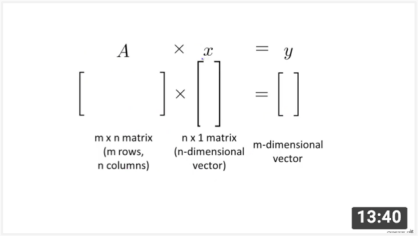
1 Matrices and Vectors

https://youtu.be/Dft1cqjwlXE



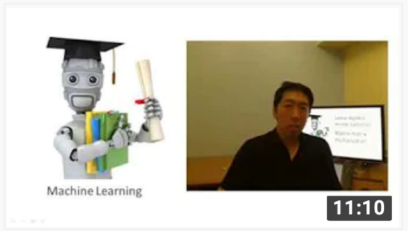
2 Addition and Scalar Multiplication

https://youtu.be/4WP6jVGIn7M



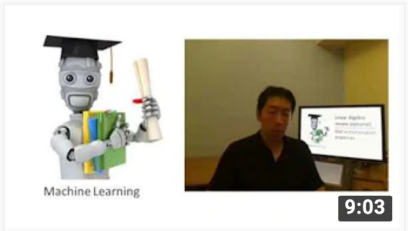
3 Matrix Vector Multiplication

https://youtu.be/gPegoVYp64w



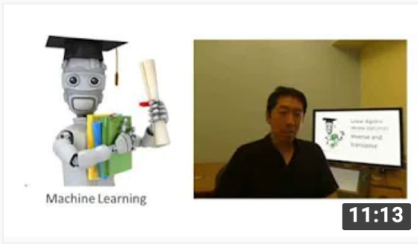
4 Matrix Matrix Multiplication

https://youtu.be/_lrHXJRukMw



5 Matrix Multiplication Properties

https://youtu.be/c7GhnL2N--I



6 Inverse and Transpose

https://youtu.be/7snro4M6ukk

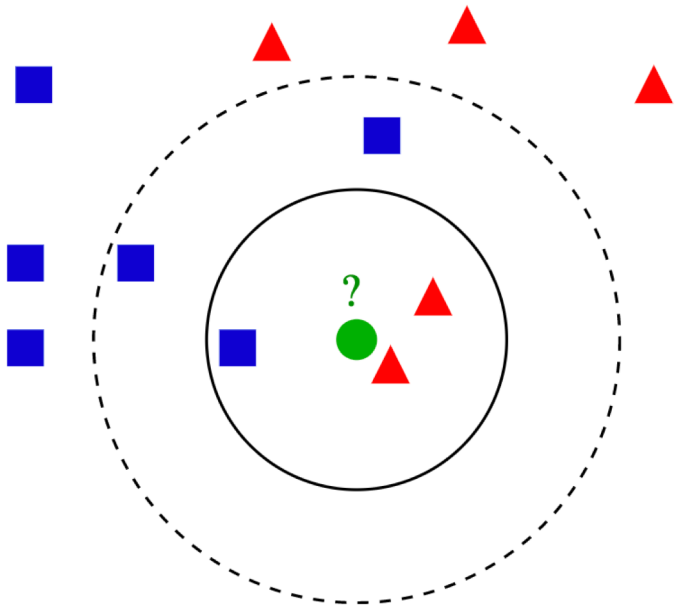
Picture: Designed by Asierromero / Freepik

k Nearest Neighbor

:: k Nearest Neighbor (kNN)

kNN categorizes objects based on the classes of their nearest neighbors in the dataset

kNN predictions assume that objects near each other are similar. Distance metrics, such as Euclidean, city block, cosine, and Chebychev, are used to find the nearest neighbor.



Example of *k*-NN classification

The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.

- If *k = 3* (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.
- If *k = 5* (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

1 Source : Mathworks, Applying Supervised Learning
Source of diagram: wikipedia

:: k Nearest Neighbor (kNN)

An object is classified by a majority vote of its neighbors

- **KNN algorithm is one of the simplest classification algorithm**

  Despite its simplicity, nearest neighbors has been successful in a large number of classification and regression problems, including handwritten digits or satellite image scenes. [1]

- **Often used in classification**

  Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.[2]

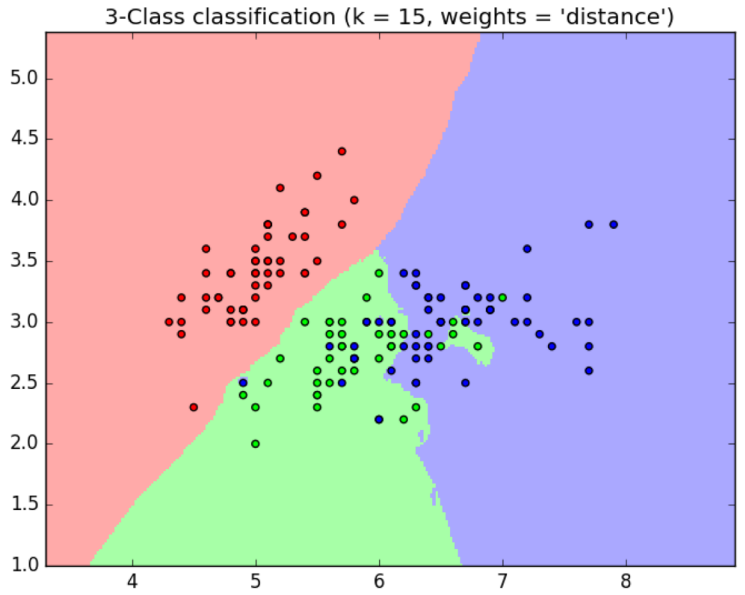
- **Classification is computed from a simple majority vote of the nearest neighbors of each point**

  In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.

- **K is constant specified by user**

  In the classification phase, k is a user-defined constant. The best choice of k depends upon the data; generally, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct.

- **KNN is computationally expensive**

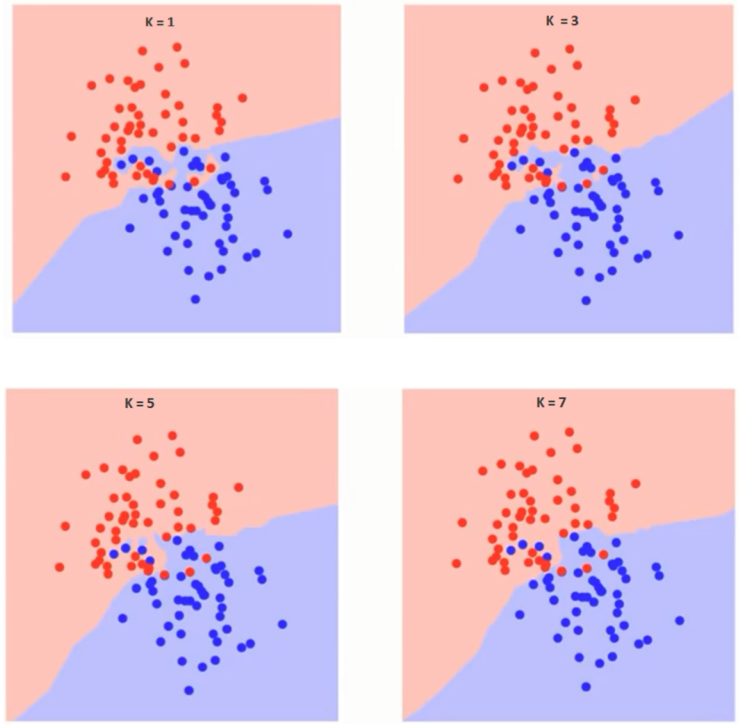


Source: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
1,2 source: http://scikit-learn.org/stable/modules/neighbors.html
Source of diagram: http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py

:: k Nearest Neighbor (kNN)

How do we choose the factor K?

let's try to see the effect of value "K" on the class boundaries. Following are the different boundaries separating the two classes with different values of K.[1]



If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority.

At times, choosing K turns out to be a challenge while performing KNN modeling.[2]

1,2 & image Source: https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/

:: k Nearest Neighbor (kNN)

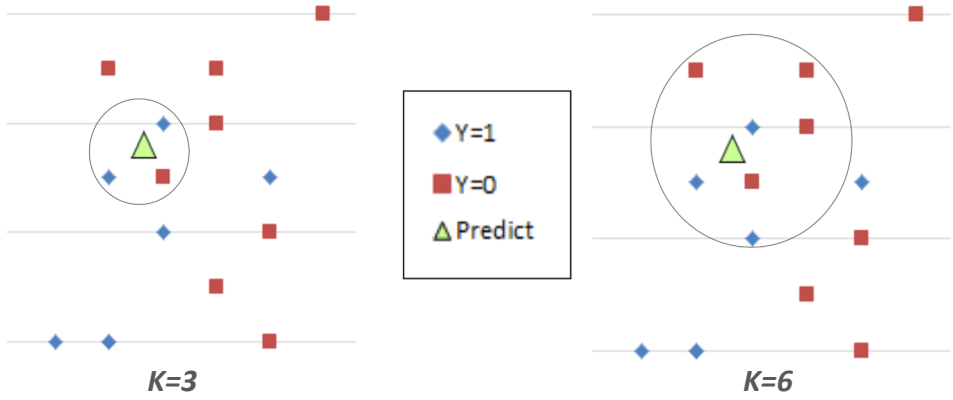
Strengths and Weakness of kNN

Strengths [1]

- It is beautifully simple and logical

Weaknesses [2]

- It may be driven by the choice of k, which may be a bad choice.
- Generally, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct.
- The accuracy of the algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance.
- In binary (two class) classification problems, it is helpful to choose k to be an odd number, as this avoids tied votes.
- It is important to review the sensitivity of the solution to different values of k.



Best Used...[3]

- When you need a simple algorithm to establish benchmark learning rules
- When memory usage of the trained model is a lesser concern
- When prediction speed of the trained model is a lesser concern

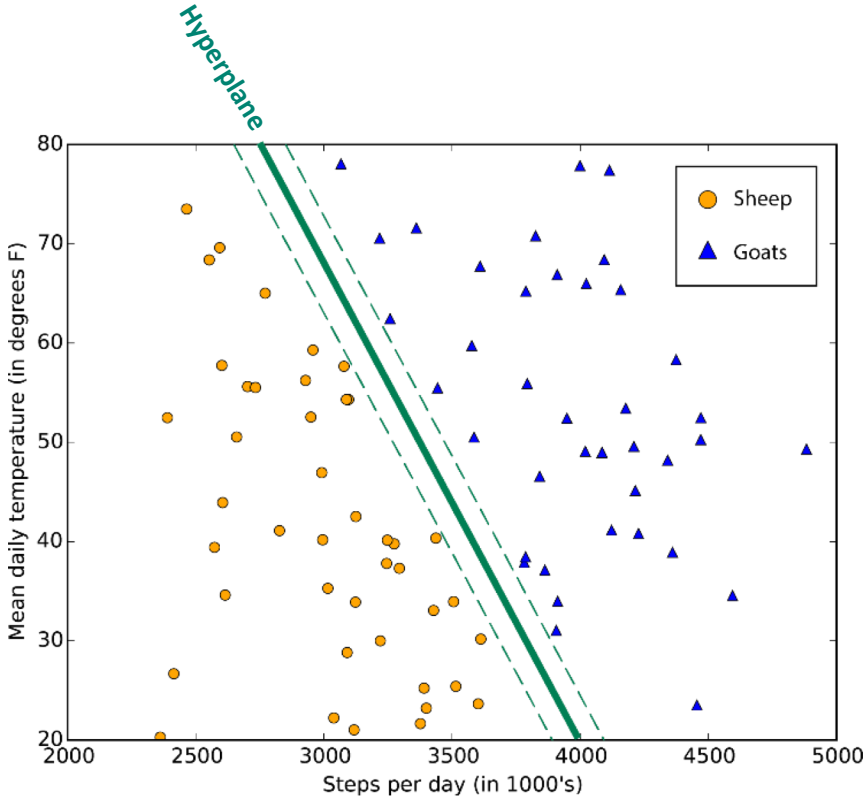
1,2 Sourece: https://open.sap.com/courses/ds1/items/4IcgeziLoUcuOfkxxLM1y8#0\
3 Source : Mathworks, Applying Supervised Learning

Support Vector Machine

:: Support Vector Machine (SVM)

SVMs are a set of supervised learning methods used for classification, regression and outliers detection

Classifies data by finding the linear decision boundary (hyperplane) that separates all data points of one class from those of the other class. The best hyperplane for an SVM is the one with the largest margin between the two classes, when the data is linearly separable. [1]



Support vector machines (SVMs) find the boundary that separates classes by as wide a margin as possible. SVMs are capable of both classification and regression.

Best Used...[2]

- For data that has exactly two classes (you can also use it for multiclass classification with a technique called error- correcting output codes)
- For high-dimensional, nonlinearly separable data
- When you need a classifier that's simple, easy to interpret, and accurate

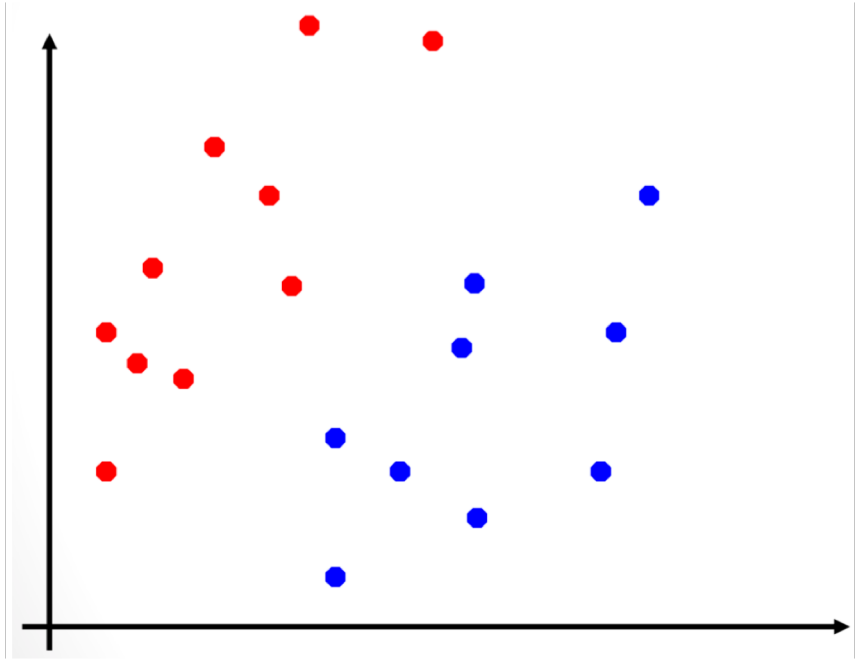
.

1 &2 Source: Mathworks, Applying Supervised Learning
Image Source: Microsoft, https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-choice

:: Support Vector Machine (SVM)

How to classify the data set in the following example ?

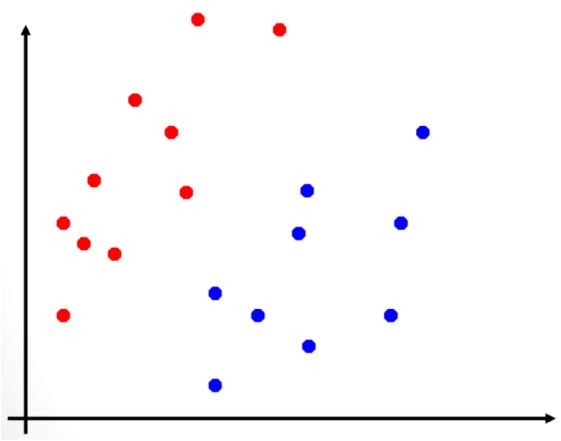
There is a two dimensional data set with two classes. How to classify it into 2 classes correctly ?



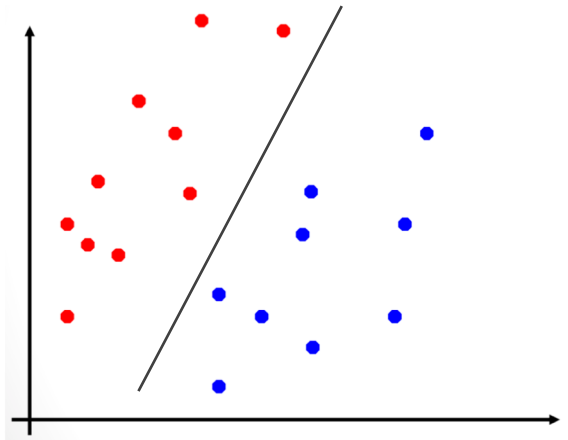
:: Support Vector Machine (SVM)

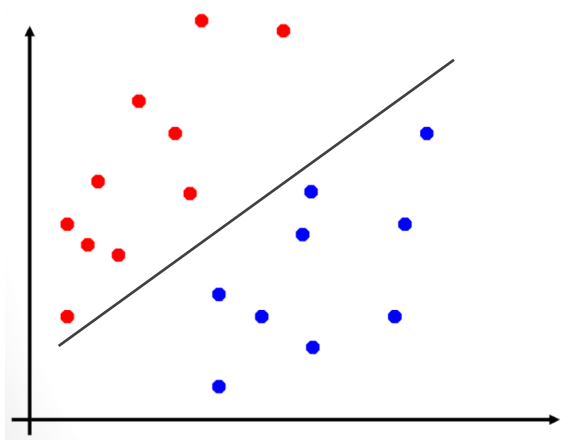
How to classify the data set in the following example ?

There is a two dimensional data set with two classes. How to classify it into 2 classes correctly ?



A two dimensional data set with two classes



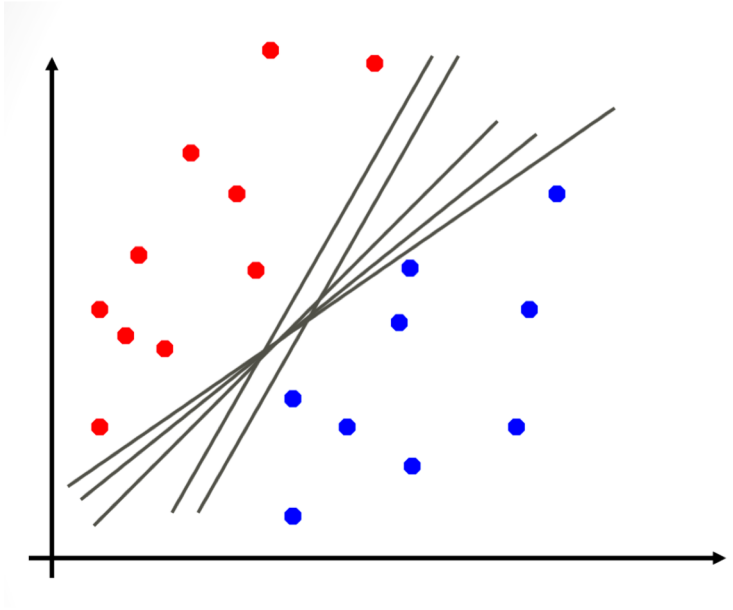


Both lines can be linear classifiers. Actually there are more lines that can be linear classifiers.

:: Support Vector Machine (SVM)

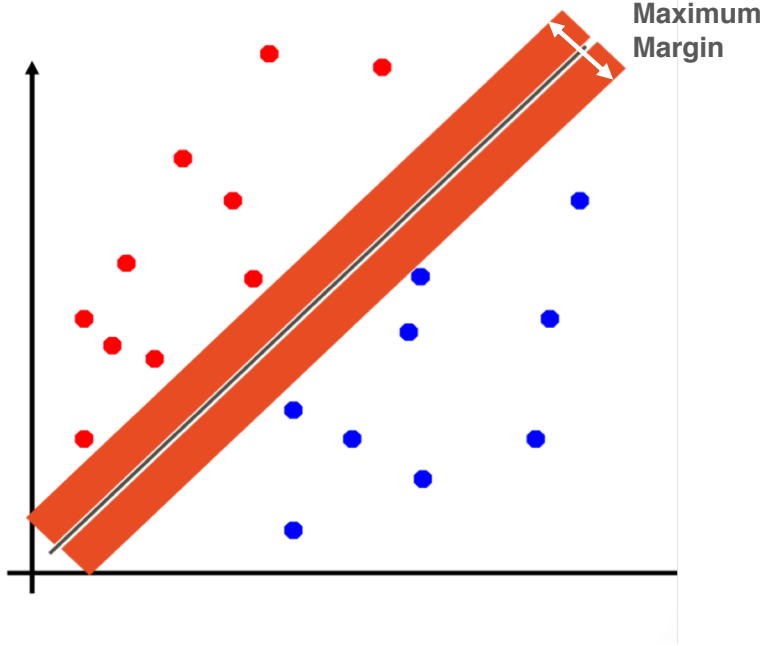
Which line is the best classifier ?

There is a two dimensional data set with two classes. How to classify it into 2 classes correctly ?



There are many lines that can be linear classifiers.

Which one is the optimal classifier ?



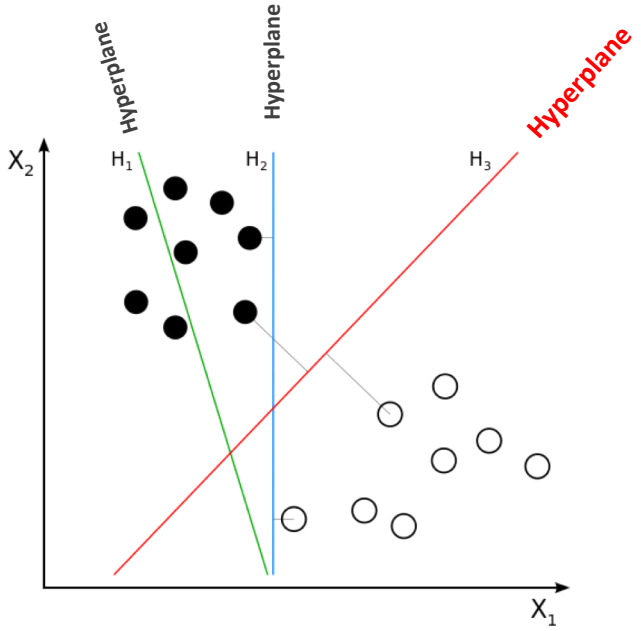
- Define the margin of a linear classifier as the width that the boundary could be increase by before hitting a data data point.[1]
- The maximum margin linear classifier is the simplest kind of SVG(called Linear SVM) [2]

1,2 source: https://www.cs.cmu.edu/~cga/ai-course/svm.pdf

:: Support Vector Machine (SVM)

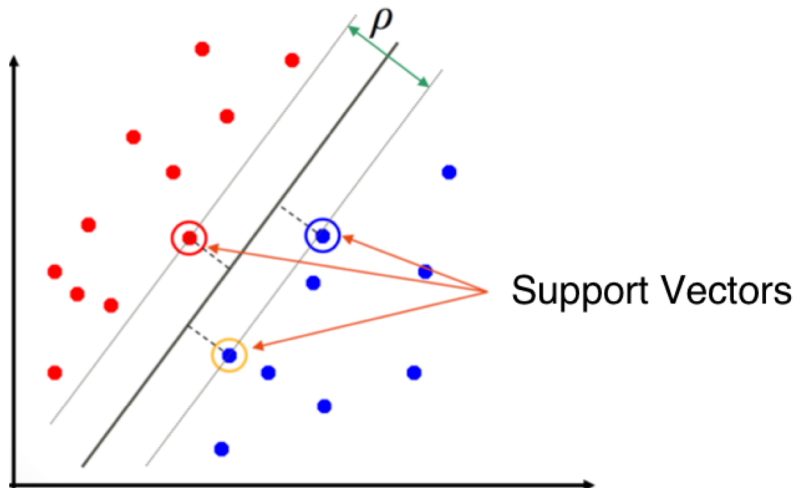
Select the hyper-plane which segregates the two classes better

a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin)



- H_1 does not separate the classes.
- H_2 does, but only with a small margin.
- **H_3 separates them with the maximum margin.**

Source: https://en.wikipedia.org/wiki/Support_vector_machine

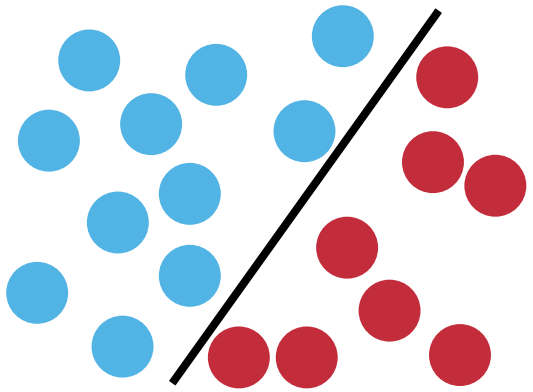


- Examples closet to the hyper-plane are ***support vectors***
- **Margin** ρ of the separator is the distance between support vectors.

:: Support Vector Machine (SVM)

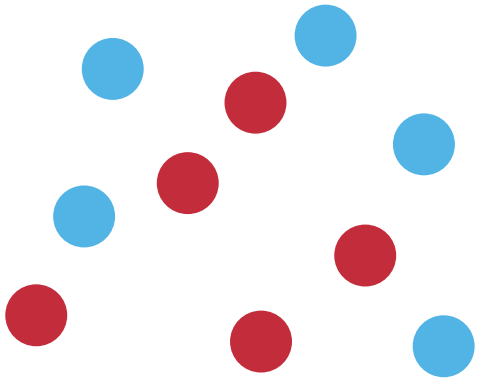
There is a new challenge !

How to classify the nonlinearly separable data by the hyperplane ?



Use Linear SVM algorithm

The linear SVM can work well on classifying this data set.



How to classify this data set ?

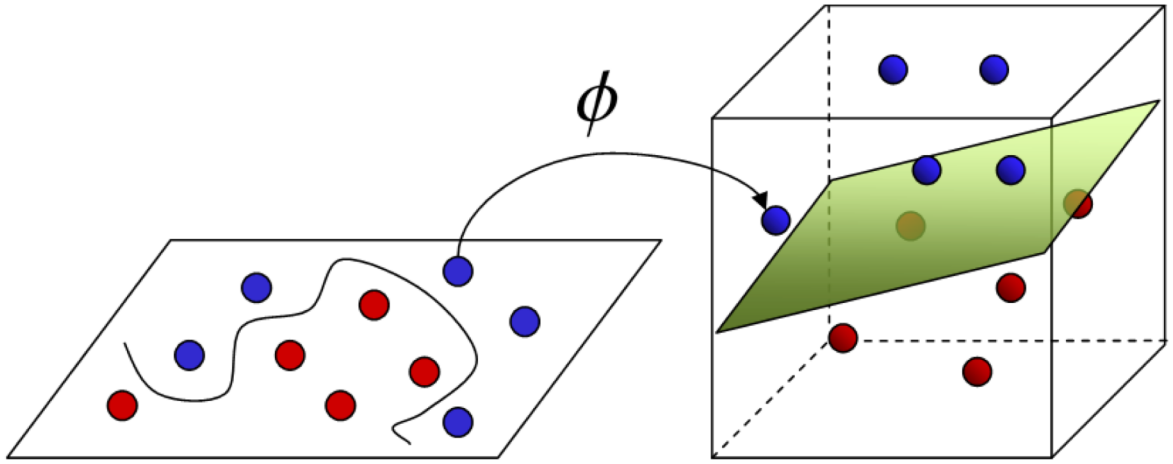
It seems that it's hard to classify this data set via the linear classifier.

Source: http://bytesizebio.net/2014/02/05/support-vector-machines-explained-well/

:: Support Vector Machine (SVM)

Create nonlinear classifiers by applying the kernel trick

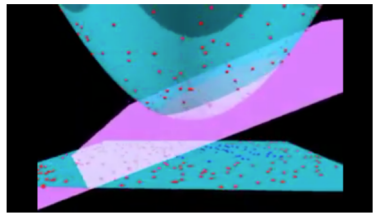
SVMs sometimes use a kernel transform to transform nonlinearly separable data into higher dimensions where a linear decision boundary can be found.



Input Space **Feature Space**

Kernel trick allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. Although the classifier is a hyperplane in the transformed feature space, it may be nonlinear in the original input space.

Cool Video (1 min)
SVM with polynomial kernel visualization



https://www.youtube.com/watch?v=3liCbRZPrZA

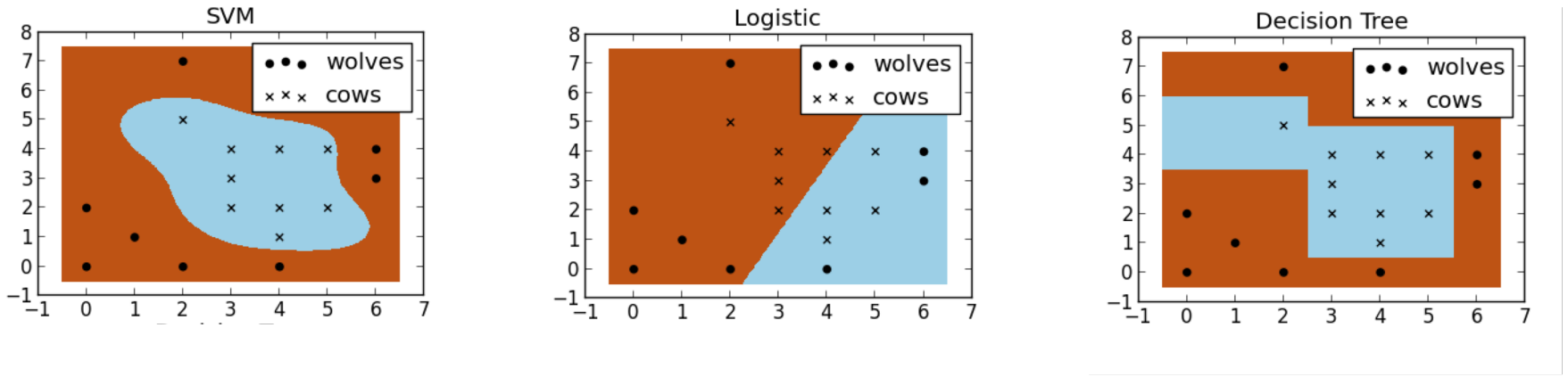
https://www.youtube.com/watch?v=9NrALgHFwTo

Image Source: http://bytesizebio.net/2014/02/05/support-vector-machines-explained-well/

:: Support Vector Machine (SVM)

Example: SVM using a non-linear kernel

If you are farmer and you need to set up a fence to protect your cows from packs of wolves, where do you build your fence ?



Where do you build your fence ?

Well if you're a really data driven farmer one way you could do it would be to build a classifier based on the position of the cows and wolves in your pasture. Trying a few different types of classifiers, we see that SVM does a great job at separating your cows from the packs of wolves. I thought these plots also do a nice job of illustrating the benefits of using a non-linear classifiers.

You can see the the logistic and decision tree models both only make use of straight lines. [1]

1 & Image source:https://www.kdnuggets.com/2017/02/yhat-support-vector-machine.html

:: Support Vector Machine (SVM)

Strengths of SVM

Advantages

- The major strengths of SVM are the training is relatively easy. No local optimal, unlike in neural networks.[1]
- Non-linear SVMs use a non-linear kernel. Non-linear SVM means that the boundary that the algorithm calculates doesn't have to be a straight line. The benefit is that you can capture much more complex relationships between your data points without having to perform difficult transformations. The downside is that the training time is much longer, because it is much more computationally intensive. [2]
- SVMs have a regularization parameter, which can help avoid over-fitting. [3]
- Effective in high dimensional spaces.[4]
- Still effective in cases where number of dimensions is greater than the number of samples.[5]
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.[6]

Source: https://... 00f11/resources/jakkula.pdf
2,3 Source: ...lcgeziLoUcuOfkxxLM1y8#0
4,5,6 http://...learn.org/stable/modules/svm.html
Image: Designed by Asierromero / Freepik

:: Support Vector Machine (SVM)

Weakness of SVM

Disadvantage

- For classification, the SVM is only directly applicable for two-class tasks. Therefore, algorithms that reduce the multi-class task to several binary problems have to be applied. Well, many SVM packages already have built-in multi-class classification functionality. Otherwise, use one-versus-the-rest approach to build a multiclass classifier
- Unlike Logistic Regression classifiers, SVMs do not directly provide probability estimates. In many classification problems you actually want the probability of class membership
- Parameters of a solved model are difficult to interpret.
- Long training time on large data sets
- Choosing a “good” kernel function can be tricky.

1 Sourece: https://open.sap.com/courses/ds1/items/4lcgeziLoUcuOfkxxLM1y8#0
2 source:https://www.kdnuggets.com/2017/02/yhat-support-vector-machine.html
Image: Designed by Asierromero / Freepik

Linear Regression

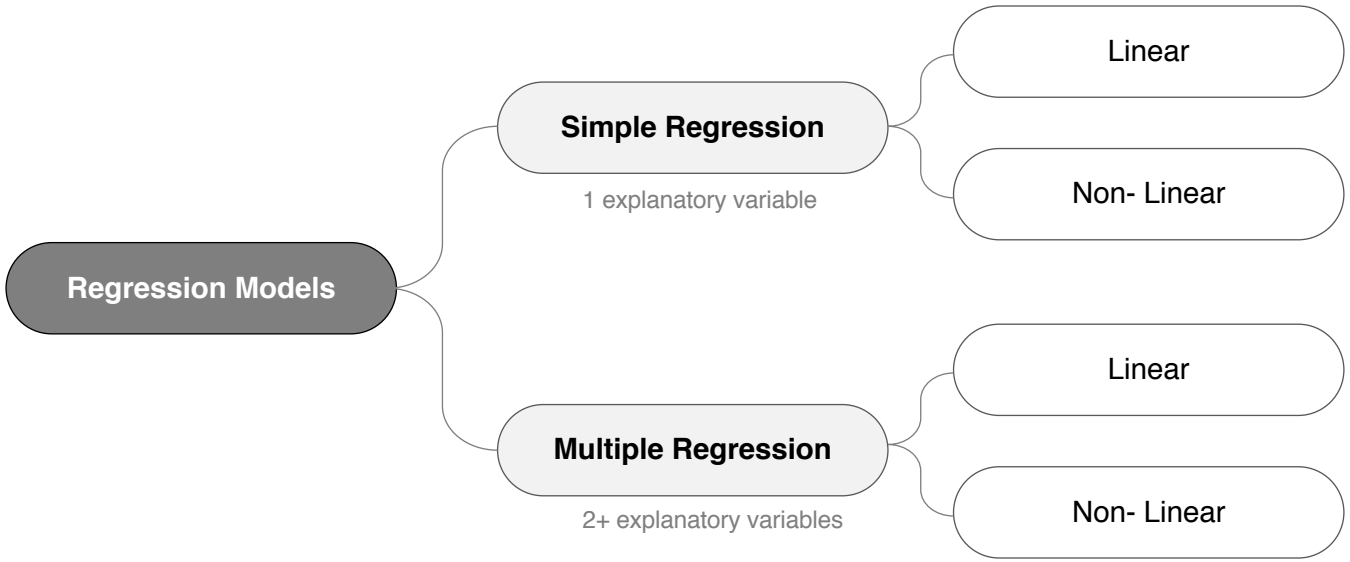
:: Linear Regression

Regression Analysis

ML models for regression problems predict a numeric value. Examples of Regression Problems

- What will the temperature be in Seattle tomorrow?
- For this product, how many units will sell?
- What price will this house sell for?

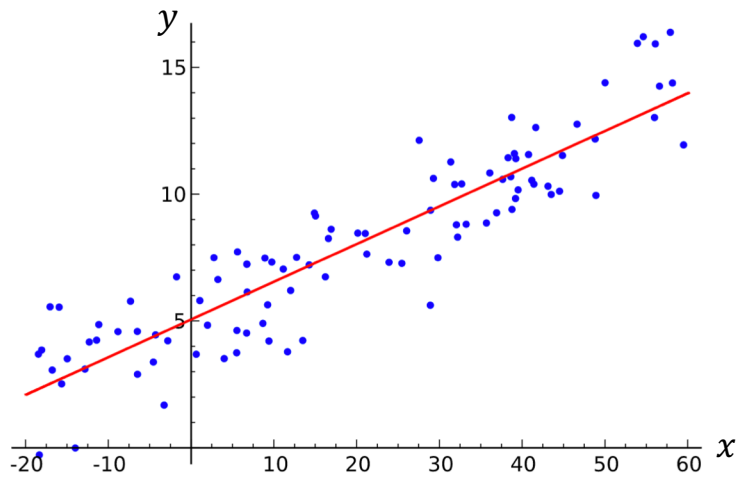
Type of regression models:



:: Linear Regression

What's Linear regression ?

Linear regression is a linear approach for modeling the relationship between a dependent variable y and explanatory variables (or independent variables) denoted x.[1]



- It's a supervised algorithm that learns from a set of training samples.
- Each training sample has one or more input values and a single output value.
- The algorithm learns the line, plane or hyper-plane that best fits the training samples.
- Use the learned line, plane or hyper-plane to predict the output value for any input sample.

- Because linear regression models are simple to interpret and easy to train, they are often the first model to be fitted to a new dataset.[2]
- Linear regression can be used as a baseline for evaluating other, more complex, regression models[3]

1 source: https://en.wikipedia.org/wiki/Linear_regression
2,3 Sourcxe: Mathworks, Applying Supervised Learning

:: Linear Regression

What's Linear regression used for ?

Linear regression has many practical uses. Most applications fall into one of the following two broad categories: [1]

1. Making Prediction

Based on the relationship we learn from the sample data, the linear regression model can be used to to forecast unobserved value.

In other words "Linear Regression" is a method to predict dependent variable y based on values of independent variables x_1 , x_2..., x_p .

Examples: What's the sales value over the next year ?

2. Establishing if there is a relationship between variables

Given a variable y and a number of variables x_1 , x_2..., x_p that may be related to y, linear regression analysis can be applied to quantify the strength of the relationship between y and the x_j , to assess which x_j may have no relationship with y at all, and to identify which subsets of the x_j contain redundant information about y. [2]

More specifically, establish if there is a **statistically significant relationship** between two variables. For example, Income might have significant relationship with education level.

1,2 source: https://en.wikipedia.org/wiki/Linear_regression

:: Linear Regression

Type of Linear Regression

simple linear regression

one explanatory variable

Multiple linear regression

more than one explanatory variable

:: Linear Regression

Type of Linear Regression

simple linear regression

one explanatory variable

Multiple linear regression

more than one explanatory variable

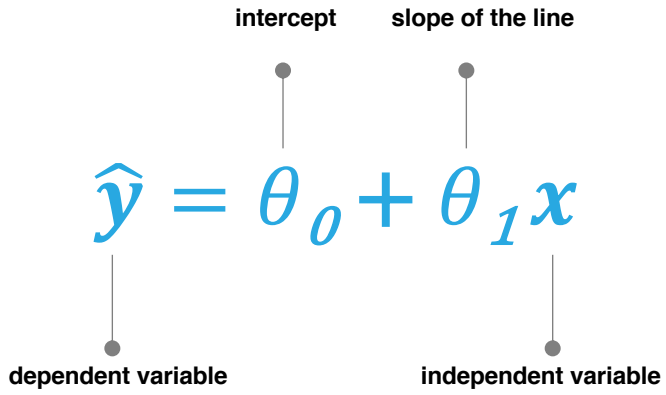
:: Linear Regression

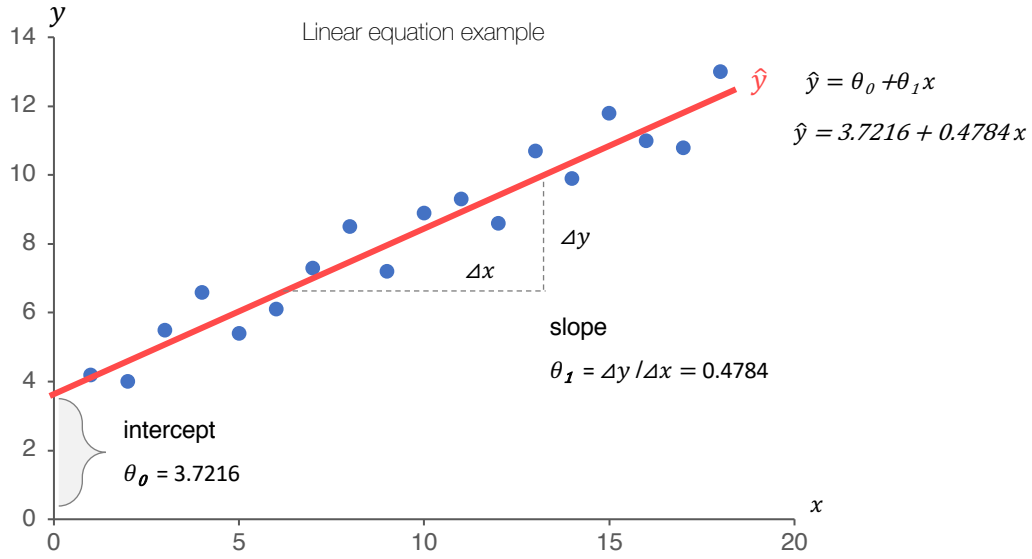
Let's take a look at Simple Linear regression

Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables: [1]

- One variable, denoted x, is regarded as the **predictor**, **explanatory**, or **independent** variable.
- The other variable, denoted y, is regarded as the **response**, **outcome**, or **dependent** variable.

Simple Linear Regression is termed as simple because there is only independent variable. The relationship between the input x and outcome y can be represented in a form of following equation:





The slope of 0.4784 means that each increase of one unit in X, we predict the average of Y to increase by an estimated 0.4784 units.

1 Source: https://onlinecourses.science.psu.edu/stat501/node/251

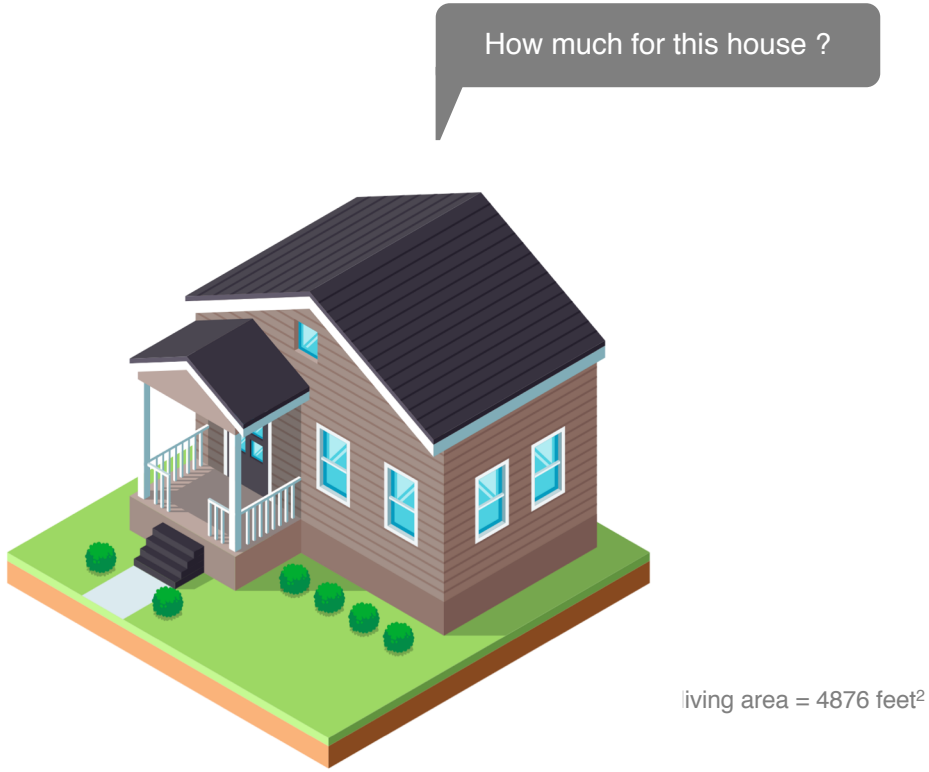
:: Linear Regression

Example: Predict house's price using linear regression

Suppose we have a dataset giving the living areas and prices of 21,613 houses from House Sales in King County, USA. Given data like this, we can learn to predict the prices of other houses in King County.

Living Area (Feet^2)	Price ($)
1180	221,900
2570	538,000
770	180,000
1960	604,000
1680	510,000
5420	1,225,000
1715	257,500
1060	291,850
1780	229,500
1890	323,000
3560	662,500
1160	468,000
1430	310,000
1370	400,000
1810	530,000
...	...
x	y

Data set



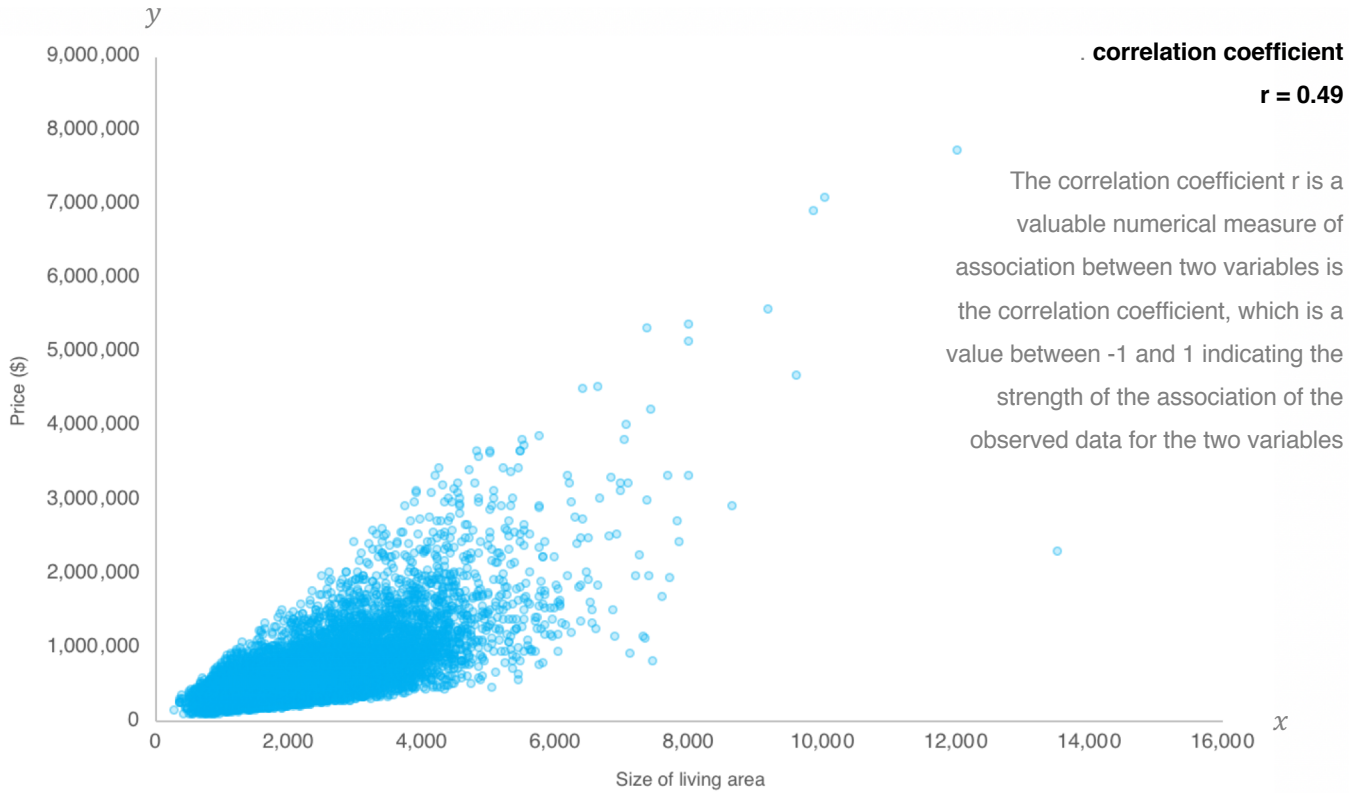
Picture of house: Designed by Freepik

:: Linear Regression

Example: Predict house's price using linear regression

Before attempting to fit a linear model to observed data, a modeler should first determine whether or not there is a relationship between the variables of interest. A scatterplot can be a helpful tool in determining the strength of the relationship between two variables [1]. In this example, the price has positive correlation with the size (r=0.49)

Living Area (Feet^2)	Price ($)
1180	221,900
2570	538,000
770	180,000
1960	604,000
1680	510,000
5420	1,225,000
1715	257,500
1060	291,850
1780	229,500
1890	323,000
3560	662,500
1160	468,000
1430	310,000
1370	400,000
1810	530,000
...	...
x	y



. **correlation coefficient**

r = 0.49

The correlation coefficient r is a valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables

1 source: http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

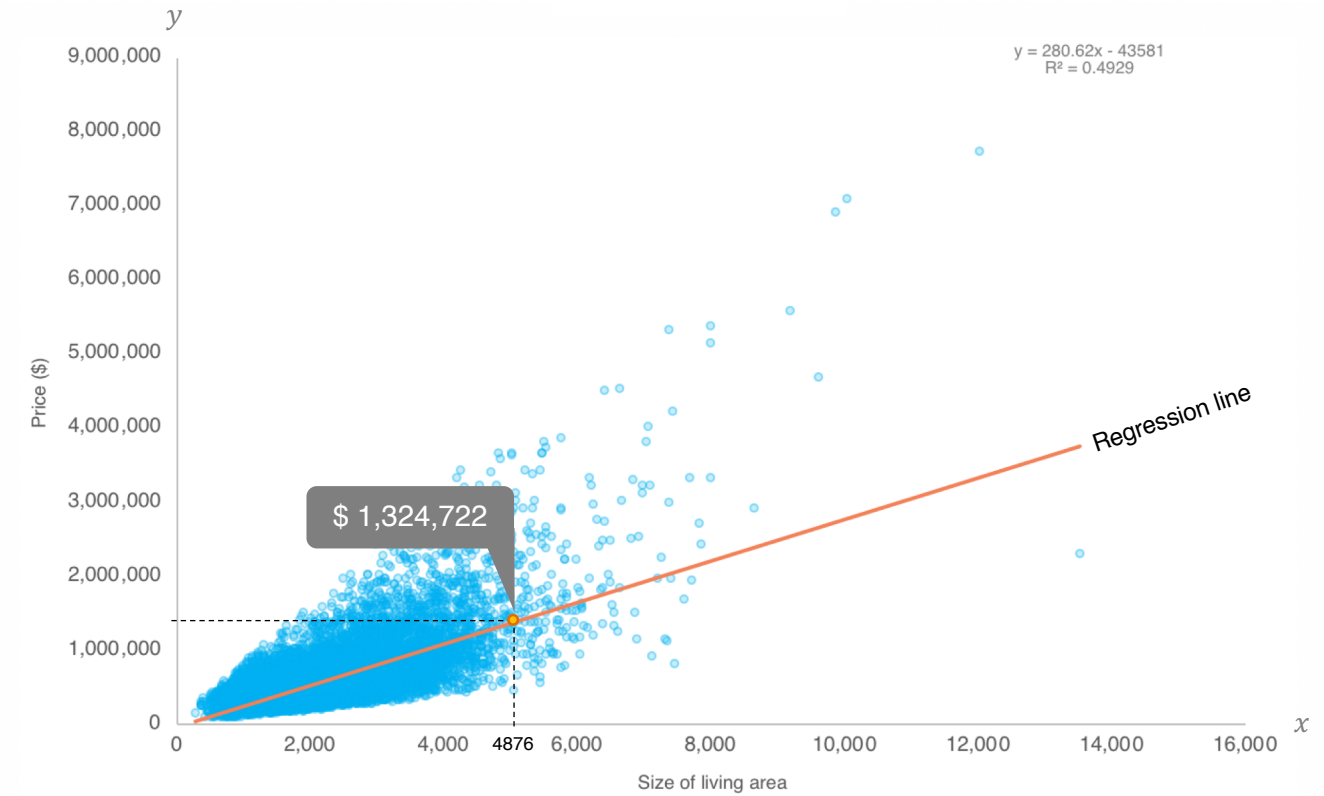
:: Linear Regression

Example: Predict house's price using linear regression

Then we can identify best-fitting relationship (line) between the variables. The regression line is the "best-fit" line and has the formula $y = \theta_0 + \theta_1 x$.Thus, given a value of X, we can predict a value of Y.

Size of living area = 4876 feet^2

Based on this regression line, we can predict the price for a house with a certain size of living area



Picture of house: Designed by Freepik

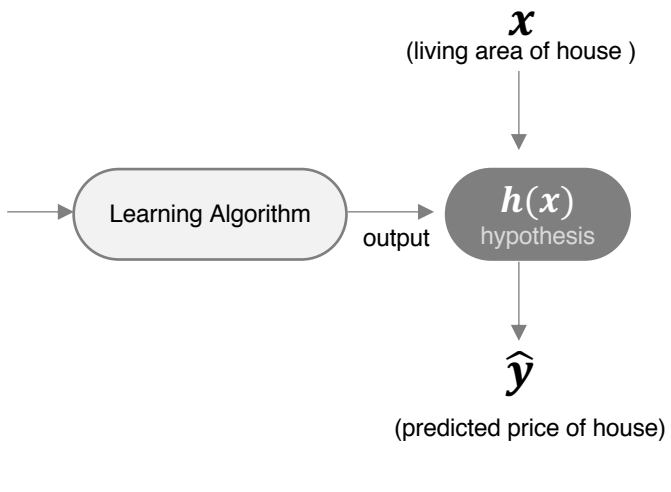
:: Linear Regression

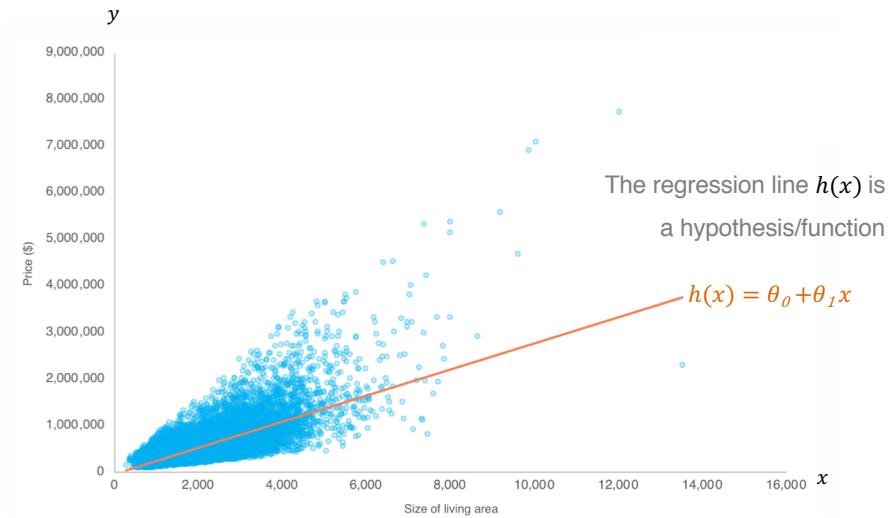
Actually linear regression model guesses a hypothesis function h(x) from training set

Given a training set, the model learn a function $h(x)$ which can be used for predicting the new/unseen data.

Training set

x Living Area (Feet^2)	y Price ($)
1180	221,900
2570	538,000
770	180,000
1960	604,000
1680	510,000
5420	1,225,000
1715	257,500
1060	291,850
1780	229,500
1890	323,000
3560	662,500
1160	468,000
1430	310,000
1370	400,000
1810	530,000
...	...





x is Input (feature)

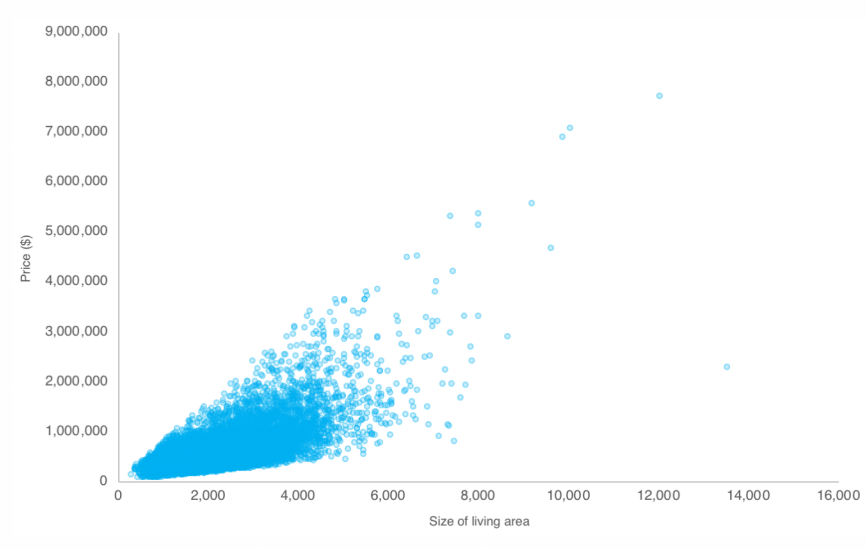
y is Output (target variable)

Diagram reference : http://cs229.stanford.edu/notes/cs229-notes1.pdf

:: Linear Regression

The goal of linear regression is to find the best fit line

The goal is to find estimated value for the parameters θ_0 and θ_1 which would provide the “best” fit for the data points within the training set.



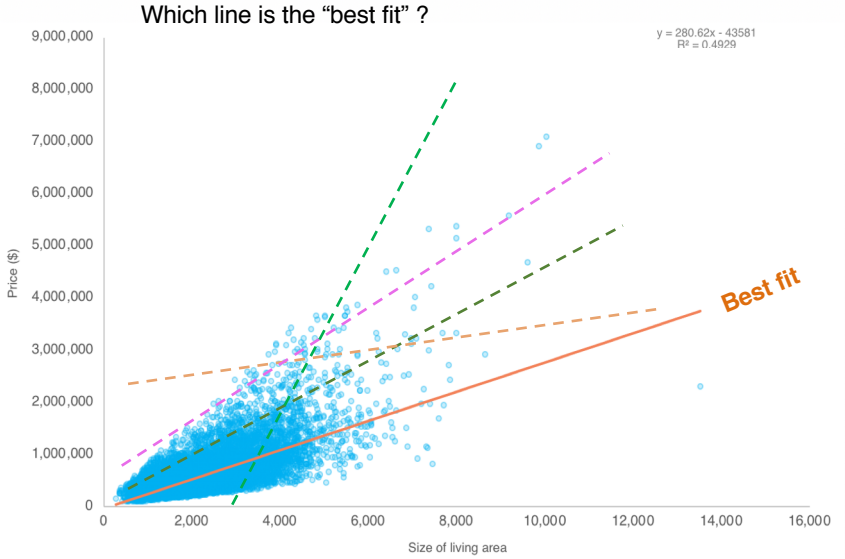
$h(x) = \theta_0 + \theta_1 x$

Parameters

θ_0, θ_1

Different parameters θ_0 and θ_1 will result in different lines

Which line is the “best fit” ?

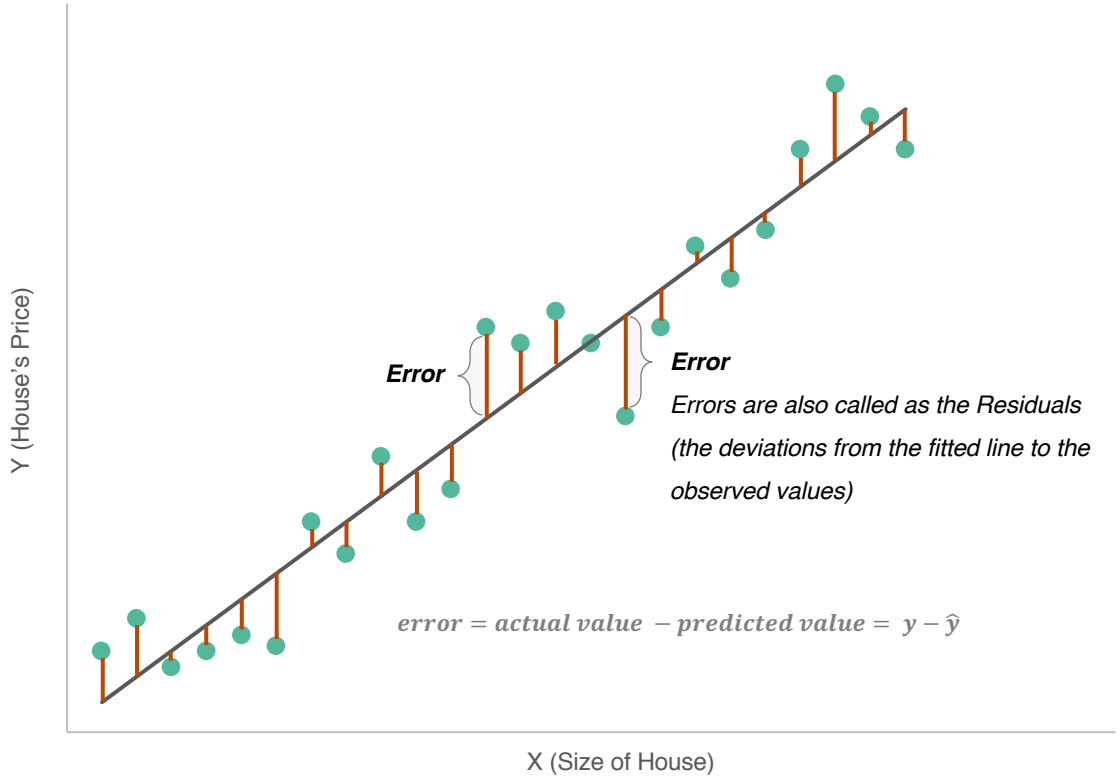


Best fit line : θ_0 = 43581 , θ_1 = 280.62

:: Linear Regression

How do we know which line is the “best fit” ?

The good line is one that minimizes the sum of the “squared differences” between the points and the regression line. Usually some data points will deviates from the line somehow (if a point lies on the fitted line exactly, then its vertical deviation is 0).



Least Squares Method

The optimization goal is to minimize the sum of “squared error” (least squares). Because the deviations are first squared, then summed, there are no cancellations between positive and negative values.

Goal: to minimize the sum of squared error

Sum of Squared Errors $= \sum_{i=1}^{m}(y_i - \hat{y}_i)^2$

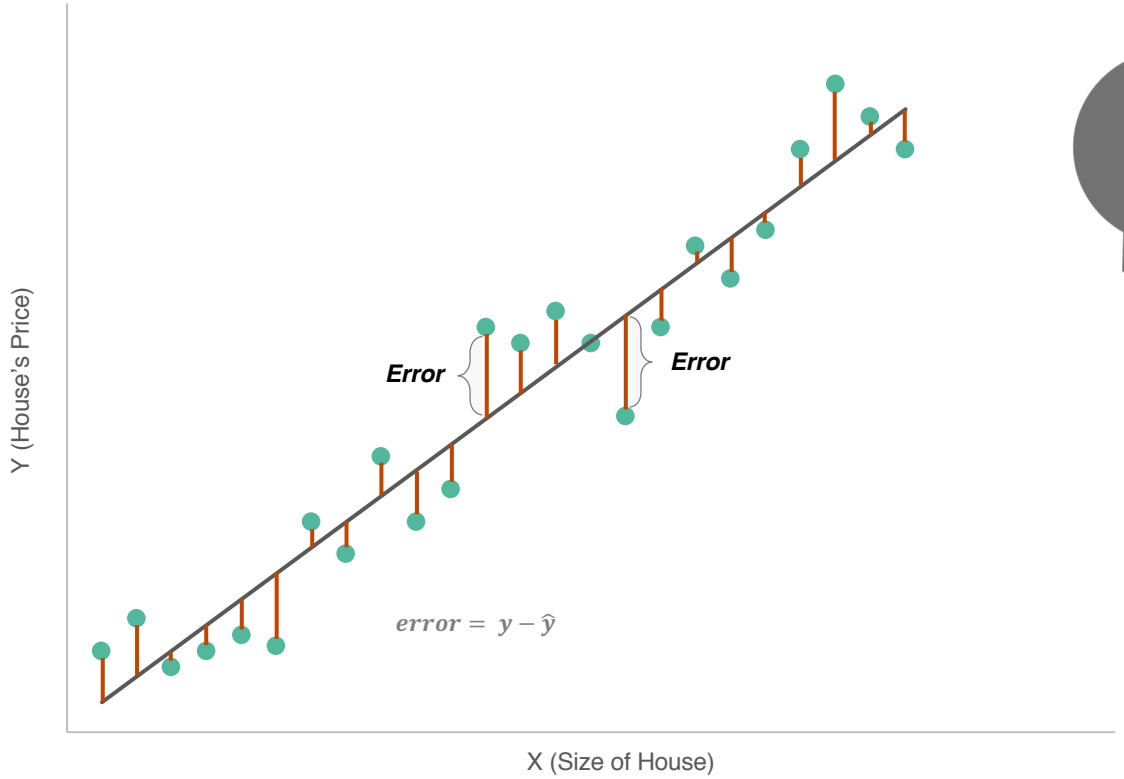
where $\hat{y} = \theta_0 + \theta_1 x$
m is number of training instances

The smaller the sum of squared differences, the better the fit of the line to the data.

:: Linear Regression

How do we come up with “best” parameter value that corresponds to a good fit to the data?

The idea is to choose the appropriate parameter θ_0 and θ_1 so that the predicted value $\hat{y}$ is close to y for training example (x, y). In other words, the sum of squared error should be least.



$$h(x) = \theta_0 + \theta_1 x$$

How to find the appropriate parameter θ_0 and θ_1 in order to minimize the error – i.e. cost function $J(\theta_0, \theta_1)$?

To **minimize** $$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

Cost function
(also known as Loss Function)

- m is number of training instances
- $\hat{y}_i$ (y hat) is the predicted value
- y_i is the actual value

Video: Cost Function Intuition #1 by Andrew Ng (11mins)

https://www.youtube.com/watch?v=yR2ipCoFvNo

:: Linear Regression

The computation methods for estimating the parameters

There are a couple of approaches to find the value of parameter θ that minimizes the cost function $J(\theta)$

.

The approaches to solving $\theta^* = \underset{\theta}{argmin}\ J(\theta)$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- **Normal Equation (Closed Form)**

  It's a method to solve for θ analytically.

  Using a direct "closed-form" equation that directly computes the model parameters that best fit the model to the training set (i.e., the model parameters that minimize the cost function over the training set). [1]

  It's suitable for small feature set (e.g. < 1000 features).

- **Gradient Descent**

  Using an iterative optimization approach, called Gradient Descent (GD), that gradually tweaks the model parameters to minimize the cost function over the training set, eventually converging to the same set of parameters as the first method. [2]

  Gradient Descent is better choice than Normal Equation when there are a large number of features, or too many training instances to fit in memory.

1, 2 Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow

Image: Designed by Asierromero / Freepik

:: Linear Regression

The computation methods for estimating the parameters

There are a couple of approaches to find the value of parameter θ that minimizes the cost function $J(\theta)$

.

The approaches to solving $\theta^* = \underset{\theta}{argmin}\ J(\theta)$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- **Normal Equation (Closed Form)**

  It's a method to solve for θ analytically.

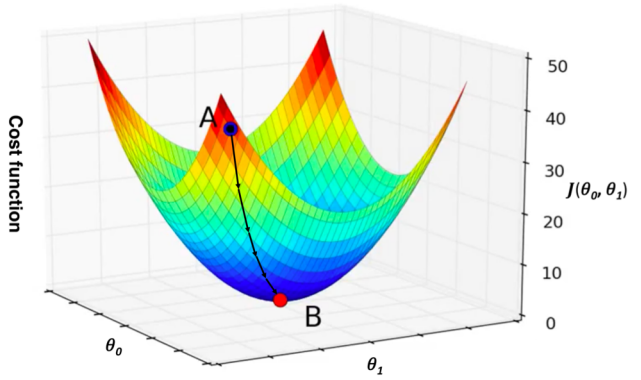
  Using a direct "closed-form" equation that directly computes the model parameters that best fit the model to the training set (i.e., the model parameters that minimize the cost function over the training set). [1]

  It's suitable for small feature set (e.g. < 1000 features).

- **Gradient Descent**

  Using an iterative optimization approach, called Gradient Descent (GD), that gradually tweaks the model parameters to minimize the cost function over the training set, eventually converging to the same set of parameters as the first method. [2]

  Gradient Descent is better choice than Normal Equation when there are a large number of features, or too many training instances to fit in memory.



1, 2 Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow

Picture: Designed by Asier_Relampagoestudio / Freepik

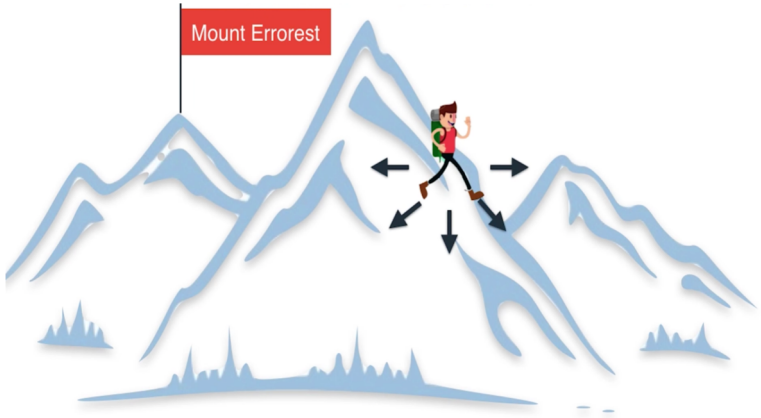
:: Linear Regression

Gradient Descent

Gradient Descent is one of the most popular optimization algorithms used in Machine Learning and Deep Learning. Many machine learning algorithms such as linear regression, logistic regression, neural networks use Gradient Descent.

If you're blindfolded Hiker,

how to get to the lowest place ?



We can illustrate Gradient Descent optimization as a process that a blindfolded hiker wants to go down a mountain as soon as possible.

Suppose you are at the top of a mountain, and you have to go to the valley which is at the lowest point of the mountain. But now you are blindfolded and you can only feel the slope of the ground below your feet. So, what approach will you take to reach the lake?

A good strategy to get to the bottom of the valley quickly is to go downhill in the direction of the steepest slope. This is exactly what Gradient Descent does: it measures the local gradient of the error function with regards to the parameter vector θ, and it goes in the direction of descending gradient. Once the gradient is zero, you have reached a minimum[1].

Video: simple introduction of Gradient Descent (2 mins)
https://www.youtube.com/watch?time_continue=62&v=BEC0uH1fuGU

1 Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow
Picture: https://www.youtube.com/watch?time_continue=62&v=BEC0uH1fuGU

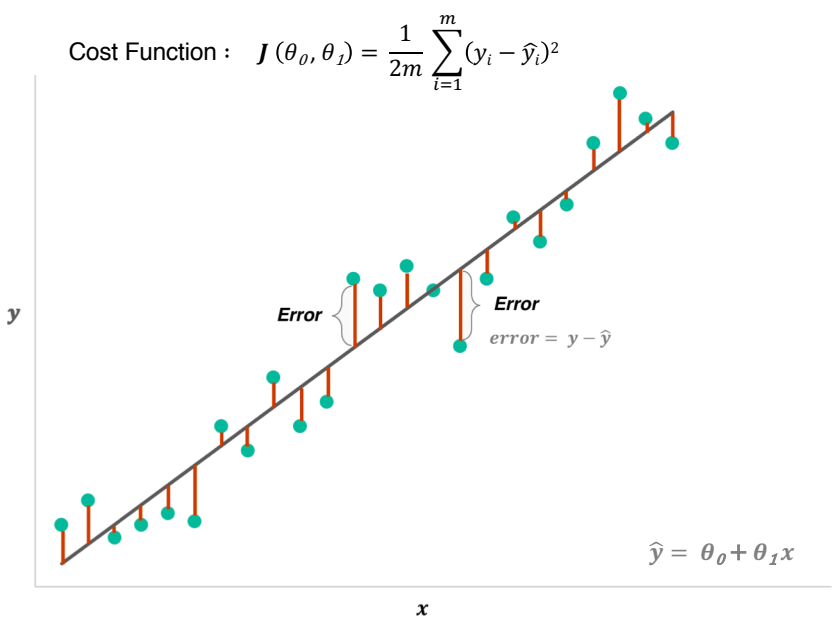
:: Linear Regression

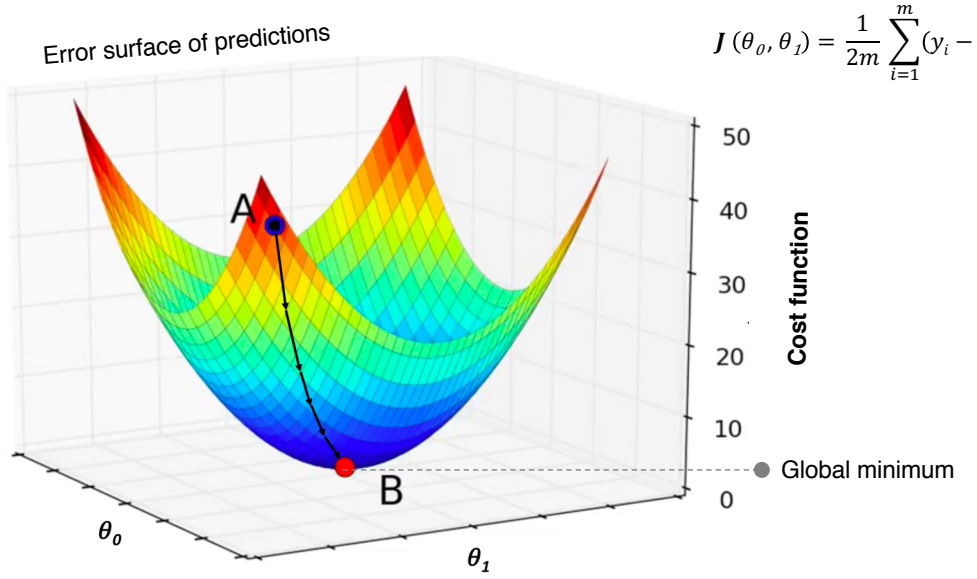
Gradient descent is an iterative algorithm to find a minimum of a function

The goal of gradient descent is to start on a random point on this error surface , and find out the best parameters (θ_0 , θ_1) which yields the minimum value of the cost function. In other words, the best parameters (θ_0 , θ_1) corresponds to the the lowest point on the error surface.

.

The cost function $J\ (\theta_0, \theta_1)$ can be visualized as a 3D plot





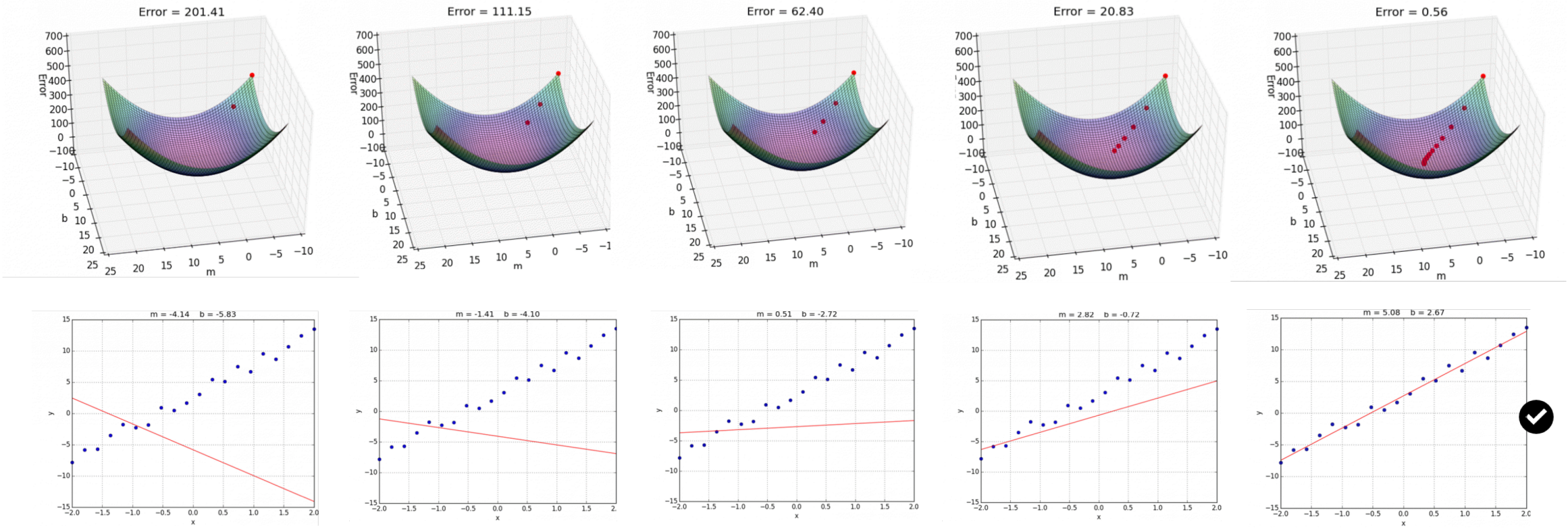
The combination of different parameters θ_0 , θ_1 leads to different error (cost function).

The error surface of the predictions corresponds to the different value of parameters.

:: Linear Regression

Example : find the best-fit line through Gradient Descent algorithm

Iteratively find the minimum of cost function



Animation: a visual representation of gradient descent in action

https://alykhantejani.github.io/images/gradient_descent_line_graph.gif

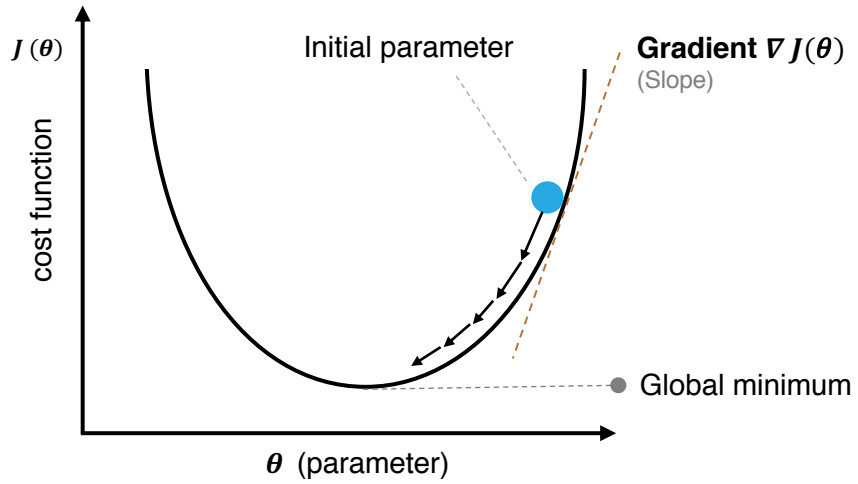
Video: Cost Function Intuition #2 | Andrew Ng (8mins)

https://youtu.be/0kns1gXLYg4?t=2m8s

:: Linear Regression

What's gradient ?

Gradient is another word for "slope". A gradient is the slope of a function at a specific point. The gradient of loss function/cost function is equal to the derivative (slope) of the curve.



$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

How to calculate the error gradient with respect to the parameter ?

$$\nabla_\theta J(\theta) = \frac{d}{d\theta}J(\theta)$$

$\frac{d}{d\theta}J(\theta)$ is the **derivative** of $J(\theta)$ —i.e. the slope of a tangent line that touches the $J(\theta)$ at given θ.[1]

What's derivative ? See next slide

1 Source: https://hackernoon.com/machine-learning-bit-by-bit-univariate-gradient-descent-9155731a9e30

:: Linear Regression

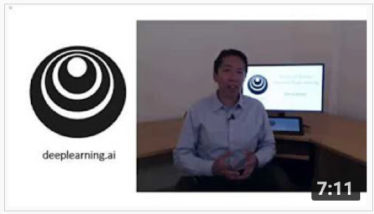
What's derivative & partial derivative?

It's important to understand the basic concept of derivative. It's strongly recommended to watch the following videos if you have already forgotten those concept.

Hold on a second! I have forgotten calculus.

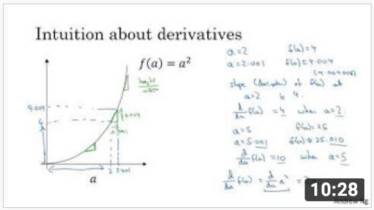
I can't understand the **partial derivative**

$$\frac{\partial J(\theta)}{\partial \theta}$$



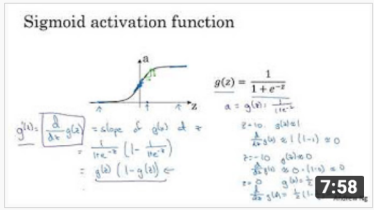
1 Derative

www.youtube.com/watch?v=GzphoJOVEcE



2 More Derivative Examples

www.youtube.com/watch?v=5H7M5Vd3-pk



3 Derivatives of Activation Functions

www.youtube.com/watch?v=-CG_t8e9Bac

4 Partial derivatives, introduction

www.youtube.com/watch?v=AXqhWeUEtQU&t=16s

Picture: Designed by Asierromero / Freepik

:: Linear Regression

How does Gradient descent work ?

When there are multiple parameters, the gradient is a vector of partial derivatives with respect to the parameters. For the sake of simplicity, consider the cost function $J(\theta)$ that depends on only one parameter θ.

Gradient Descent algorithm

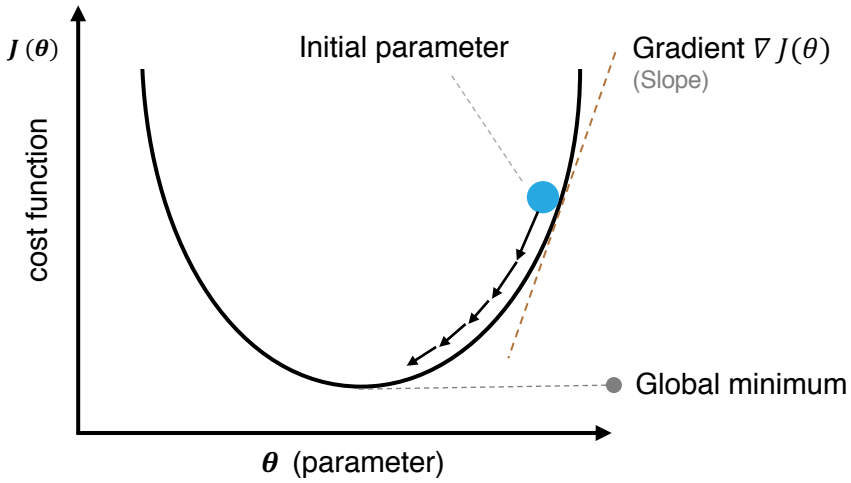
The parameter are iteratively updated in the following equation:

Learning rate (Step size)

$$\theta_{new} = \theta_{old} - \alpha \cdot \nabla_\theta J(\theta)$$

Gradient $\nabla_\theta J(\theta) = \frac{d}{d\theta} J(\theta)$

1. Pick a value for the learning rate α
2. Start with a random point θ
3. Calculate the gradient $\nabla J(\theta)$ at the point θ. Follow the opposite direction of gradient to get new parameter θ_{new}
4. Repeat until the cost function converges to the minimum



In this example, initially the slope is large and positive. So, in the update equation, θ is reduced. As θ keeps getting reduced, notice that the gradient also reduces, and hence the updates become smaller and smaller and eventually, it converges to the minimum.[1]

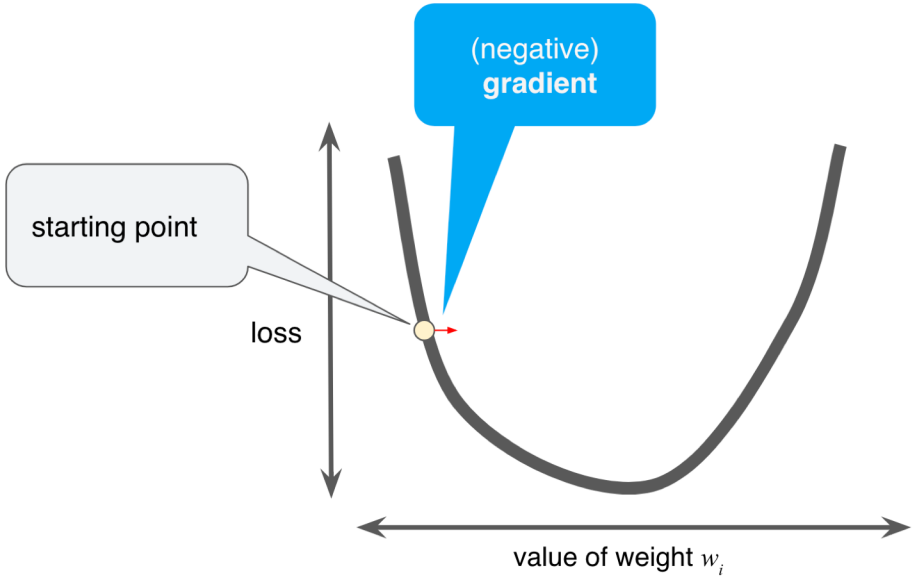
1 Source: https://hackernoon.com/dl03-gradient-descent-719aff91c7d6

:: Linear Regression

How does Gradient descent work ? - continued

The gradient of loss is equal to the derivative (slope) of the curve. Note that a gradient is a vector, so it has both of the following characteristics: [1]

- a direction
- a magnitude



The gradient always points in the direction of steepest increase in the loss function.

The gradient descent algorithm takes a step **in the direction of the negative gradient** in order to reduce loss as quickly as possible.[2]

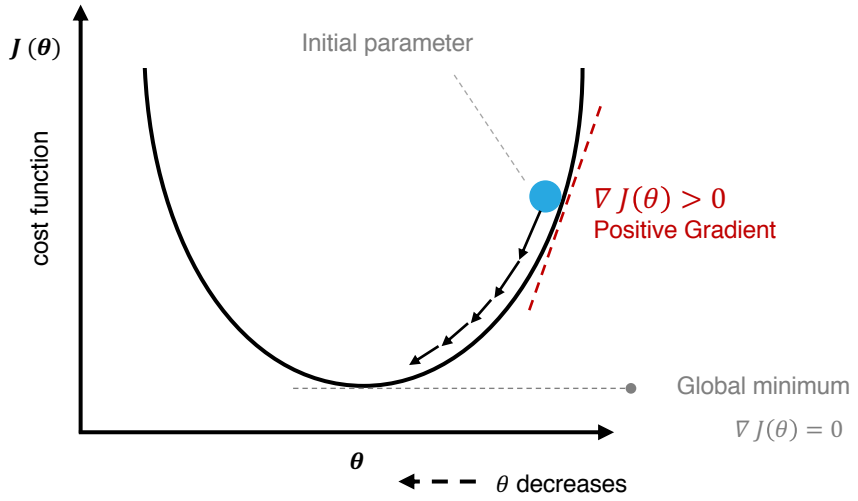
1 ,2 & picture source: https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent

:: Linear Regression

How does Gradient descent work ? - continued

For the sake of simplicity, consider the cost function $J(\theta)$ that depends on only one parameter θ.

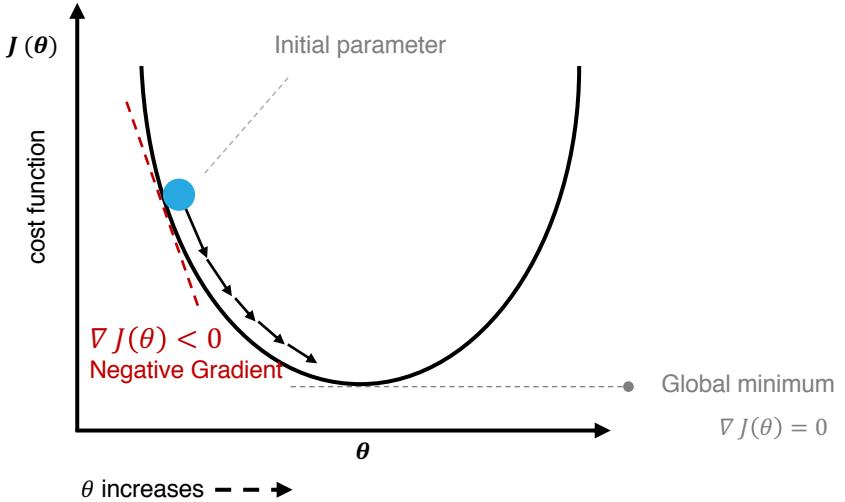
If the gradient > 0
(positive derivative)



$$\theta_{new} = \theta_{old} - \alpha \cdot \nabla_\theta J(\theta)$$

- Because the gradient is positive value , the parameter update will be negative and will reduce the current values of θ to converge to the minimum

If the gradient < 0
(negative derivative)



$$\theta_{new} = \theta_{old} - \alpha \cdot \nabla_\theta J(\theta)$$

- Because the gradient is negative value , the parameter update will be positive and will increase the current values of θ to converge to the minimum

1 Source: https://hackernoon.com/dl03-gradient-descent-719aff91c7d6

:: Linear Regression

How does Gradient descent work ? - continued

More generally, the cost function depends on more than one parameters. In Simple Linear Regression, there are 2 parameters θ_0, θ_1

Gradient Descent algorithm

The parameter are iteratively updated in the following equation:

repeat until convergence {

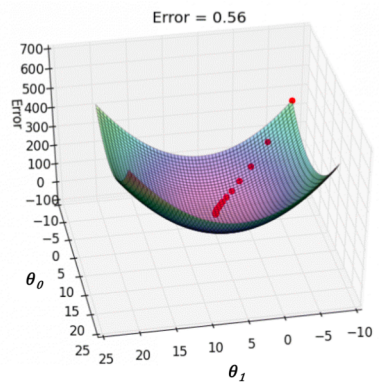
// Simultaneously update θ_0, θ_1

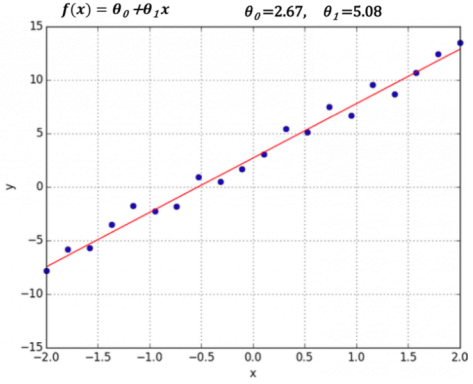
$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_1, \theta_1)$$

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

}

- Start with some random value θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta)$ until we hopefully end up at a minimum





Youtube Video: Gradient Descent by Andrew Ng (11 mins)

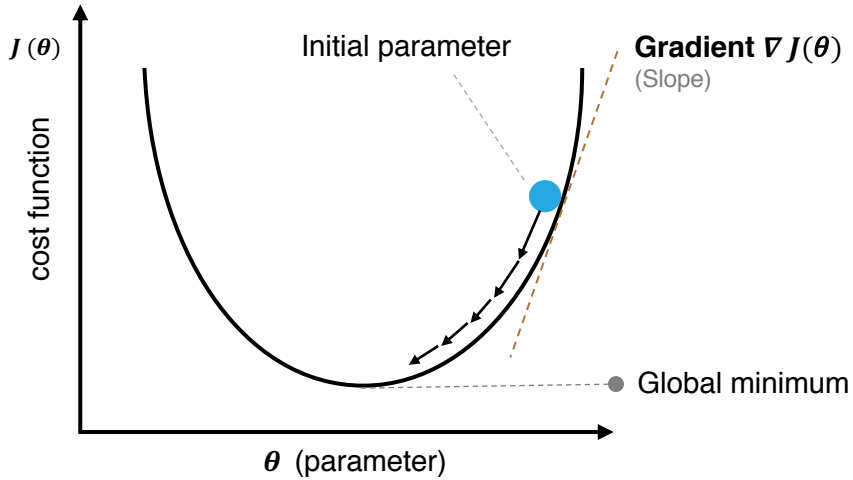
https://youtu.be/F6GSRDoB-Cg

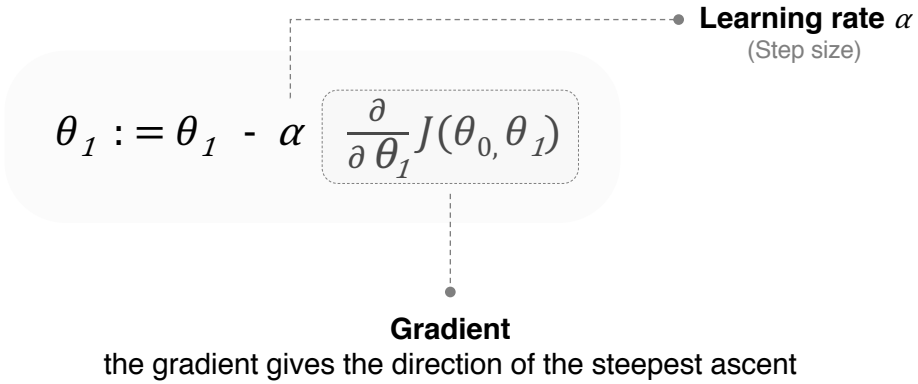
1 Source: https://hackernoon.com/dl03-gradient-descent-719aff91c7d6

:: Linear Regression

What's Learning rate ?

An important parameter in Gradient Descent is the size of the steps, determined by the *learning rate* hyper-parameter.





The gradient vector has both a direction and a magnitude. Gradient descent algorithms multiply the gradient by a scalar known as the **learning rate** (also sometimes called **step size**) to determine the next point.

For example, if the gradient magnitude is 2.5 and the learning rate is 0.01, then the gradient descent algorithm will pick the next point 0.025 away from the previous point.[1]

1 Source: https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate

:: Linear Regression

Learning rate is a hyperparameter

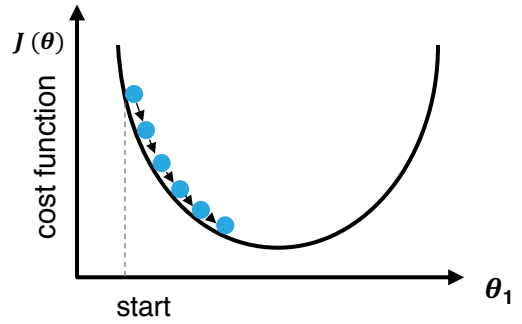
Hyperparameters are the knobs that programmers tweak in machine learning algorithms. Most machine learning programmers spend a fair amount of time tuning the learning rate.[1]

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

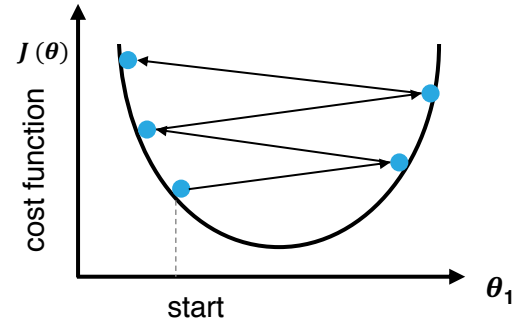
Learning rate α
(Step size)

Gradient
the gradient gives the direction of the steepest ascent

- **If the learning rate is too small**
  gradient descent would take long time to converge and can be very slow



- **If the learning rate is too large**
  gradient descent can overshoot the minimum. You might jump across the valley and end up on the other side, possibly even higher up than you were before[2]. So the algorithm may fail to converge, or even diverge.

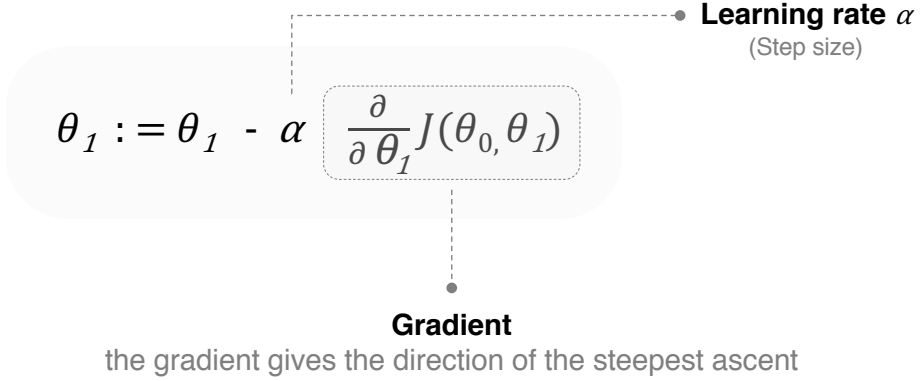


1, Source: https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate
2 Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow

:: Linear Regression

How to choose learning rate ?

Choosing a suitable value for learning rate is a sort of art. So try different values for learning rate and plot the the cost function. You can see how it is performing.



- If α is too small: slow convergence
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

The most commonly used rates are : ***0.001, 0.003, 0.01, 0.03, 0.1, 0.3***

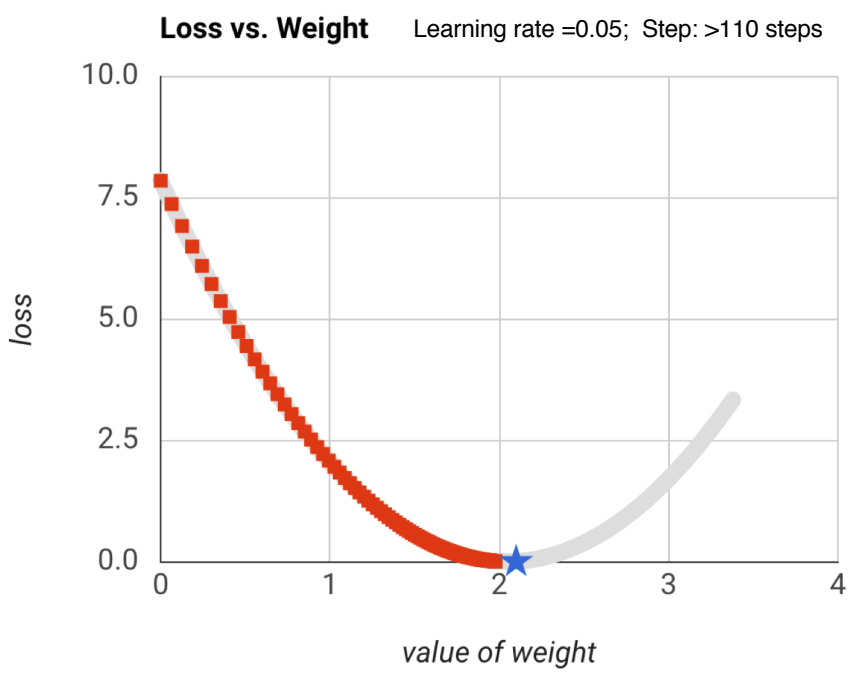
Youtube Video: Learning Rate by Andrew Ng (8 mins)
https://youtu.be/CYIR9oYhYuY

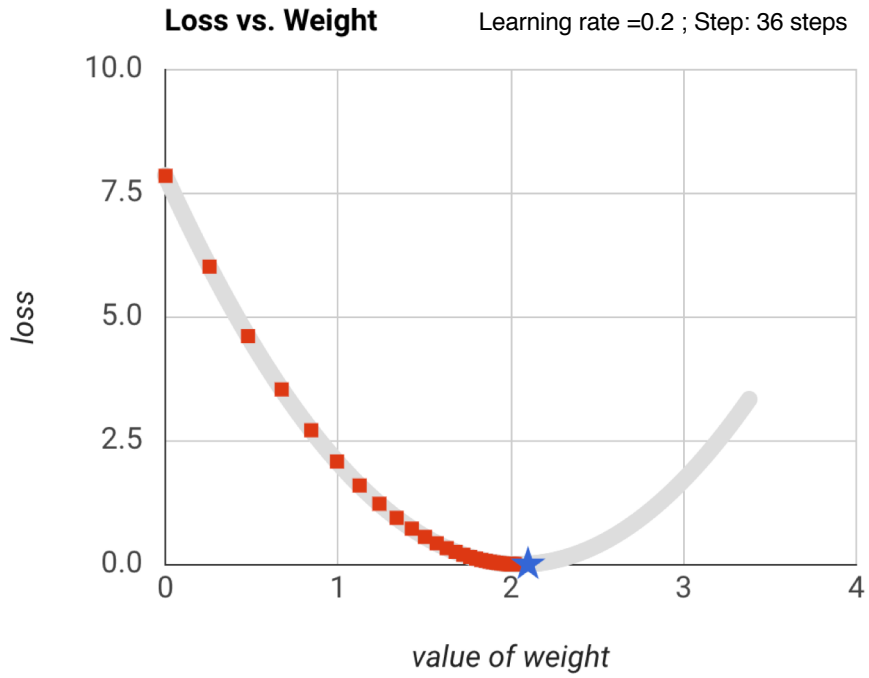
1,2 Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow

:: Linear Regression

Try different learning rates by yourself in Google's interactive demo

Experiment with different learning rates and see how they affect the number of steps required to reach the minimum of the loss curve.





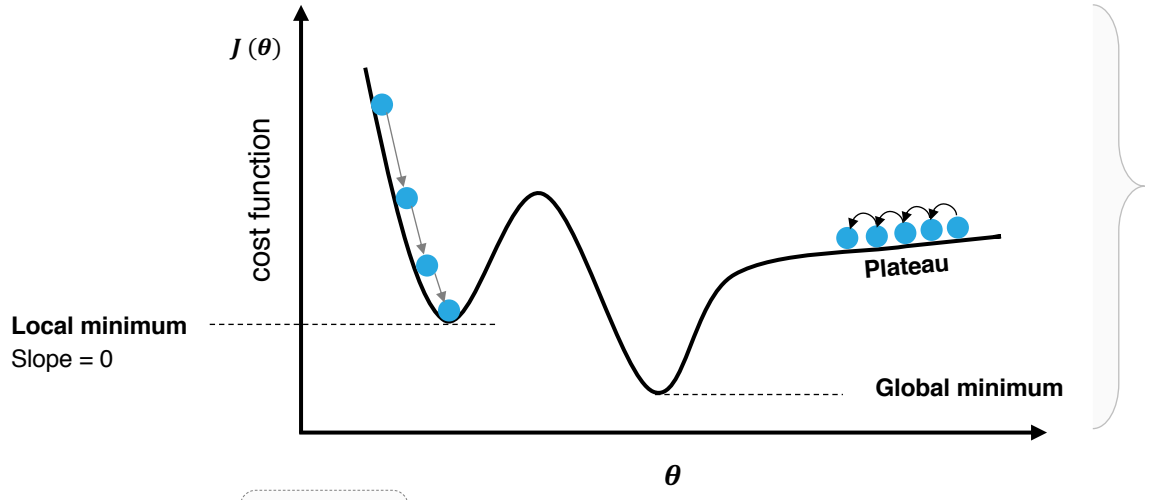
Live playground from google
https://developers.google.com/machine-learning/crash-course/fitter/graph

:: Linear Regression

Gradient Descent's issue

Not all cost functions look like nice regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum very difficult. [1]

Gradient Descent might get stuck in local optima



- If the random initialization starts the algorithm on the left, then it will converge to a local mini-mum, which is not as good as the global minimum. [2]
- If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum. [3]

$\theta_1 := \theta_1 - \alpha \frac{d}{d\,\theta_1} J(\theta_1)$ // as the gradient(slope) at the local minimum is 0

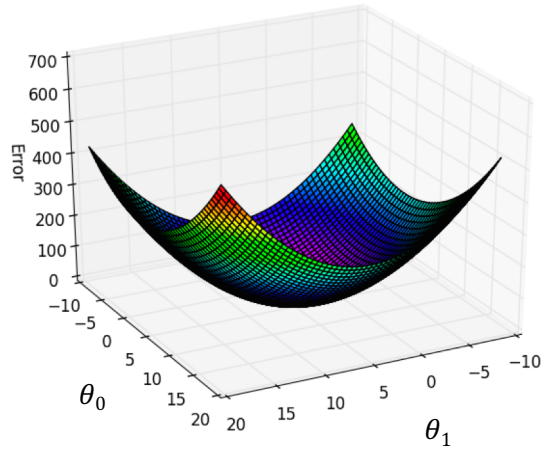
So $\theta_1 := \theta_1 - \alpha \cdot 0$ // so the gradient descent might get stuck in local minimum

1,2 ,3 & diagram Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow

:: Linear Regression

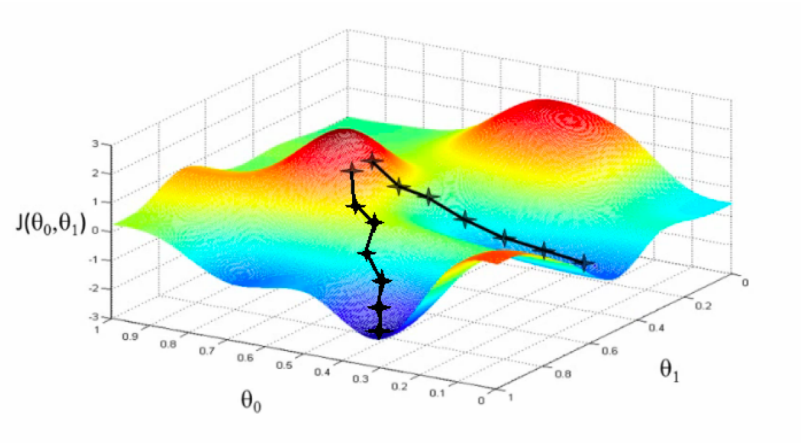
Gradient Descent for convex function & non-convex function

If your error surface is convex and with an appropriate learning rate (i.e. not too high), then convergence to the global minima is guaranteed. However, in many real-world problems, the error surface is generally highly non-convex, meaning there are a plethora of local minima. In this situation, depending on the initial parameters we choose, the solution could converge to a local minima that is not necessarily the global minima.[1]



Convex function : Just one minimum

- Fortunately, the MSE cost function for a Linear Regression model happens to be a *convex function.*[2]
- This implies that there are no local minima, just one global minimum. [3]



Non-Convex function: More than one minimum

$J(\theta)$ is usually non-convex in many real-world problem, gradient descent

- Can be susceptible to local minimum[4]
- Strong dependency on initial values. Different starting points end up with completely different local minimum.
- In practice converging to local minimum is not usually a huge problem[5]

1& Fig1 Source: https://alykhantejani.github.io/a-brief-introduction-to-gradient-descent/
2,3 Soure: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow
Fig.2 &4,5 Sourcce: Adnrew Ng

:: Linear Regression

More about Gradient Descent

:: Linear Regression

The computation methods for estimating the parameters

There are a couple of approaches to find the value of parameter θ that minimizes the cost function $J(\theta)$

.

The approaches to solving $\theta^* = \underset{\theta}{argmin}\ J(\theta)$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- **Normal Equation (Closed Form)**

  It's a method to solve for θ analytically.

  Using a direct "closed-form" equation that directly computes the model parameters that best fit the model to the training set (i.e., the model parameters that minimize the cost function over the training set). [1]

  It's suitable for small feature set (e.g. < 1000 features).

- **Gradient Descent**

  Using an iterative optimization approach, called Gradient Descent (GD), that gradually tweaks the model parameters to minimize the cost function over the training set, eventually converging to the same set of parameters as the first method. [2]

  Gradient Descent is better choice than Normal Equation when there are a large number of features, or too many training instances to fit in memory.

1, 2 Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow

Picture: Designed by Asierromero / Freepik

:: Linear Regression

The computation methods for estimating the parameters

There are a couple of approaches to find the value of parameter θ that minimizes the cost function $J(\theta)$

.

The approaches to solving $\theta^* = \underset{\theta}{argmin}\ J(\theta)$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- **Normal Equation (Closed Form)**

  It's a method to solve for θ analytically.

  Using a direct "closed-form" equation that directly computes the model parameters that best fit the model to the training set (i.e., the model parameters that minimize the cost function over the training set). [1]

  It's suitable for small feature set (e.g. < 1000 features).

- **Gradient Descent**

  Using an iterative optimization approach, called Gradient Descent (GD), that gradually tweaks the model parameters to minimize the cost function over the training set, eventually converging to the same set of parameters as the first method. [2]

  Gradient Descent is better choice than Normal Equation when there are a large number of features, or too many training instances to fit in memory.

In a simple linear regression the coefficients are calculated as:

$$\beta_0 = \bar{y} - \beta_1 \cdot \bar{x} \qquad \beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

In a matrix notation, the coefficients are calculated as:

$$\beta = (X^T X)^{-1} \cdot X^T \cdot y$$

1, 2 Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow

:: Linear Regression

Normal Equation (advanced topic)

To find the value of θ that minimizes the cost function, there is a closed-form solution —in other words, a mathematical equation that gives the result directly.

This is called the Normal Equation[1].

- **Equation of multivariate linear regression：**

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n$$

Now we'll represent the problem in matrix notation. :

$$x = \begin{bmatrix} 1 & x_{11} & \ldots & x_{1n} \\ 1 & x_{21} & \ldots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \ldots & x_{mn} \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

So we can now rewrite the hypothesis function as

$$h_\theta(x) = \theta^T x$$

- **Set derivatives equal to zero and solve for parameters** $\theta_0, \theta_1 \ldots \theta_n$

We will minimize $J(\theta)$ by explicitly taking its derivatives with respect to the θ_j's, we set its derivatives to zero, and obtain the normal equations:

$$\frac{\partial}{\partial \theta} J(\theta) = 2X^T X_\theta - 2X^T y = 0 \quad \Rightarrow \quad X^T X_\theta = X^T \cdot y$$

Thus, the value of θ that minimizes $J(\theta)$ is given in closed form by the equation

$$\theta = (X^T X)^{-1} \cdot X^T \cdot y \qquad // \ (X^T X)^{-1} \text{ is inverse of matrix } X^T X$$

The Normal Equation gets very slow when the number of features grows large (e.g., 100,000) [2]

Video： Normal Equation

https://www.youtube.com/watch?v=B-Ks01zR4HY&t=555s

https://youtu.be/FZ1qPqVeMSQ

1, 2 Source: Aurélien Géron , Hands-On Machine Learning with Scikit-Learn & TensorFlow

:: Linear Regression

Multiple Linear Regression

Multiple Linear regression is also known as Multivariate Linear Regression.

simple linear regression

one explanatory variable

Multiple linear regression

more than one explanatory variable

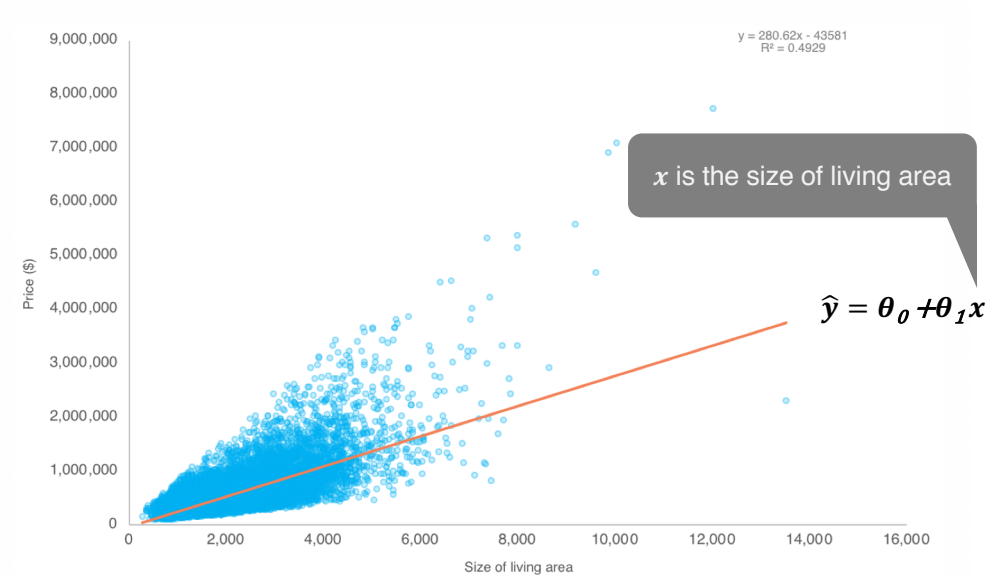
:: Linear Regression

Multiple Linear Regression

In previous example for predicting house price, we only use the size of living area to predict the price. It's pretty simple but of very less use in real world scenarios.

Living Area (Feet^2)	Price ($)
1180	221,900
2570	538,000
770	180,000
1960	604,000
1680	510,000
5420	1,225,000
1715	257,500
1060	291,850
1780	229,500
1890	323,000
3560	662,500
1160	468,000
1430	310,000
1370	400,000
1810	530,000
...	...
x	y

This simple linear regression model is only based on one feature " Living Area"



Picture of house: Designed by Freepik

:: Linear Regression

Multiple Linear Regression

Generally one dependent variable depends on multiple factors. For example, the price of a house depends on many factors like the neighborhood it is in, size of it, no.of rooms, attached facilities, distance of nearest station from it, distance of nearest shopping area from it, etc [1].

The house's price depends on **multiple features** (variables)

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

`1 Source: https://www.hackerearth.com/practice/machine-learning/linear-regression/multivariate-linear-regression-1/tutorial
Picture of house: Designed by Freepik/

:: Linear Regression

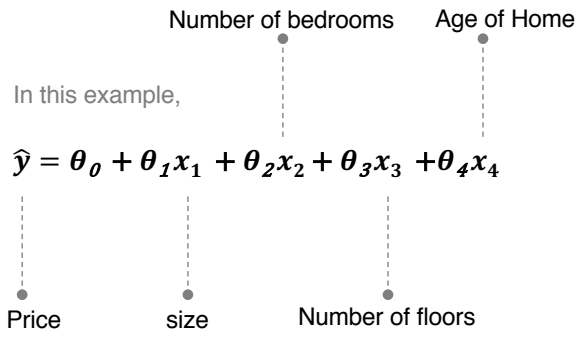
Multiple Linear Regression

This is quite similar to the simple linear regression model we have discussed previously, but with multiple independent variables contributing to the dependent variable. Each training sample has an $\boldsymbol{x}$ made up of multiple input values and a corresponding $\boldsymbol{y}$ with a single value. The the equation of multivariate linear regression is as follow,

- **Simple Linear Regression:** $\hat{y} = h(x) = \theta_0 + \theta_1 x$
- **Multiple Linear Regression:** $\hat{y} = h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$

example

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...



1 Source: https://onlinecourses.science.psu.edu/stat501/node/382

:: Linear Regression

Multiple Linear Regression

For convenience of notation, we can define x_0 =1. Thus, we simplify the equation of multiple linear regression as follow.

Equation :

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$$

For convenience of notation, define x_0 =1, then

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$$

In the multiple regression setting, because of the potentially large number of predictors, it is more efficient to use matrices to define the regression model and the subsequent analyses [1]

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n \qquad \underbrace{[\theta_0 \quad \theta_1 \quad \theta_2 \quad \cdots \quad \theta_n]}_{\theta^T} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{X}$$

$$\hat{y} = \theta^T X$$

Linear Regression With Multiple Variables (8mins)
https://youtu.be/Q4GNLhRtZNc (by Andrew Ng)

1 Source: https://onlinecourses.science.psu.edu/stat501/node/382

:: Linear Regression

Multiple Linear Regression

For multiple linear regression model,

We define the **cost function**:

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

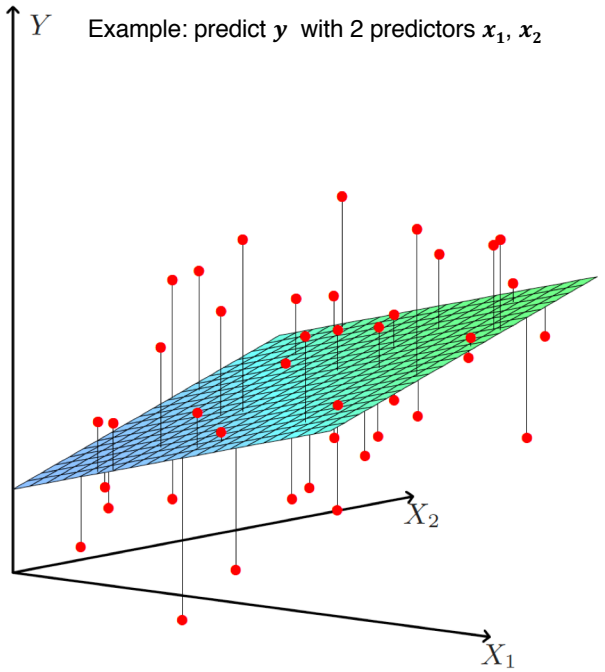
m is number of training instances

$$where \quad \hat{y} = h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i$$

$$\boldsymbol{J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n))^2}$$

How to find the appropriate parameter $\theta_0, \theta_1, \dots, \theta_n$ in order to minimize the cost function/loss function $J(\theta)$?

- Normal Equation
- Gradient Descent



Example: predict y with 2 predictors x_1, x_2

we want the hyper-plane that “best fits’ the training samples. In other words, we seek the linear function of X that minimizes the sum of squared residuals (error) from Y [1]

1 & Figure Source: The Elements of Statistical Learning by Trevor Hastie,etc

:: Linear Regression

Multiple Linear Regression

Example:, we can use **Normal Equation** to find the value of the coefficients/parameters θ that minimizes the cost function

example

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$x = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

- The inputs can be represented as an X matrix in which each row is sample and each column is a dimension.
- The outputs can be represented as y matrix in which each row is a sample.

$$\hat{y} = \theta^T X$$

$$\theta = (X^T X)^{-1} \cdot X^T \cdot y$$ // value of θ that minimizes the cost function can be solved by this equation

:: Linear Regression

Multiple Linear regression

Use Gradient Descent to find the value of parameters θ that minimize the cost function $J(\theta)$

Gradient Descent for multiple Linear Regression :

repeat until convergence {

// Simultaneously update θ_j for every j=0,1, …,n

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Gradient

$\nabla_\theta J(\theta)$ is the partial derivatives of the cost function with respect to the parameters $\theta_0, \theta_1, \dots, \theta_n$

Detail

Hypothesis: $h_\theta(x) = \theta^T X = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n$

Parameters: $\theta\ (\theta_0, \theta_1, \dots, \theta_n)$

Cost Function:

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

Gradient $\nabla_\theta J(\theta)$:

$$\frac{\partial}{\partial\ \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$$

$x_j^{(i)}$ is the j[th] features of i[th] observation

Gradient Descent:

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$$

(Simultaneously update θ_j for every j=0,1, …,n)

}

:: Linear Regression

Multiple Linear regression

Use Gradient Descent to find the value of parameters θ that minimize the cost function J (θ)

Gradient Descent for multiple Linear Regression :

repeat until convergence {

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

(Simultaneously update θ_j for every j=0,1, …,n)

}

Video: Gradient Descent For Multiple Variables (5mins)
https://youtu.be/pkJjoro-b5c

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

example

$$\hat{y} = h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$$\theta_3 := \theta_3 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_3^{(i)}$$

$$\theta_4 := \theta_4 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_4^{(i)}$$

:: Linear Regression

Summary: **Gradient Descent** vs **Normal Equation**

- Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm[1].
- In Normal Equation approach the value of θ that minimizes the cost function is solved directly rather than iteratively. $\theta = (X^T X)^{-1} \cdot X^T \cdot y$

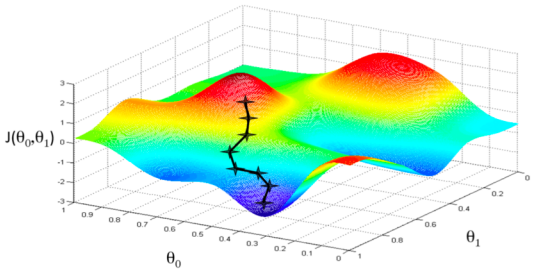
Gradient Descent

Disadvantage

- Need to choose learning rate
- Needs many iterations

Advantage

- Works well even when n (the number of features) is large. e.g. n =10,000



Normal Equation

Advantage

- No Need to choose learning rate
- Don't' need to iterations
- Works well when n (the number of features) is small. e.g. n =100

Disadvantage

- Need to compute $(X^T X)^{-1}$
- Slow if the number of features is very large

Computational Complexity

The Normal Equation computes the inverse of $(X^T X)$, which is a $n \times n$ matrix (where n is the number of features). The computational complexity of inverting such a matrix is typically about $O(n^{2.4})$ to $O(n^3)$. If the number of the feature is large, the computing can be very slow

1 Source:https://machinelearningmastery.com/gradient-descent-for-machine-learning/
2 Source: http://cs229.stanford.edu/notes/cs229-notes1.pdf

:: Linear Regression

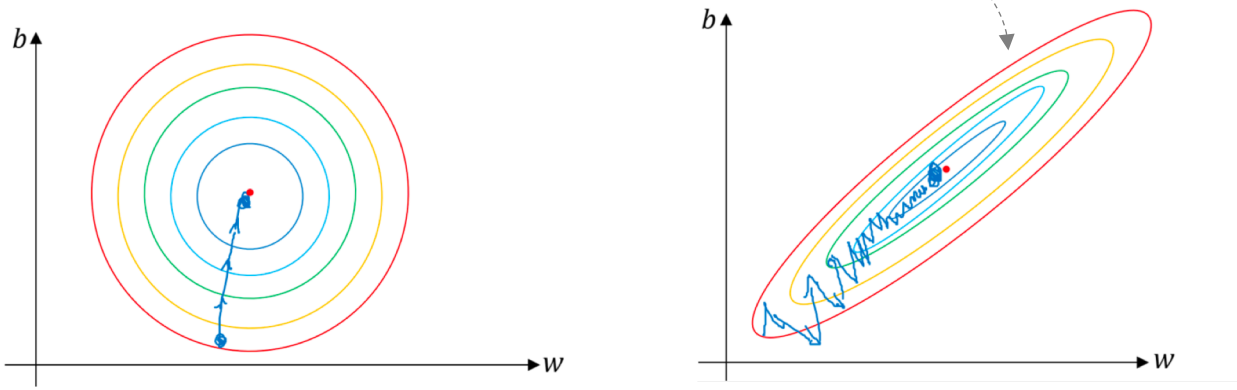
Feature Scaling

One of the most important transformations you need to apply to your data is feature scaling. With few exceptions, Machine Learning algorithms don't perform well when the input numerical attributes have very different scales[1]. When using Gradient Descent, you should ensure that all features have a similar scale, or else it will take much longer to converge[2].

For example,

- x_1 = size (100-2000 feet^2)
- x_2 = number of bedrooms (1-5)

Because they have different scale, it would take longer time to converge



Gradient Descent with feature scaling

Gradient Descent without feature scaling

As you can see, on the left the Gradient Descent algorithm goes straight toward the minimum, thereby reaching it quickly, whereas on the right it first goes in a direction almost orthogonal to the direction of the global minimum, and it ends with a long march down an almost flat valley. It will eventually reach the minimum, but it will take a long time[3].

1,2 ,3 Source: Hands-On Machine Learning with Scikit-Learn & TensorFlow
Source of Figure: https://medium.com/@ImadPhd/gradient-descent-algorithm-and-its-variants-10f652806a3

:: Linear Regression

Feature Scaling

How to make sure multiple features are on a similar scale ?

There are two common ways to get all attributes to have the similar scale

- Min-max Scaling (many people call this Normalization): the data is scaled to a fixed range - usually 0 to 1
- Standardization: Standardizing the features so that they are centered around 0 with a standard deviation of 1

Min-Max Normalization

$$x' = \frac{x - x_{\min}}{x_{max} - x_{\min}}$$

where

- x is an original value
- x' is the normalized value

For example,

- x_1 = size (100-2000 feet^2)
- x_2 = number of bedrooms (1-5)

$$x_1 = \frac{size - 100}{1800}$$

$$x_2 = \frac{\#bedrooms - 1}{4}$$

Z-score Normalization / Standardization

$$z = \frac{x - \mu}{\sigma}$$

where

- x is an original value,
- z is the standard score
- μ is the mean of that feature vector
- Sigma σ is its standard deviation from the mean

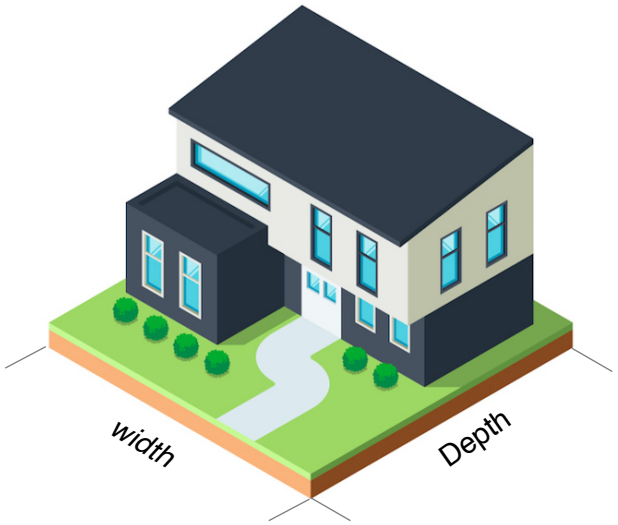
Standardization involves rescaling the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one [1]

1 Source: http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.htm

:: Linear Regression

Create New Feature

Feature / Variable creation is a process to generate a new variables / features based on existing variable(s).



Previously

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ // Housing Prices Prediction

Frontage's width Depth

Area x

x = Width * Depth. // combine frontage and depth as new feature "Area"

Now

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ ⟹ $h_\theta(x) = \theta_0 + \theta_1 x$ // so just use one feature "area" instead of two features

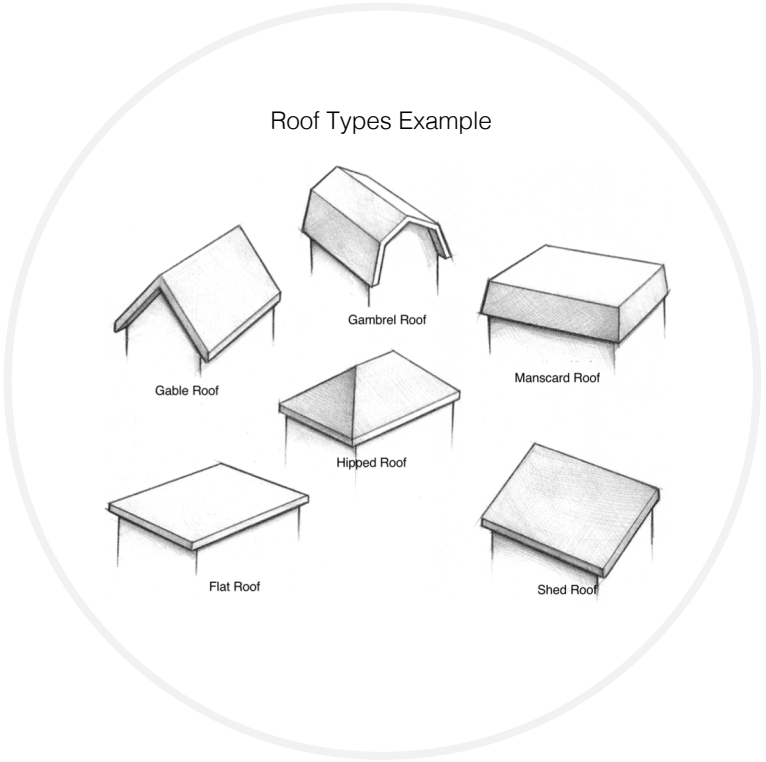
simplified as

Picture of house: Designed by Freepik

:: Linear Regression

If the inputs are categorical features instead of number

Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers.[1]



Roof types

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \cdots + \theta_n x_n$$

Suppose x_4 is the **roof types** which include Flat Roof, Gable Roof, Gambrel Roof, Manscard Roof, Shed Roof, Hipped Roof, Dormer Roof, etc.

However, Linear Regression algorithm can't work with such categorical data directly.

What to do next ?

1 Source: https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/
Picture of house: Designed by Freepik/
Picture of roof: http://www.johnriebli.com/roof-types--house-styles.html

:: Linear Regression

One Hot Encoding

Often features are not given as continuous values but categorical. For example, a person's gender is not number. We need to represent the category values as number so that they can be used in our linear regression algorithm. We can use one-hot encoding to convert categorical features into numbers.

1 Source: http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-categorical-features
2 Source: https://www.quora.com/What-is-one-hot-encoding-and-when-is-it-used-in-data-science

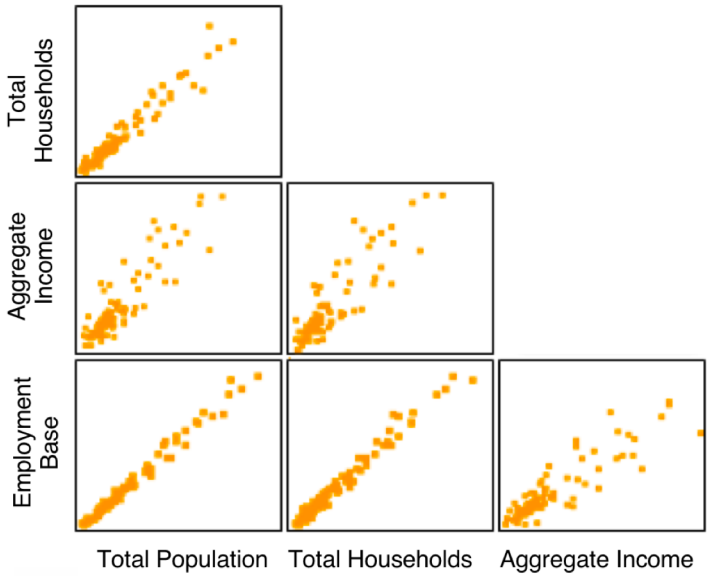
:: Linear Regression

Multicollinearity

Multicollinearity is one of the most important issues in regression analysis, as it produces unstable coefficients' estimates and makes the standard errors severely inflated.[1]

"multicollinearity" refers to predictors that are correlated with other predictors in regression

Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other. In other words, it results when you have factors that are a bit redundant.[2]



There are strong correlations among these variables

Multicollinearity causes the following two basic types of problems:[3]

- The coefficient estimates can swing wildly based on which other independent variables are in the model. The coefficients become very sensitive to small changes in the model.
- Multicollinearity reduces the precision of the estimate coefficients, which weakens the statistical power of your regression model. You might not be able to trust the p-values to identify independent variables that are statistically significant.

Multicollinearity is NOT a big concern in Machine Learning

- Multicollinearity makes it hard to interpret your coefficients, and it reduces the power of your model to identify independent variables that are statistically significant. [4]
- Although multicollinearity affects the coefficients and p-values, but it does not affect the overall fit or the predictions of the model. In most ML applications, we don't care about coefficients themselves, just the loss of our model predictions[5]. If your primary goal is to make predictions, and you don't need to understand the role of each independent variable, you don't need to reduce severe multicollinearity[6].

1 source: https://rd.springer.com/article/10.1007/s11135-017-0571-y
2 http://blog.minitab.com/blog/understanding-statistics/handling-multicollinearity-in-regression-analysis
3,,4 6 source: http://statisticsbyjim.com/regression/multicollinearity-in-regression-analysis/
5 source: https://stats.stackexchange.com/questions/168622/why-is-multicollinearity-not-checked-in-modern-statistics-machine-learning

:: Linear Regression

Polynomial Regression

What if your data is actually more complex than a simple straight line? Surprisingly, you can actually use a linear model to fit nonlinear data. A simple way to do this is to add powers of each feature as new features, then train a linear model on this extended set of features. This technique is called Polynomial Regression.[1]

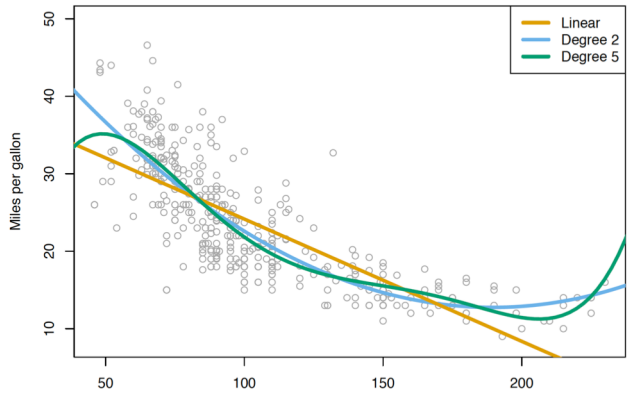
$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \cdots + \theta_n x^n$$

where *n* is called the **degree** of the polynomial

For lower degrees, the relationship has a specific name like quadratic.

- n = 1, a first degree polynomial is simply a **straight line**. ⇢ $\hat{y} = \theta_0 + \theta_1 x$
- n = 2 , a second degree polynomial is called a **quadratic.** ⇢ $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2$
- n = 3, it is **cubic.** ⇢ $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$
- n = 4, it's called **quartic.** ⇢ $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

The polynomial models can be used to approximate a complex nonlinear relationship



Video: Polynomial Regression by Andrew Ng (5 mins)
https://youtu.be/Hwj_9wMXDVo?t=2m22s

live demo for Ploynomial Regression

https://arachnoid.com/polysolve/

Image Source: https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/linear_regression.pdf
Source: Hands-On Machine Learning with Scikit-Learn & TensorFlow I

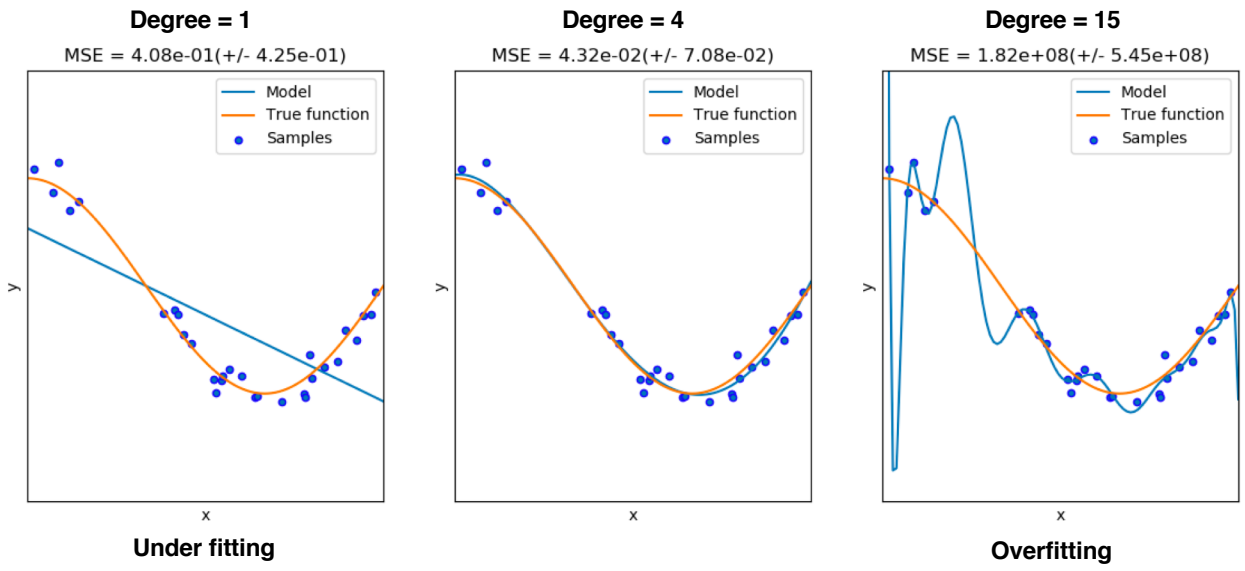
:: Linear Regression

In regression, overfitting occurs frequently

The more parameters or features in your model, the more freedom it has to fit the data. However, the model is also more prone to overfitting the training data.

Example: the models have polynomial features of different degrees.

Overfitting is due to higher order polynomials



Under fitting | **Overfitting**

Plots of polynomials having various orders n

The plot shows the real function that we want to approximate, which is a part of the cosine function.

- We can see that a linear function (polynomial with degree 1) is not sufficient to fit the training samples. This is called underfitting.
- A polynomial of degree 4 approximates the true function almost perfectly.
- However, for higher degrees the model will overfit the training data, i.e. it learns the noise of the training data.

http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html#

:: Linear Regression

How to reduce overfitting problem ?

There are some strategies to address overfitting problem in Linear regression model

Reduce model's complexity

- **Reduce the number of features**
  It's prone to overfitting if there are lots of features but with very little training data

  Throw away some of features. E.g.

  $\hat{y} = \theta_0 + \theta_1 x_1 \ \cancel{+\theta_2 x_2} + \theta_3 x_3 \ \cancel{+\theta_4 x_4} + \cdots + \theta_n x_n$

- **Reduce the degree of polynomial**
  Try lower degree in case of overfitting. E.g.

  $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \ \cancel{+\theta_4 x^4} + \cancel{+\theta_5 x^5}$

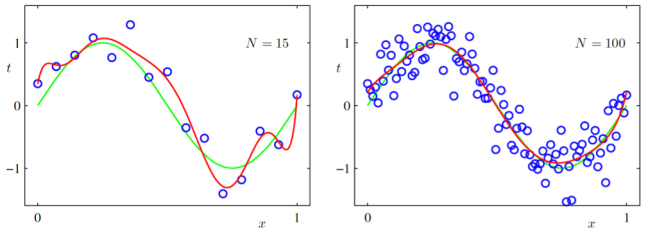
Use L1 / L2 regularization

- **Ridge regression** is a regularized version of Linear Regression. It uses L2-norm penalty. Ridge regression includes all of the features in the model. As the value of alpha/Lambda increases, the coefficient shrinks.

- **Lasso regression** is a regularized version of Linear Regression. It uses L1-norm penalty
  L1 will push certain weights to be exactly 0. Lasso Regression automatically performs feature selection and outputs a *sparse model* .

Get more data

For a given model complexity, the over-fitting problem become less severe as the size of the data set increases.[1]

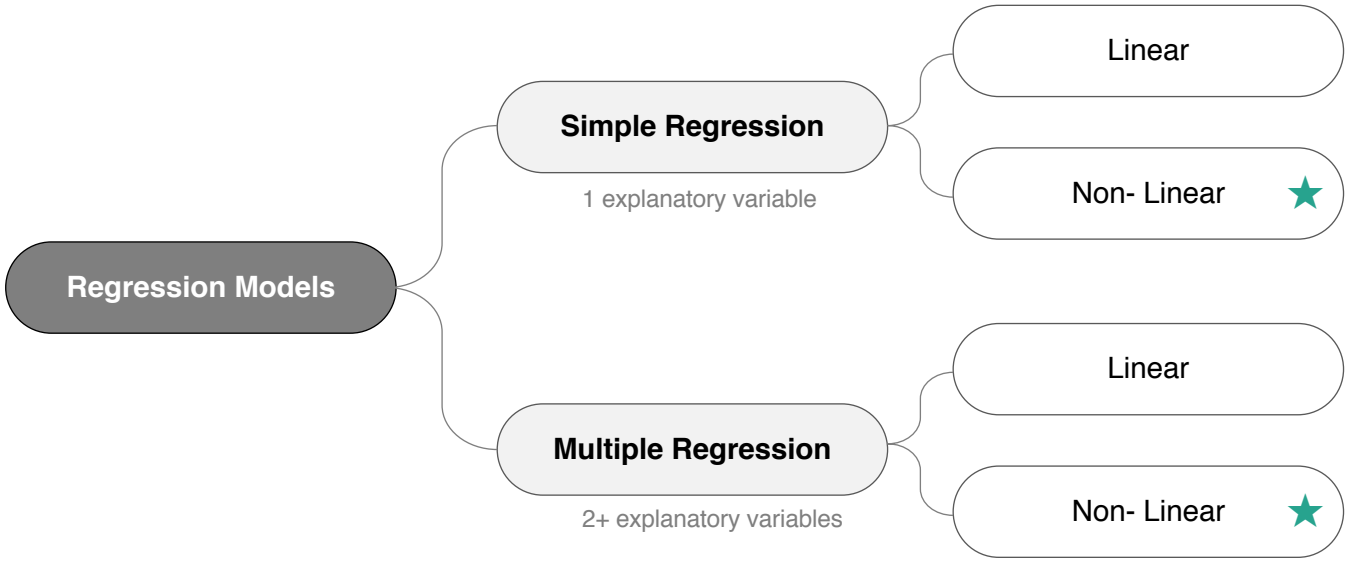


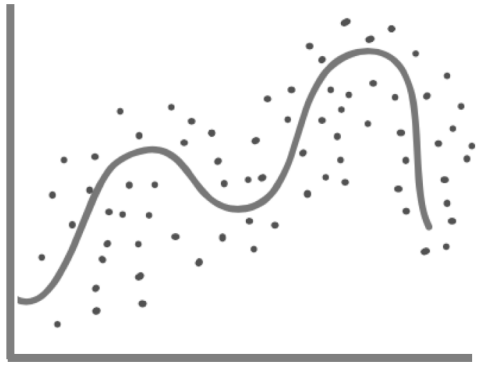
N = 15 data points (left plot) and N = 100 data points (right plot). We see that increasing the size of the data set reduces the over-fitting problem.[2]

1, 2 & image Source: Christopher M. Bishop, Pattern Recognition and Machine Learning

:: Nonlinear Regression

Type of Regression Model





Nonlinear regression is appropriate when the relationship between the dependent and independent variables is not intrinsically linear.[1]

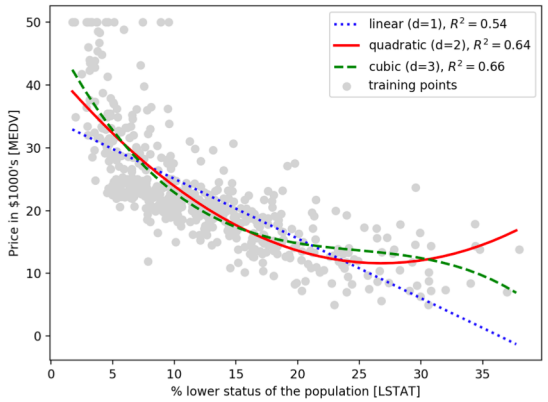
1 source: https://www.ibm.com/support/knowledgecenter/en/SSLVMB_20.0.0/com.ibm.spss.statistics.cs/nonlinearregressionanalysis_table.htm

:: Nonlinear Regression

What's nonlinear regression?

Nonlinear regression is a method of finding a nonlinear model of the relationship between the dependent variable and a set of independent variables. Unlike traditional linear regression, which is restricted to estimating linear models, nonlinear regression can estimate models with arbitrary relationships between independent and dependent variables. [1]

Polynomial regression is NOT considered as non-linear regression



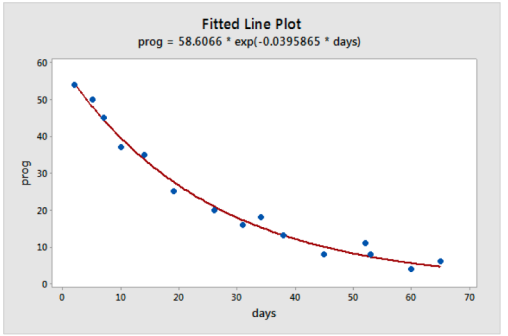
Although this model allows for a nonlinear relationship between *Y* and *X*, polynomial regression is still considered linear regression since it is linear in the regression coefficients, $\theta_1,\theta_2,...,\theta_n$. [2]

"Nonlinear" refers to a fit function that is a nonlinear function of the parameters

For example, the equation $\hat{y} = \frac{b_0 x^{b1}}{x+b_2}$ is a nonlinear function of the fitting parameters.

One simple nonlinear model is the exponential regression model[3]

$$y_i = \theta_0 + \theta_1 \exp(\theta_2 x_{i,1} + \ldots + \theta_{p+1} x_{i,1})$$



Some other examples of equation for nonlinear regression models:

$$y_i = \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}} + \epsilon_i$$

$$y_i = \frac{\beta_0+\beta_1 x_i}{1+\beta_2 e^{\beta_3 x_i}} + \epsilon_i$$

$$y_i = \beta_0 + (0.4-\beta_0)e^{-\beta_1(x_i-5)} + \epsilon_i$$

1 Source: https://www.ibm.com/support/knowledgecenter/en/SSLVMB_20.0.0/com.ibm.spss.statistics.help/idh_nlre.htm
2,3 Source: https://onlinecourses.science.psu.edu/stat501/node/324
Image Soure: https://charleshsliao.wordpress.com/2017/06/16/ransac-and-nonlinear-regression-in-python/

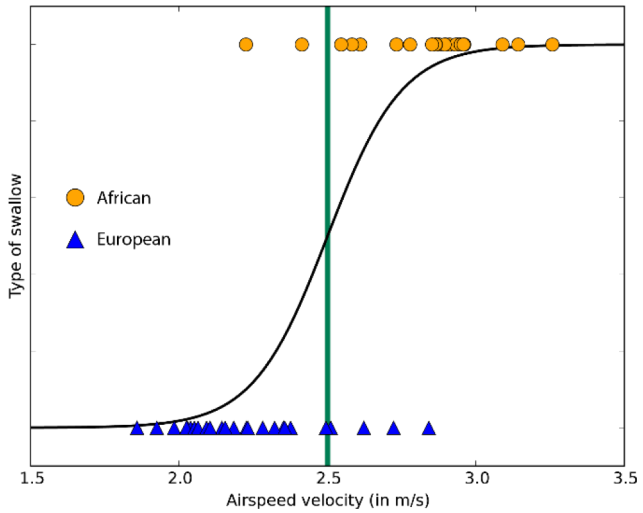
Logistic Regression

:: Logistic Regression

What's logistic regression ?

Logistic regression is a simple classification algorithm that can **predict the probability** of a binary response belonging to one class or the other.

- Logistic Regression is commonly used to estimate the probability that an instance belongs to a particular class. (e.g., what is the probability that this email is spam?). This makes it a binary classifier.[1]
    - If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labeled "1"). For example, if the model infers a value of 0.932 on a particular email message, it implies a 93.2% probability that the email message is spam.[2]
    - If the estimated probability is less than 50%, it predicts that it does not belong to that class (i.e., it belongs to the negative class, labeled "0").
- Because of its simplicity, logistic regression is commonly used as a starting point for binary classification problems. [3] Logistic regression can be used as a baseline for evaluating more complex classification methods



Example

A logistic regression to two-class data with just one feature

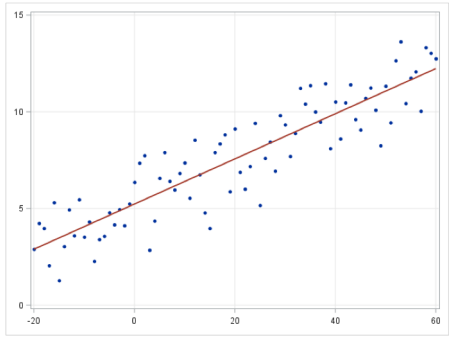
1 source:: Hands-On Machine Learning with Scikit-Learn and TensorFlow
2 source: https://developers.google.com/machine-learning/crash-course/logistic-regression/video-lecture
3Source : Mathworks, Applying Supervised Learning
Source of diagram: https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-choice

:: Logistic Regression

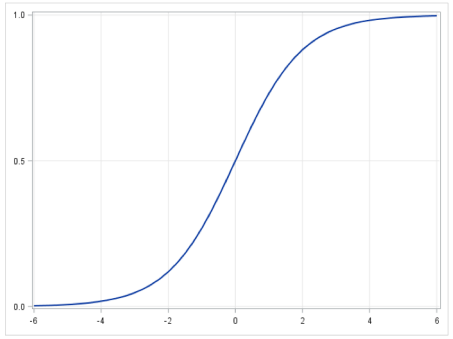
Don’t get confused by its name! It’s a classification rather than regression algorithm

Although it confusingly includes ‘regression’ in the name, logistic regression is actually a powerful tool for two-class and multiclass classification.[1]

- It's fast and simple. The fact that it uses an 'S'-shaped curve instead of a straight line makes it a natural fit for dividing data into groups. [2]



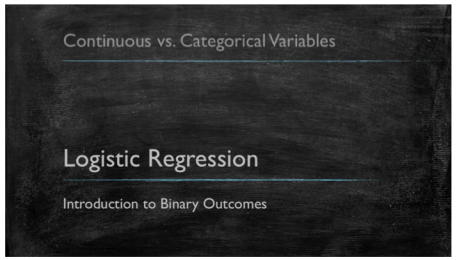
Linear regression



Logistic regression

- In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.
- It predicts the probability, its output values lies between 0 and 1 (as expected).

Logistic Regression – Introduction



YouTube Video (12min)
https://www.youtube.com/watch?v=gNhogKJ_q7U

1&2 Ssource: https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-choice
Source of diagram: https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/

:: Logistic Regression

Logistic regression: Calculating a Probability

Many problems require a probability estimate as output. Logistic regression is an extremely efficient mechanism for calculating probabilities.[1] You might forget how the conditional probability is written, don't worry about it. Let's learn it through the following examples.

It tells us how often A happens given that B happens

$$P(A \mid B)$$

P (A I B) is "Probability of A given B", the probability of A given that B happens.

For example:

- P (Fire I Smoke) means how often there is fire when we see smoke.
- P (Smoke I Fire) means how often we see smoke when there is fire.
- P (bark I night) =0.05 mean the probability that a dog will bark during the middle of the night is 5%

Well, how to calculate the probability ? See next slides for the details

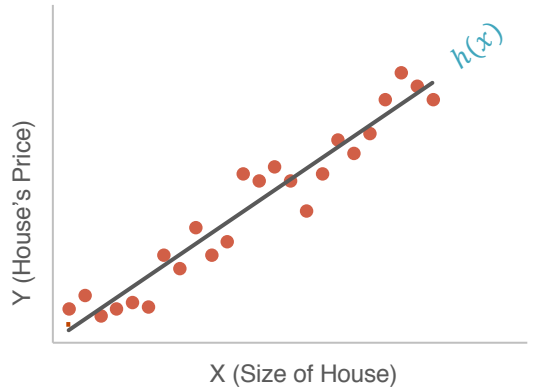
1 source: https://developers.google.com/machine-learning/crash-course/logistic-regression/calculating-a-probability

:: Logistic Regression

Let's recall the Linear regression hypothesis $h_\theta(x)$ first

- **Univariate linear regression** (Simple linear regression)

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



- **multivariate linear regression**

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i$$

$$h_\theta(x) = \theta^T x \quad \textit{// vectorized form}$$

But linear regression is not a good approach for classification problem

Although we can use linear regression to classify the dataset, but often it's not a good idea to do that. One of the key reason is as follow,

- In linear regression the output value is a continues range. So linear regression will predict values outside the acceptable range (e.g. predicting probabilities outside the range 0 to 1) [1]

Andrew Ng explains this in a short video.

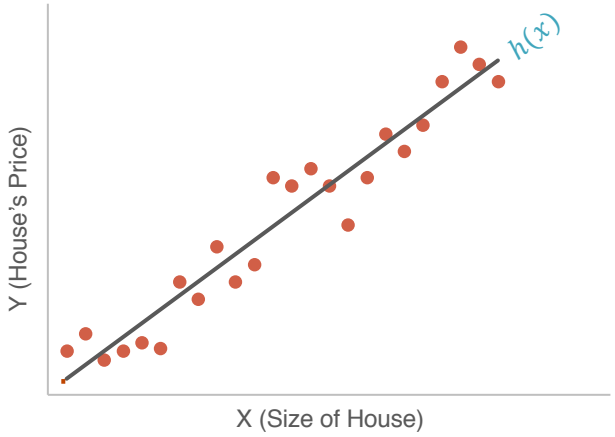
Logistic Regression | Classification
https://www.youtube.com/watch?v=-la3q9d7AKQ

:: Logistic Regression

Now we take a look at the logistic regression hypothesis : $h_\theta(x)$

Linear regression hypothesis $h_\theta(x)$

- Univariate linear regression (Simple linear regression)



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

- Multivariate linear regression

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i$$

$$h_\theta(x) = \theta^T x$$ *// rewrite the equation in vectorized form*

Logistic regression hypothesis $h_\theta(x)$

In a binary classification we use a different hypothesis.

We predict the probability that a given example belongs to the "1" class versus the probability that it belongs to the "0" class

- 1 (positive) : e.g. malignant tumor
- 0 (negative) : e.g. non-malignant tumor

We want $0 \leq h_\theta(x) \leq 1$

So we need to find a function for our hypothesis so that the output is bounded (0,1). Sigmoid function can help us.

$$h_\theta(x) = \theta^T x$$

$$h_\theta(x) = \sigma(\theta^T x)$$

Use sigmoid function to "transform" the linear regression equation $h_\theta(x) = \theta^T x$

Sigmoid/Logistic function: $\sigma(z) = \frac{1}{1+exp^{(-z)}}$

$$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1+exp^{(-\theta^T x)}}$$

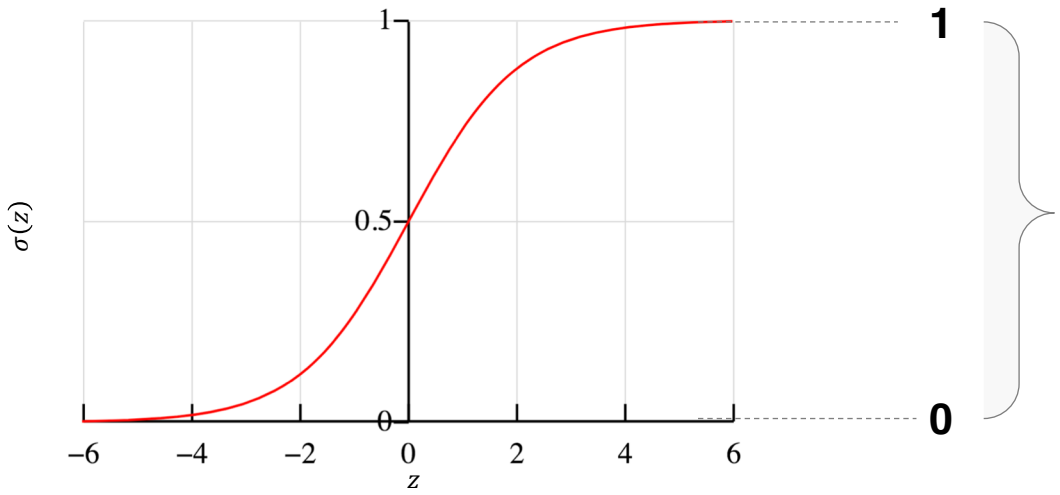
:: Logistic Regression

But what's Logistic/Sigmoid Function ?

"sigmoid" or "logistic" function – it is an S-shaped function that "squashes" the value of $\theta^T x$ into the range [0,1] so that we may interpret $h_\theta(x)$ as a probability.[1] A logistic function or logistic curve is a common "S" shape (sigmoid curve).

- **The sigmoid $\sigma(z)$ takes the S-curve shape.**

  Here is a plot showing $\sigma(z)$



$$h_\theta(x) = \sigma(z) = \frac{1}{1+e^{(-z)}}$$

Squash the value z into [0, 1] , using sigmoid function

Sigmoid function squashes the value (any value) and gives the value between 0 and 1

- $\sigma(z) \geq 0.5 \; when \; z \geq 0$
- $\sigma(z)$ tends towards 1 as $z \to \infty$

- $\sigma(z) \leq 0.5 \; when \; z \leq 0$
- $\sigma(z)$ tends towards 0 as $z \to -\infty$.

$\sigma(z)$, and hence also $h_\theta(x)$, is always bounded between 0 and 1.

:: Logistic Regression

More about Logistic/Sigmoid Function

The benefit of this curve is that the input values can range from $-\infty < t < \infty$ whilst the output ranges from 0 to 1, exactly the range for probability values. Hence its common use as a transfer function.[1]

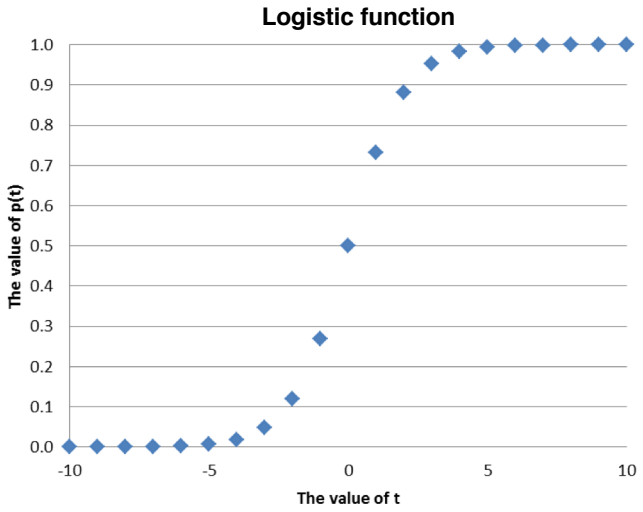
$$\sigma(t) = \frac{1}{1+e^{(-t)}}$$

e^x is exponential function which is also denoted by $exp\,(x)$.

The constant $e \approx 2.71828...$, the base of the natural logarithm.

Example

The Logistic Function or Logisitc Curve			$p(t) = 1/(1+e^{-t})$
t	EXP(t)	EXP(-t)	= 1/(1+EXP(-t))
-10	0.000045	22026.465795	0.000045
-9	0.000123	8103.083928	0.000123
-8	0.000335	2980.957987	0.000335
-7	0.000912	1096.633158	0.000911
-6	0.002479	403.428793	0.002473
-5	0.006738	148.413159	0.006693
-4	0.018316	54.598150	0.017986
-3	0.049787	20.085537	0.047426
-2	0.135335	7.389056	0.119203
-1	0.367879	2.718282	0.268941
0	1.000000	1.000000	0.500000
1	2.718282	0.367879	0.731059
2	7.389056	0.135335	0.880797
3	20.085537	0.049787	0.952574
4	54.598150	0.018316	0.982014
5	148.413159	0.006738	0.993307
6	403.428793	0.002479	0.997527
7	1096.633158	0.000912	0.999089
8	2980.957987	0.000335	0.999665
9	8103.083928	0.000123	0.999877
10	22026.465795	0.000045	0.999955



If you're not familiar with exponential function $exp\,(x)$, you can try to calculate it easily in Excel which provides this function.

1 & image Source :: open.sap.com

:: Logistic Regression

We can treat the output of logistic regression hypothesis $h_\theta(x)$ as estimated probability

"sigmoid" or "logistic" function – it is an S-shaped function that "squashes" the value of $\theta^T x$ into the range [0,1] so that we may interpret $h_\theta(x)$ as a probability.[1]

Linear regression hypothesis

$$h_\theta(x) = \theta^T x$$

Logistic regression hypothesis

When our hypothesis $h_\theta(x)$ outputs some number, we can treat that number as **estimated probability** that $y = 1\ (\text{i.e. positive})$ on input x.

- For example, if we use Size of tumor to predict if the tumor is malignant, and suppose the hypothesis outputs 0.7. i.e. $h_\theta(x)$ =0.7, ⇢ then I'm going to tell my patient that the tumor, has a 70 % chance, or a 0.7 chance of being malignant.[2]

To write this out in math, I'm going to interpret my hypothesis output as $h_\theta(x) = P(y = 1 \mid x)$

- $\boldsymbol{P(y = 1 \mid x)} = h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1+e^{(-\theta^T x)}}$

  // e.g., $P(y = Spam \mid x)$ is the probability of "incoming email x is spam".

- $\boldsymbol{P(y = 0 \mid x)} = 1 - P(y = 1 \mid x) = 1 - h_\theta(x)$

  // e.g. $P(y = normal\ email \mid x)$ is the probability of "incoming email x is normal email"

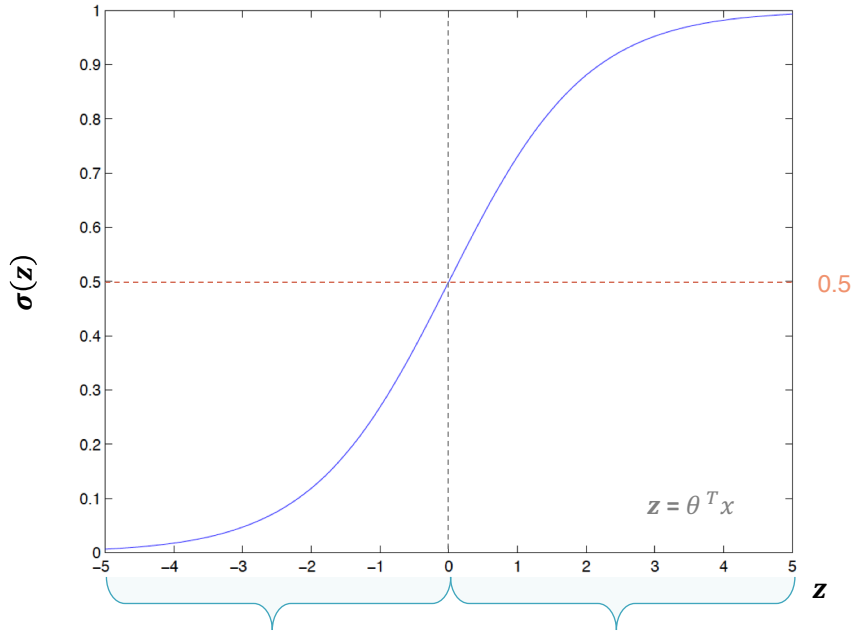
  // note : $P(y = 1 \mid x) + P(y = 0 \mid x) = 1$

1,2 Source: Andrew Ng, https://www.youtube.com/watch?v=t1IT5hZfS48

:: Logistic Regression

Use the logistic regression model to predict

Once the Logistic Regression model has estimated the **probability** $\hat{p} = h_\theta(x)$ that an instance x belongs to the positive class, it can make its prediction ŷ easily. [1]



$\sigma(z) < 0.5$ when $z < 0$ $\sigma(z) \geq 0.5$ when $z \geq 0$,

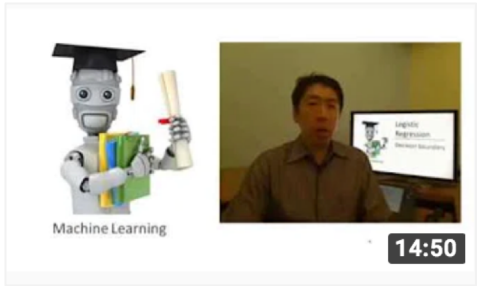
so a Logistic Regression model predicts 1 if $\theta^T x$ is positive, and 0 if it is negative.[2]

$$\hat{p} = h_\theta(x) = \sigma(z) = \frac{1}{1+e^{(-z)}} = \frac{1}{1+e^{(-\theta^T x)}} \quad \text{Where } z = \theta^T x$$

$$\hat{y} = \begin{cases} 0 & if\ \hat{p} < 0.5 \\ 1 & if\ \hat{p} \geq 0.5 \end{cases}$$

// e.g. classify the email as a normal email if $\hat{p} < 0.5$

// e.g. classify the email as a spam email if $\hat{p} \geq 0.5$



Video: Logistic Regression - Decision Boundary
https://www.youtube.com/watch?v=F_VG4LNjZZw

1,2 source:: Hands-On Machine Learning with Scikit-Learn and TensorFlow

:: Logistic Regression

How to find the value of θ for fitting the model ? -1

Our goal is to search for a value of θ so that model estimates high probabilities for positive instances (y =1) and low probabilities for negative instances (y = 0). Instead of finding the best fitting line by minimizing the squared residuals as in ordinary least squares, Logistic regression model uses a different approach with logistic— Maximum Likelihood Estimation.

- **Logistic Regression loss function (log loss)**

  The loss function for linear regression is squared loss. The loss function for logistic regression is **Log Loss.**[1] The loss function over the whole training set is simply the average loss over all training instances. The loss function (known as cost function) for the θ parameters can be defined as:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} \log(\hat{p}^{(i)})\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{p}^{(i)}\right)$$

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

$$h_\theta(x) = \frac{1}{1+e^{(-\theta^T x)}}$$

  - $y^{(i)}$ is the label in a labeled example. Since this is logistic regression, every value of $y^{(i)}$ must either be 0 or 1.
  - $h_\theta(x)$ is the predicted value (somewhere between 0 and 1), given the set of features in x.

- **Why should we use Log Loss as the cost function ?**

  Andrew Ng explains the reason in his excellent video tutorial.

  YouTube Video: Logistic Regression I Cost Function (11mins)
  https://www.youtube.com/watch?v=HIQImHxI6-0

1 source: https://developers.google.com/machine-learning/crash-course/logistic-regression/model-training

:: Logistic Regression

How to find the value of θ for fitting the model ? -2

We now have a cost function that measures how well a given hypothesis $h_\theta(x)$ fits our training data. We can learn to classify our training data by minimizing $J(\theta)$ to find the best choice of θ.

Minimize the cost function $J(\theta)$ to find the best choice of θ

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

As the cost function is convex, so Gradient Descent can be used to find the global minimum (if the learning rate is not too large and you wait long enough).

- **Partial derivative of $J(\theta)$**

  The partial derivative of $J(\theta)$ as given on the left side with respect to θ_j is

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$$

- **Gradient Descent for logistic regression**

  As we want to minimize the cost function $J(\theta)$, so

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$$

// (Simultaneously update θ_j for every j=0,1, …,n)

}

For more details, ⇢ video: Logistic Regression by Andrew Ng, https://www.youtube.com/watch?v=TTdcc21Ko9A

:: Logistic Regression

Regularization in Logistic Regression

Regularization is extremely important in logistic regression modeling. Without regularization, the asymptotic nature of logistic regression would keep driving loss towards 0 in high dimensions. Consequently, most logistic regression models use one of the following two strategies to dampen model complexity:[1]

- **L_2 regularization**

  L_2 regularization helps drive outlier weights (those with high positive or low negative values) closer to 0 but not quite to 0. L_2 regularization always improves generalization in linear models.[2]

$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log h_\theta(x^{(i)}) \right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta(x^{(i)})\right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Add this penalty term to the original Log loss

You can refer to L2 regularization for more details

Regularized Logistic Regression by Andrew Ng (8mins)
https://www.youtube.com/watch?v=IXPgm1e0IOo

- **Early stopping, that is, limiting the number of training steps or the learning rate.**

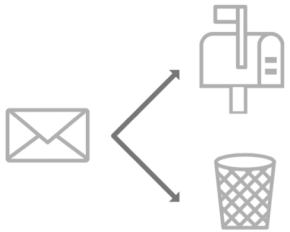
1 source: https://developers.google.com/machine-learning/crash-course/logistic-regression/model-training
2 source: https://developers.google.com/machine-learning/crash-course/glossary#L2_regularization

:: Logistic Regression

About Multiclass Classification

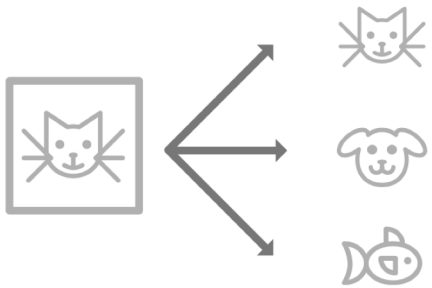
Whereas binary classifiers distinguish between two classes, multiclass classifiers (also called multinomial classifiers) can distinguish between more than two classes [1]

Binary Classification



In a binary classification problem, a single training or test item (instance) can only be divided into two classes—for example, if you want to determine whether an email is genuine or spam [2]

Multiclass Classification



In a multiclass classification problem, it can be divided into more than two—for example, if you want to train a model to classify an image as a dog, cat, or other animal. [3]

2,3 Source: Mathworks, Applying Supervised Learning
1 Source: Hands-On Machine Learning with Scikit-Learn and TensorFlowc

:: Logistic Regression

Use “one-versus-the-rest ” approach for Multiclass Classification

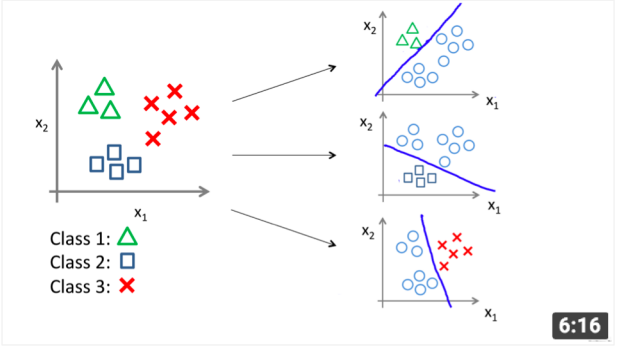
Some algorithms (such as Random Forest classifiers or naive Bayes classifiers) are capable of handling multiple classes directly. Others (such as Support Vector Machine classifiers or Linear classifiers) are strictly binary classifiers. However, there are various strategies that you can use to perform multiclass classification using multiple binary classifiers.[1]

- **The most commonly used strategy is One-Vs-The-Rest**

  Pick a good technique for building binary classifiers (e.g., logistic regression, SVM). Build N different binary classifiers. For the i^{th} classifier, let the positive examples be all the points in class i, and let the negative examples be all the points not in class i. Let f_i be the i^{th} classifier [1]. On a new input x to make a prediction, pick the class i that maximizes $f(x) = arg\max_i f_i(x)$

For example, one way to create a system that can classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on).

Then when you want to classify an image, you get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score. This is called the one-versus-all (OvA) strategy (also called one-versus-the-rest).



Video: One-vs-Rest by Andrew Ng
https://www.youtube.com/watch?v=-Elfb6vFJzc

1 Source: Hands-On Machine Learning with Scikit-Learn and TensorFlowc
2 Source:: http://www.mit.edu/~9.520/spring09/Classes/multiclass.pdf

:: Logistic Regression

Recap: Classification with Logistic regression

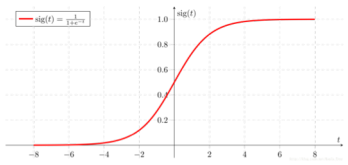
- **Train the model**

  determine best parameters θ_j which can minimize the cost function $J(\theta)$

- **Classify new instance using the trained model**

  - Compute :

    $$\hat{p} = P(y = 1 \mid x) = \frac{1}{1+e^{(-\theta^T x)}}$$

  - Classify new instance as:

    $$\hat{y} = \begin{cases} 0 & if\ \hat{p} < 0.5 \\ 1 & if\ \hat{p} \geq 0.5 \end{cases}$$

Key notation

$$\sigma(z) = \frac{1}{1+e^{-z}}$$ — **Sigmoid function**

$$\theta^T x = \sum_{i=0}^{n} \theta_i x_i = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n$$ — **Weighted sum**

$$\sigma(z) = \sigma(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$ — **Sigmoid function of weighted sum**

logistic regression can handle any number of numerical and/or categorical variables

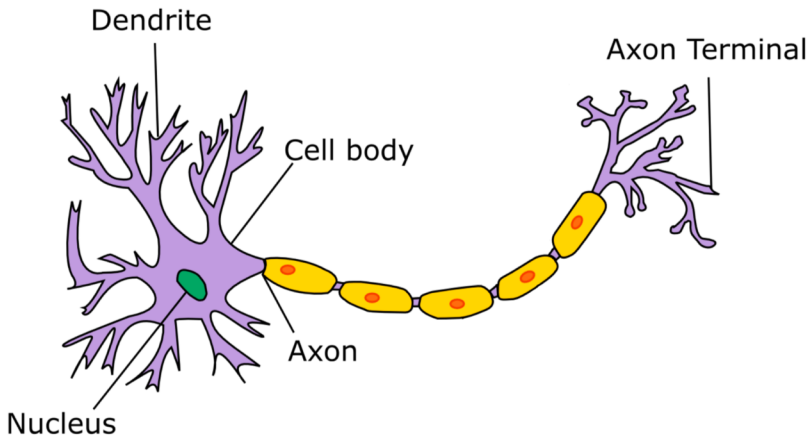
$$P(y = 1 \mid x) = \sigma(z) = \frac{1}{1+e^{-(\theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n)}} = \frac{1}{1+e^{-\theta^T x}}$$

Neural Network – part 1

:: Neural Network

Structure of Neurons in human brain

The human brain can be described as a biological neural network—an interconnected web of neurons transmitting elaborate patterns of electrical signals. Dendrites receive input signals and, based on those inputs, fire an output signal via an axon[1].



- ***Dendrite***: *It receives signals from other neurons*
- ***Cell body***: *It sums all the incoming signals to generate input*
- ***Axon***: *When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons*

1 http://natureofcode.com/book/chapter-10-neural-networks/

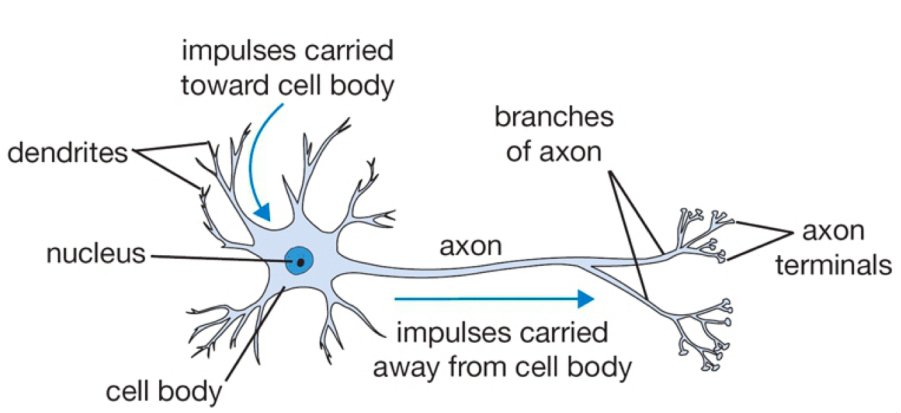
:: Neural Network

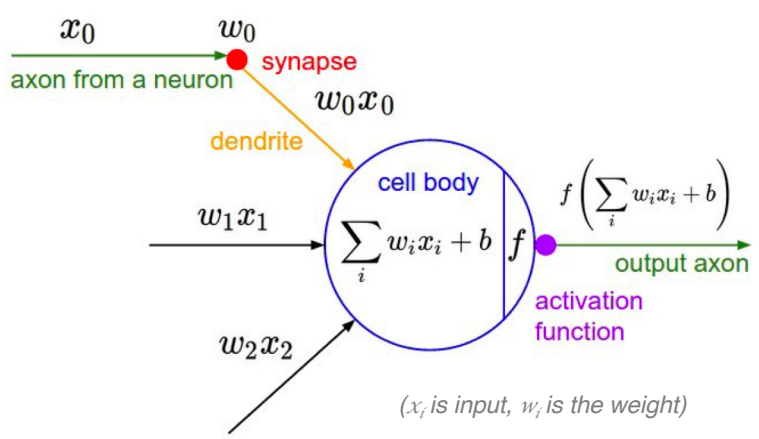
The neuron is the basic computational unit of the brain

Neuron , often called a node or unit, receives input from some other neurons, or from an external source and computes an output. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function to the weighted sum of its inputs.

The idea is that the synaptic strengths (the weights *w*) are learnable and control the strength of influence and its direction: excitory (positive weight) or inhibitory (negative weight) of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can *fire*, sending a spike along its axon[1].

Artificial neural networks are computing systems inspired by the biological neural networks.





biological neuron (left) and a common mathematical model (right)

1 https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc

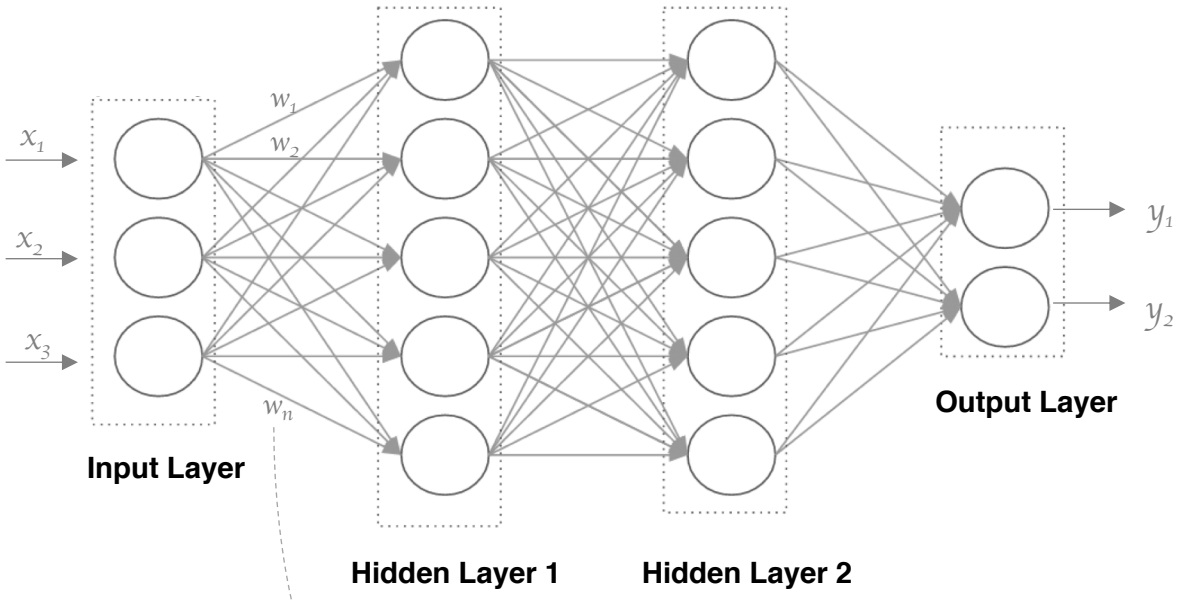
:: Neural Network

Artificial neural network (ANN)

Inspired by the human brain, an ANN is comprised of a network of artificial neurons (also known as "nodes"). These nodes are connected to each other, and the strength of their connections to one another is assigned a value based on their strength. If the value of the connection is high, then it indicates that there is a strong connection[1]. The network is trained by iteratively modifying the strengths of the connections so that given inputs map to the correct response[2].

Example

Feedforward Neural Network



All these connections have **weights** associated with them.

w is the weight which represents the strengths of the connections

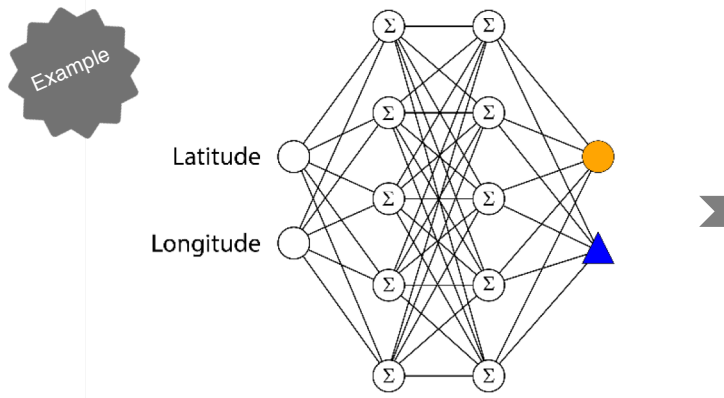
A feedforward neural network is an artificial neural network where connections between the units do not form a cycle. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

1 Source: http://www.saedsayad.com/artificial_neural_network.htm
2Source : Mathworks, Applying Supervised Learning

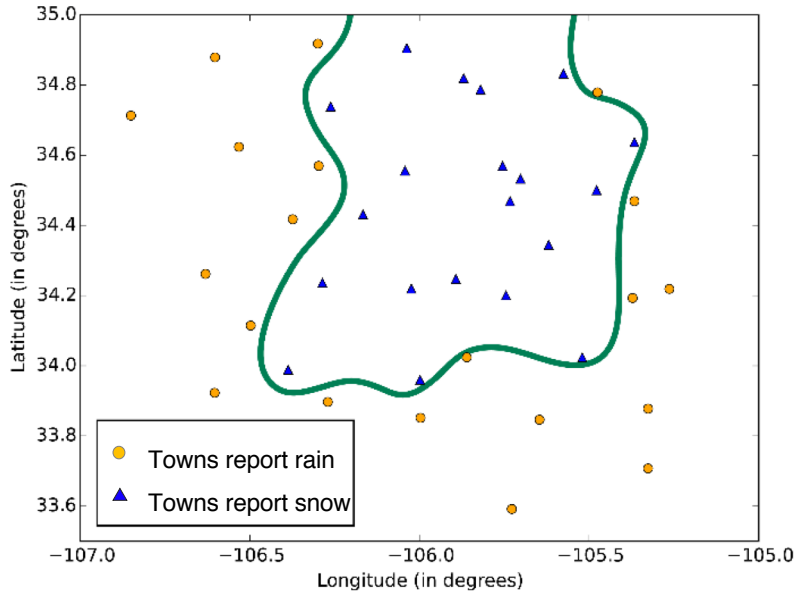
:: Neural Network

Best Used...[1]

- For modeling highly nonlinear systems
- When data is available incrementally and you wish to constantly update the model
- When there could be unexpected changes in your input data
- When model interpretability is not a key concern



Neural network for classification problem



The boundaries learned by neural networks can be complex and irregular

This high performance doesn't come for free, though. Neural networks can take a long time to train, particularly for large data sets with lots of features. They also have more parameters than most algorithms, which means that parameter sweeping expands the training time a great deal [2]

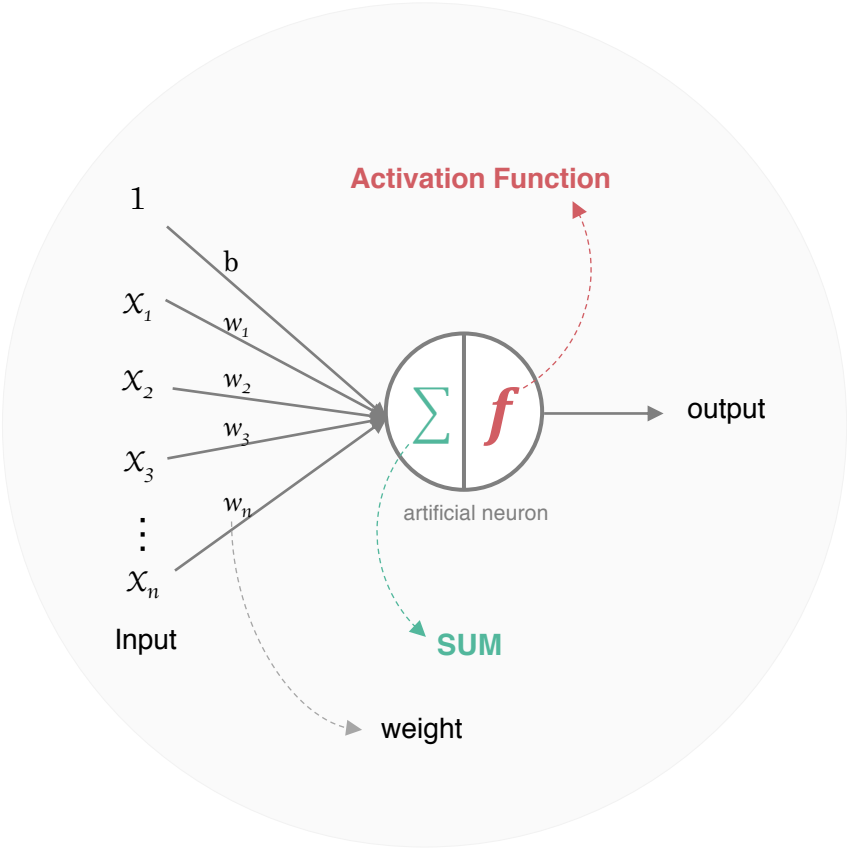
1 Source : Mathworks, Applying Supervised Learning
2 Source: https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice

:: Neural Network

How does artificial neuron work ?

Artificial neurons are most basic units of information processing in an artificial neural network. Each neuron takes information from a set of neurons, processes it, and gives the result to another neuron for further processing.

- The artificial neuron receives one or more inputs and sums them to produce an output . Each of neurons has some weight associated with it.



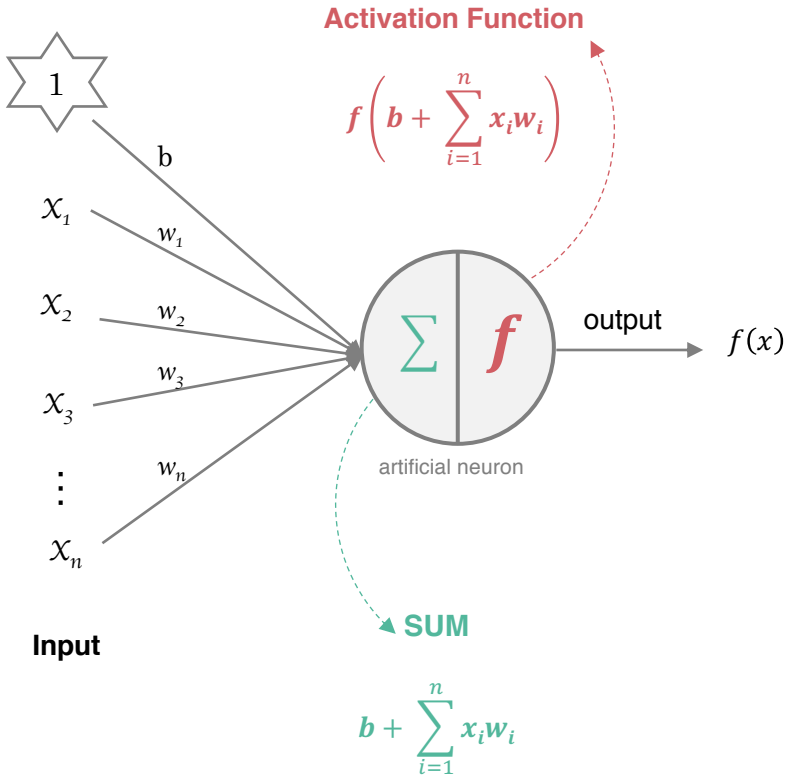
Basic structure of an artificial neuron

- Usually each input is separately weighted, and the sum is passed through a non-linear function known as an activation function or transfer function. [1]

1 source :https://en.wikipedia.org/wiki/Artificial_neuron

:: Neural Network

How does artificial neuron work ? - continued



1 **All the inputs x are multiplied with their weights w, and added together**

$$weighted\ sum = b*1 + x_1*w_1 + \ldots + x_n*w_n = b + \sum_{i=1}^{n} x_i w_i$$

2 **And then the weighted sum is passed to activation function**

The activation function is set of the transfer function used to get desired output. There are many types of activation function. One of the simplest would be step function.

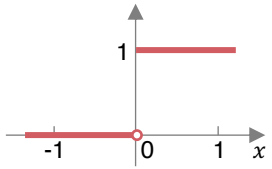
$$f(x) = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

$$Step\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

e.g. use **Heaviside Step function** as Activation Function

A step function will typically output a 1 if the input is higher than a certain threshold, otherwise it's output will be 0.

$$Hevside(x) = \begin{cases} 1, & for\ x \geq 0 \\ 0, & for\ x < 0 \end{cases}$$

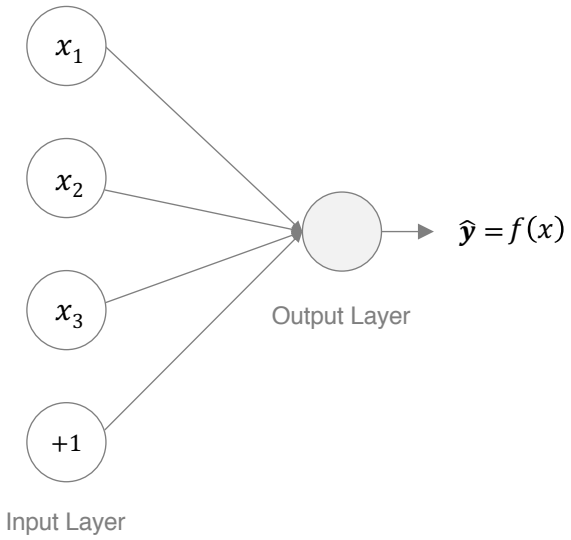


:: Neural Network

Single-layer Perceptron

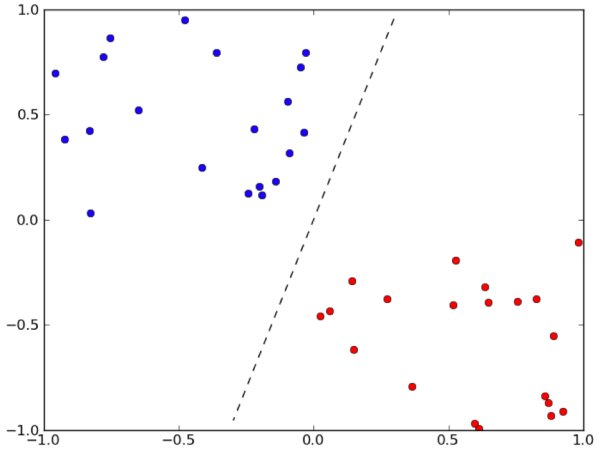
Let's take a look at Perceptron. As a linear classifier, the Perceptron is the simplest feedforward neural network. It's also termed the single-layer perceptron, to distinguish it from a multilayer neural network. It does not contain any hidden layers, which means it only consists of a single layer of output nodes.

Perceptron can be defined as a single artificial neuron that computes its weighted input, and uses a threshold activation function to output a quantity. It's also called as TLU (Threshold Logic Unit).



Use Perceptron to classify data

Where we use Perceptron?



Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier[1].

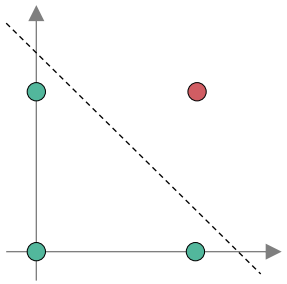
1 source :https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

:: Neural Network

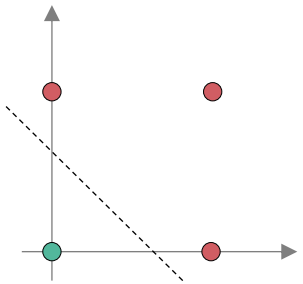
Limitation of Perceptron

A single layer perceptron can only learn linearly separable problems. If the vectors are not linearly separable, learning will never reach a point where all vectors are classified properly.

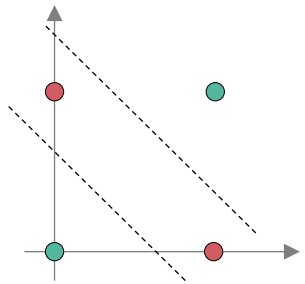
linearly separable ?



Boolean **And**



Boolean **OR**



Boolean **XOR**

A single perceptron can only learn linearly separable functions.

Boolean And function, Boolean OR function are linearly separable, whereas XOR (exclusive or) function is not linearly separable (Its positive and negative instances can't be separated by a line or hyperplane).

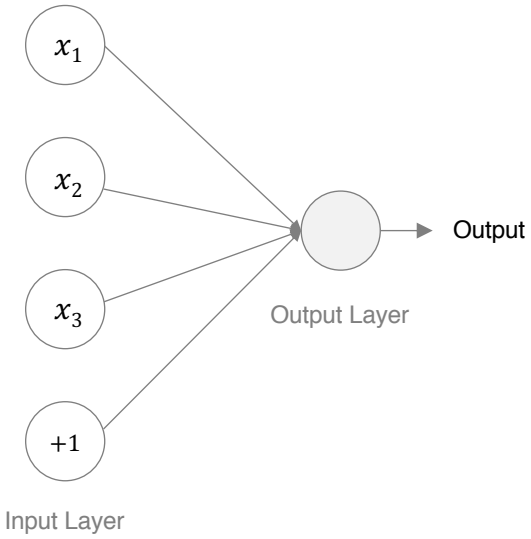
Even when the data set is linearly separable, there may be many solutions, and which one is found will depend on the initialization of the parameters and on the order of presentation of the data points. Furthermore, for data sets that are not linearly separable, the perceptron learning algorithm will never converge [1].

1 Source: Pattern Recognition and Machine Learning

:: Neural Network

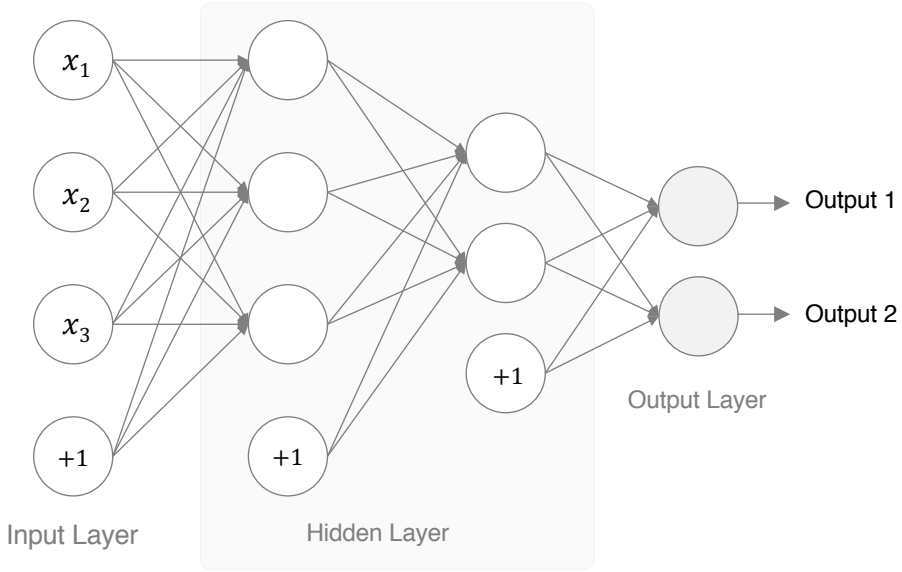
Multi-layer Perceptron can solve non-linear separable problem

A multi-layer perceptron (MLP) is a class of feedforward artificial neural network. Given a set of features $X = x_1, x_2, \ldots x_m$ and a target y, Multi-layer Perceptron can learn a non-linear function approximator for either classification or regression[1].



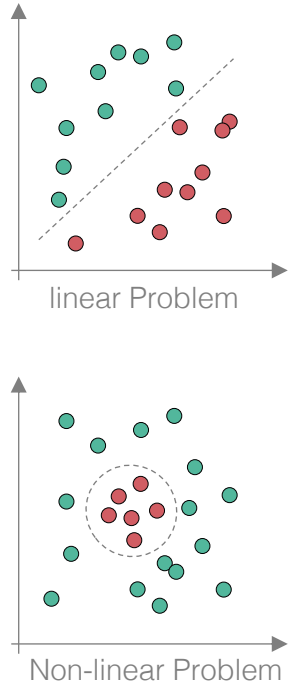
Single-Layer Perceptron

It does not contain any hidden layer. It performs binary classification (either this or that). But it can't learn non-linearly separable problem.



Multi-Layer Perceptron (MLP)

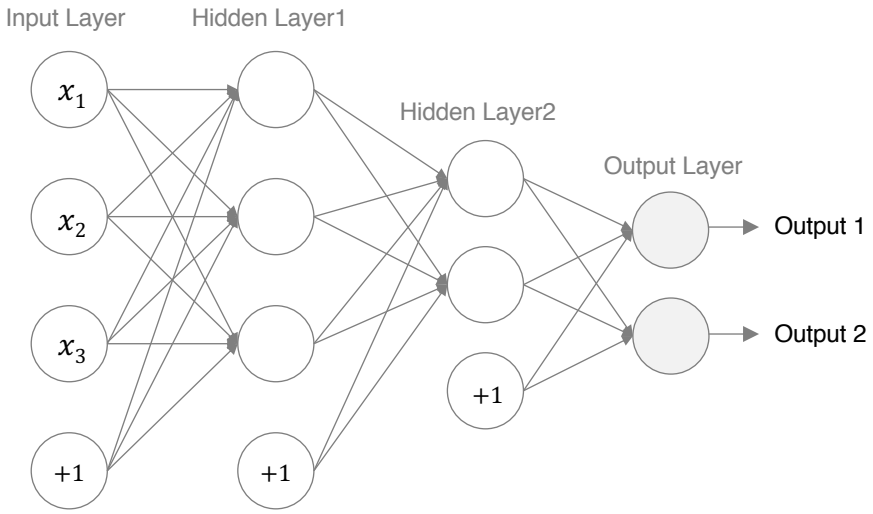
It contains hidden layers. It's able to learn not only linear problems but also non-linear problems. In most cases the data is not linearly separable. An MLP neuron is free to either perform classification or regression, depending upon its activation function[2]



1 Source: http://scikit-learn.org/stable/modules/neural_networks_supervised.html
2 Source: https://en.wikipedia.org/wiki/Multilayer_perceptron

:: Neural Network

Let’s dive into Multi-layer Perceptron



- **Input layer**

  The Input nodes provide information from the outside world to the network and are together referred to as the “Input Layer”. No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes[1].

- **Hidden Layers**

  The hidden layer is the collection of neurons which has activation function applied on it and it is an intermediate layer found between the input layer and the output layer. Its job is to process the inputs obtained by its previous layer[2]. They perform computations and transfer information from the input nodes to the output nodes[3]. Also there can be multiple hidden layers in a Neural Network.

- **Output Layer**

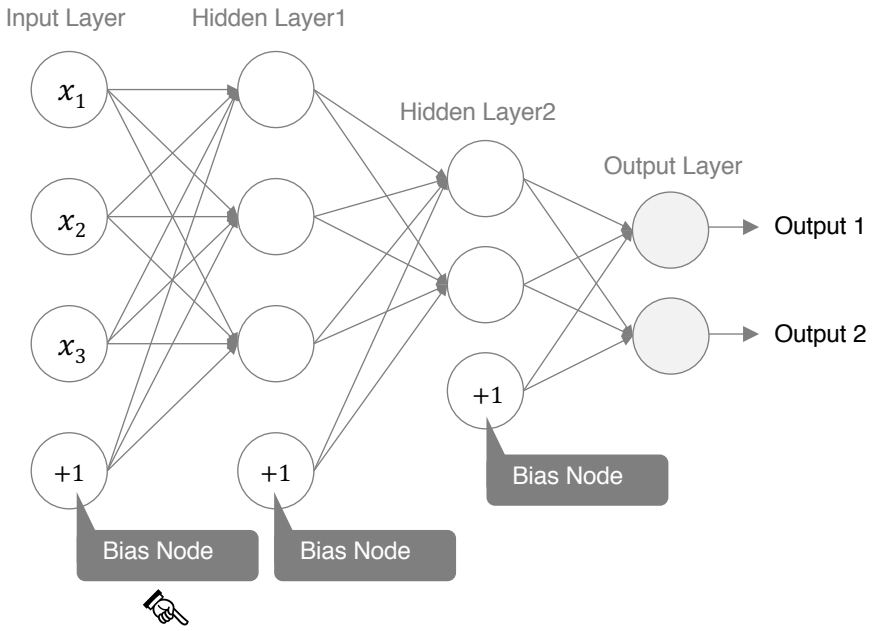
  The Output nodes are collectively referred to as the “Output Layer” and are responsible for computations and transferring information from the network to the outside world[4].

1 Soure: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
2,4 Source: https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c
3 Source: https://medium.com/@ariesiitr/an-artificial-neural-network-ann-is-a-computational-model-that-is-inspired-by-the-way-biological-c17b07166d4c

:: Neural Network

Let's dive into Multi-layer Perceptron

Bias Node / Bias Unit are included in the input layer and hidden layers. The Bias node has a value of 1.



Biases are added to the sums calculated at each node (except input nodes) during the feedforward phase.

$$b + \sum_{i=1}^{n} x_i w_i$$

Bias nodes are added to increase the flexibility of the model to fit the data

The introduction of Bias neurons allows you to shift the transfer function curve horizontally (left/right) along the input axis while leaving the shape/curvature unaltered. This will allow the network to produce arbitrary outputs different from the defaults and hence you can customize/shift the input-to-output mapping to suit your particular needs[1]. The negative of a bias is sometimes called a threshold (Bishop, 1995a).

A simpler way to understand what the bias is: it is somehow similar to the constant *b* of a linear function $y = ax + b$. It allows you to move the line up and down to fit the prediction with the data better. Without *b* the line always goes through the origin (0, 0) and you may get a poorer fit[2].

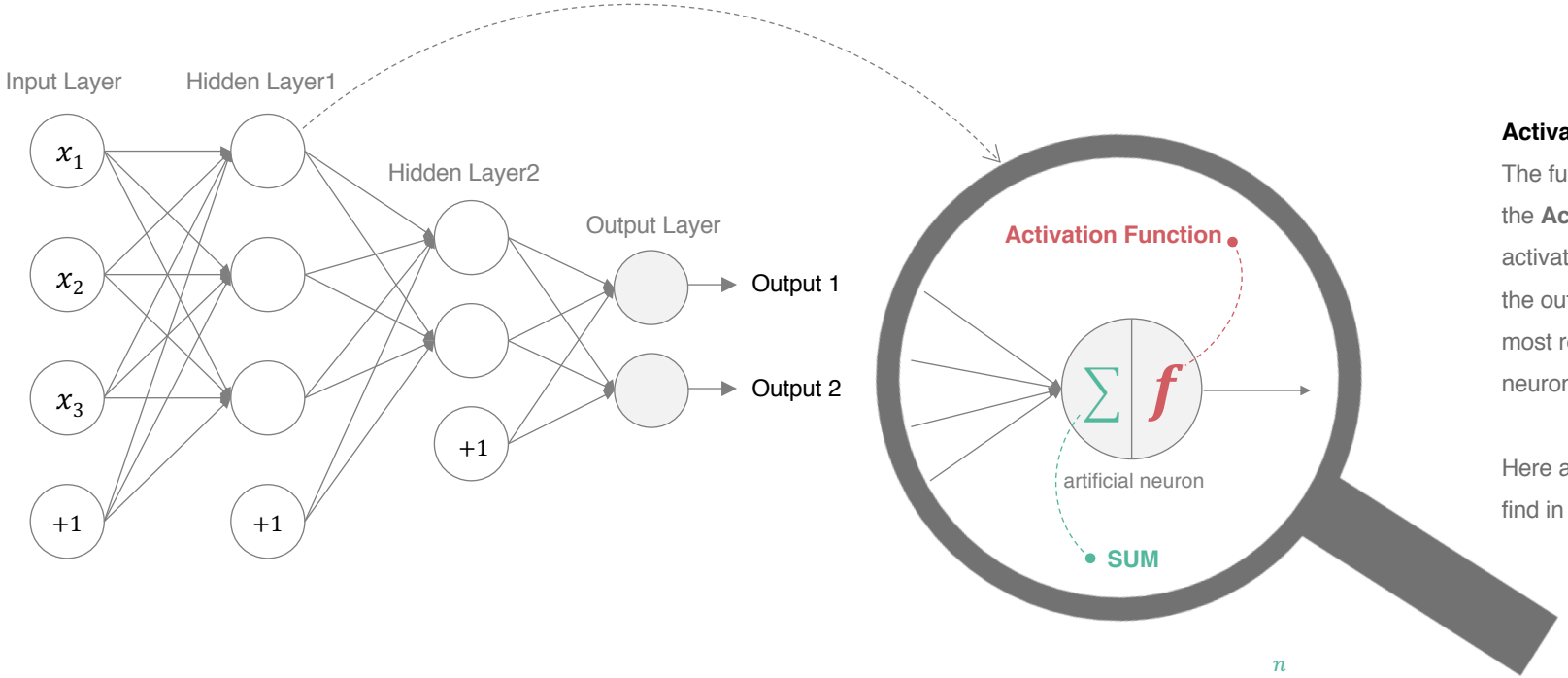
For more detailed explanation on why Bias is used, Click [this link]

1 source: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks
2 source: https://stats.stackexchange.com/questions/185911/why-are-bias-nodes-used-in-neural-networks

:: Neural Network

Let's dive into Multi-layer Perceptron

The activation function of a node defines the output of that node. Every activation function takes a single number and performs a certain fixed mathematical operation on it.



weighted sum = $b + \sum_{i=1}^{n} x_i w_i$

Activation Function

The function ***f*** is non-linear and is called the **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real world data is non linear and we want neurons to *learn* these non linear representations.

Here are some activations functions you will often find in practice:

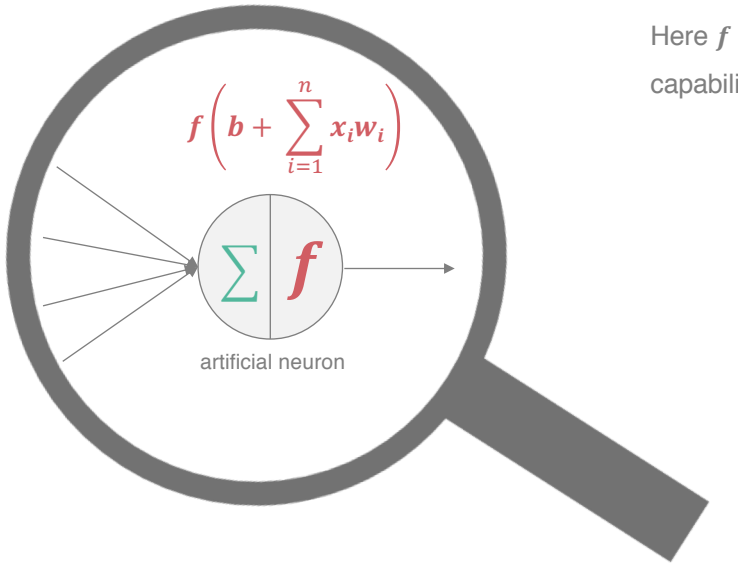
- **Sigmoid**
- **Tanh**
- **ReLU**

1 source: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks
2 source: https://stats.stackexchange.com/questions/185911/why-are-bias-nodes-used-in-neural-networks

:: Neural Network

Why is Activation Functions used in neural networks?

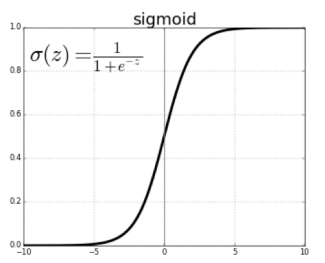
- Most important reason is to **introduce nonlinearity** to the network *.
- A neural network without a non-linear activation function is essentially just a linear regression model which is less useful
- The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks[1].



Here f is known an **activation function**. This makes a Neural Network extremely flexible and imparts the capability to estimate complex non-linear relationships in data [3].

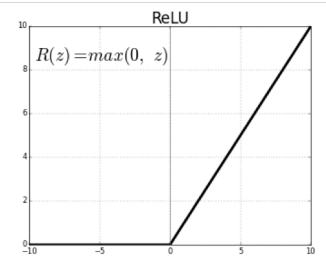
For example, we can use Sigmoid or ReLu as activation function

$f(z)$ ➡



$\sigma(z)$

Or



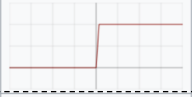
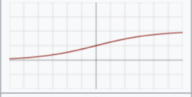
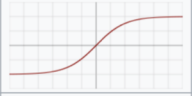
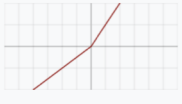
$relu(z)$

Note: The activation function can be a linear function (which represents straight line or planes) or a non-linear function (which represents curves). Most of the time, the activation functions used in neural networks will be non-linear.

[2]Source: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
[1, 3] Source: https://www.analyticsvidhya.com/blog/2016/03/introduction-deep-learnfundamentals-neural-networks

:: Neural Network

Some activation functions

Name	Plot	Equation	Derivative (with respect to x)	Range
Identity function		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified linear unit (ReLu)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Parameteric rectified linear unit (PReLU)		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Randomized leaky rectified linear unit (RReLU)		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \text{ [1]} \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Exponential linear unit (ELU)		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\alpha, \infty)$

- **Identify function / linear activation function**

  The function is a line or linear. The function is then used to implement a standard linear regression model for basic forecasting problems and therefore rarely used in neural networks[1]

- **Step Function**

  A step function originally used in a perceptron. The binary step function is extremely simple. It can be used while creating a binary classifier, which outputs 1 if the input is above threshold and otherwise outputs 0 .

  The gradient of the step function is zero. So it's not useful for back-propagation algorithm of neural network

Next let's look into the Most popular types of Activation functionsin more details

- Sigmoid (Logistic)
- TanH
- ReLU
- SoftMax

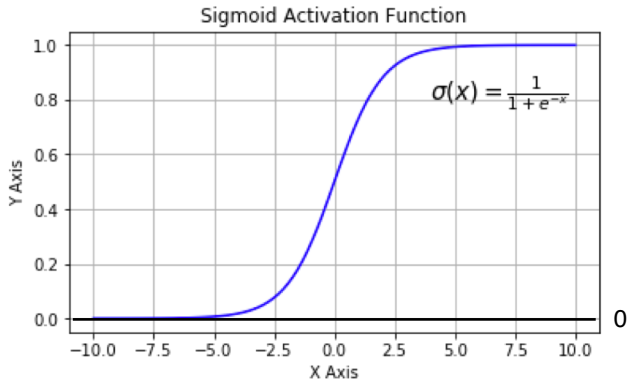
1 http://www.srutisj.in/blog/research/designing-neural-network-what-activation-function-you-plan-to-implement/

Source of table: https://en.wikipedia.org/wiki/Activation_function

Source of picture(of girl) : https://pngtree.com/freepng/a-girl-looking-through-a-telescope_3335998.html

:: Neural Network

Most popular types of Activation functions- Sigmoid



0

The Sigmoid Function curve looks like a S-shape. It is also known as Logistic Activation Function.

Mathematically it is represented as

$$f(x) = \frac{1}{1 + e^{-x}}$$

This is a smooth function and is continuously differentiable which is is necessary for enabling gradient-based optimization methods.

Sigmoid takes a real-valued number and “squashes” it into range between **0 and 1**. In particular

- large negative numbers become 0
- large positive numbers become 1

It is also used in the output layer where our end goal is to predict probability. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks [1] :

- **Vanishing gradients** (*Sigmoids saturate and kill gradients*)

  The neuron’s activation saturates at either tail of 0 or 1. The curve is flat near 0 and 1. The gradient at these regions is almost zero. if the local gradient is very small, it will effectively “kill” the gradient during backpropagation[2]. Thus, the weights in these neurons do not update. Not only that, the weights of neurons connected to such neurons are also slowly updated. This problem is also known as vanishing gradient [3].

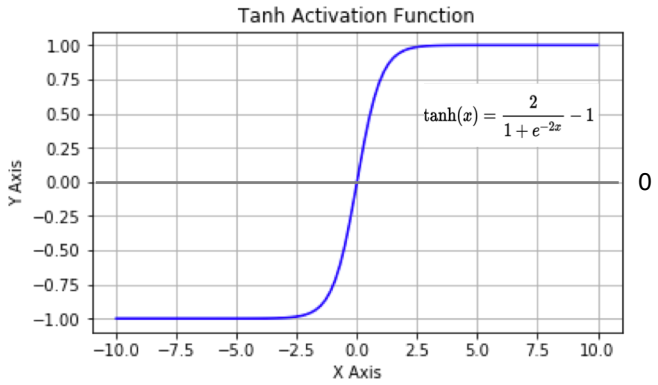
- **Sigmoid outputs are not zero-centered**

  This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights [4]

1 2,,4 source: http://cs231n.github.io/neural-networks-1/
3 Source: https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

:: Neural Network

Most popular types of Activation functions- Tanh



0

Tanh function is also like sigmoid but better.

- The range of tanh is from (-1 to 1)
- The range of sigmoid is from (0 to 1)

Mathematically it is represented as

$$Tanh\ (x) = 2 \cdot \sigma\ (2x) - 1$$

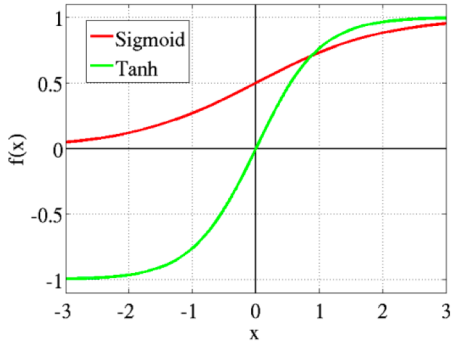
- Where σ(x) is the sigmoid function

You can think of a tanh function as two sigmoids put together[2].

Tanh function takes a real-valued input and “squashes” it into range between **-1 and 1**.

Unlike sigmoid, tanh outputs are zero-centered since the scope is between -1 and 1. The negative inputs considered as strongly negative, zero input values mapped near zero, and the positive inputs regarded as positive[1].

In practice, tanh is preferable over sigmoid



The tanh function also suffers from the vanishing gradient problem and therefore kills gradients when saturated [3].

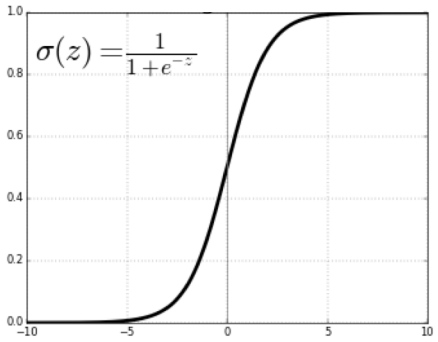
1,2,3 Source: https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/
Source of picture : https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

:: Neural Network

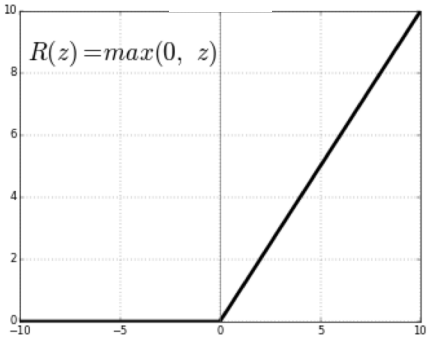
Most popular types of Activation functions- ReLU

The ReLU (Rectified Linear Unit) is the **most used activation** function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning. It takes a real-valued input and thresholds it at zero (replaces negative values with zero)

Sigmoid



ReLU



the ReLU is half rectified (from bottom)

$$f(x) = \max(0, x)$$

This means that when the input x < 0 the output is 0 and if x > 0 the output is x. This activation makes the network converge much faster. It does not saturate which means it is resistant to the vanishing gradient problem at least in the positive region (when x > 0), so the neurons do not backpropagate all zeros at least in half of their regions.

ReLU is computationally very efficient because it is implemented using simple thresholding.

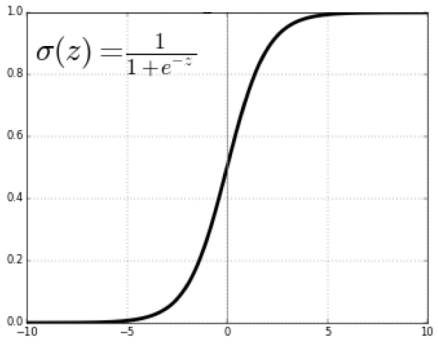
1 2,,4 source: http://cs231n.github.io/neural-networks-1/
3 Source: https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

:: Neural Network

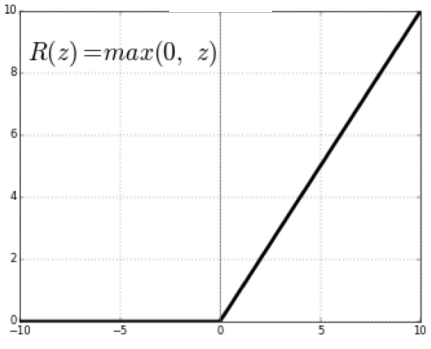
Most popular types of Activation functions- ReLU(continued)

But there are few drawbacks of ReLU neuron :

Sigmoid



ReLU



the ReLU is half rectified (from bottom)

$$f(x) = \max(0, x)$$

- **Not zero-centered**

  The outputs are not zero centered similar to the sigmoid activation function

- **Unfortunately, ReLU units can be fragile during training and can "die"**

  The issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. if $x < 0$ during the forward pass, the neuron remains inactive and it kills the gradient during the backward pass. Thus weights do not get updated, and the network does not learn. For example, you may find that as much as 40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high

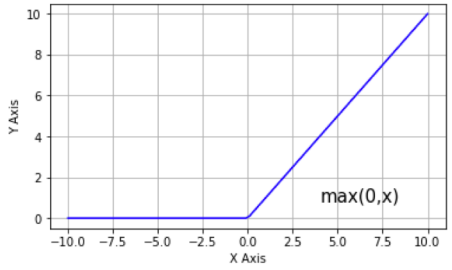
1 2,,4 source: http://cs231n.github.io/neural-networks-1/
3 Source: https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

:: Neural Network

Most popular types of Activation functions- Leaky ReLU

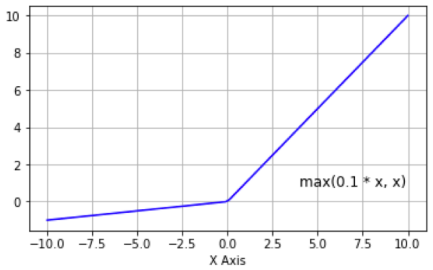
To address the vanishing gradient issue in ReLU activation function when x < 0, we have something called **Leaky ReLU** which was an attempt to fix the dead ReLU problem.

ReLU



$$f(x) = \max(0, x)$$

Leaky ReLU



$$f(x) = \max(0.1x, x)$$

$$f(x) = \max(0.1x, x)$$

Leaky ReLU is one attempt to fix the “dying ReLU” problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small negative slope (of 0.01, or so)[1].

Some people report success with this form of activation function, but the results are not always consistent[2].

The idea of leaky ReLU can be extended even further. Instead of multiplying x with a constant term we can multiply it with a hyperparameter which seems to work better the leaky ReLU.

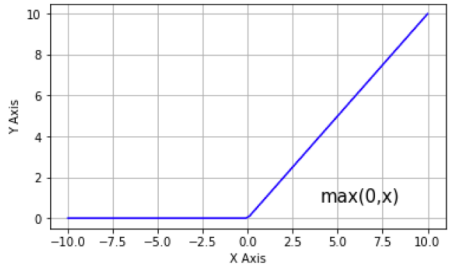
1 2, source: http://cs231n.github.io/neural-networks-1/
3,4 Source: https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

:: Neural Network

Most popular types of Activation functions- PReLU

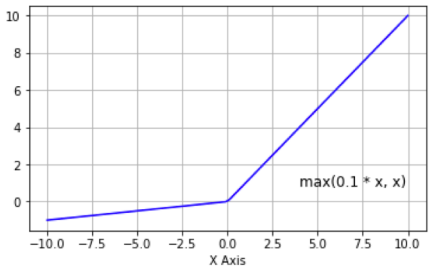
The idea of leaky ReLU can be extended even further. Instead of multiplying x with a constant term we can multiply it with a hyperparameter which seems to work better the leaky ReLU. This extension to leaky ReLU is known as **PReLu** (Parametric ReLU).

ReLU



$$f(x) = \max(0, x)$$

Leaky ReLU



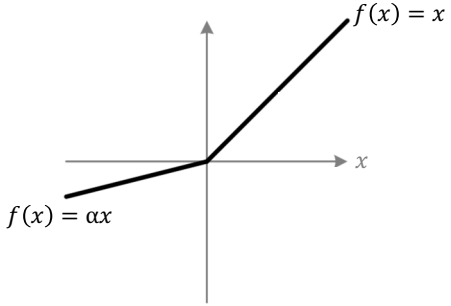
$$f(x) = \max(0.1x, x)$$

$$f(x) = \max(\alpha x, x)$$

Where α is a hyperparameter which can be learned since you can backpropagate into it.

This gives the neurons the ability to choose what slope is best in the negative region, and with this ability, they can become a ReLU or a leaky ReLU[4].

PReLU



$$f(x) = \max(\alpha x, x)$$

In summary, it is better to use ReLU, but you can experiment with Leaky ReLU or Parametric ReLU to see if they give better results for your problem

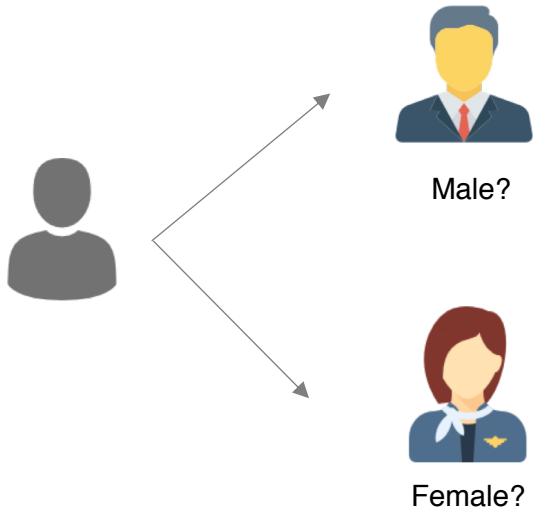
1 2, source: http://cs231n.github.io/neural-networks-1/
3,4 Source: https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

:: Neural Network

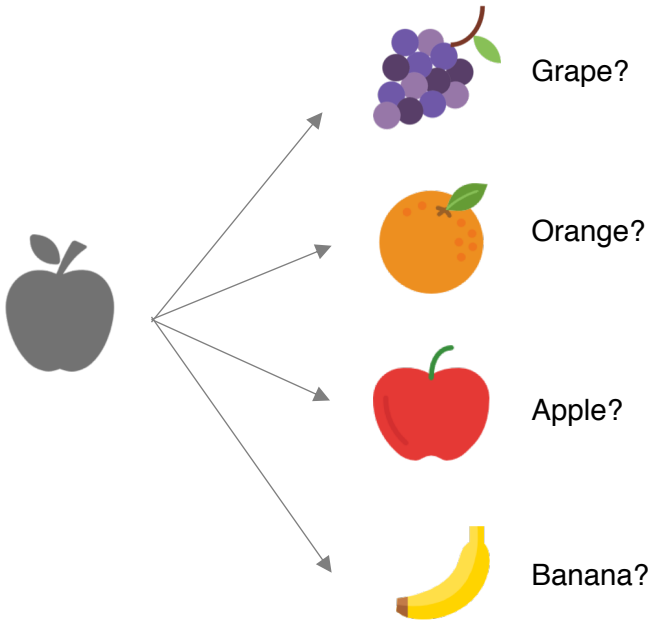
Most popular types of Activation functions– SoftMax

The softmax function is used in various **multiclass classification** methods, such as softmax regression, naive Bayes classifiers, and artificial neural networks.

Binary Classification



Multiclass Classification

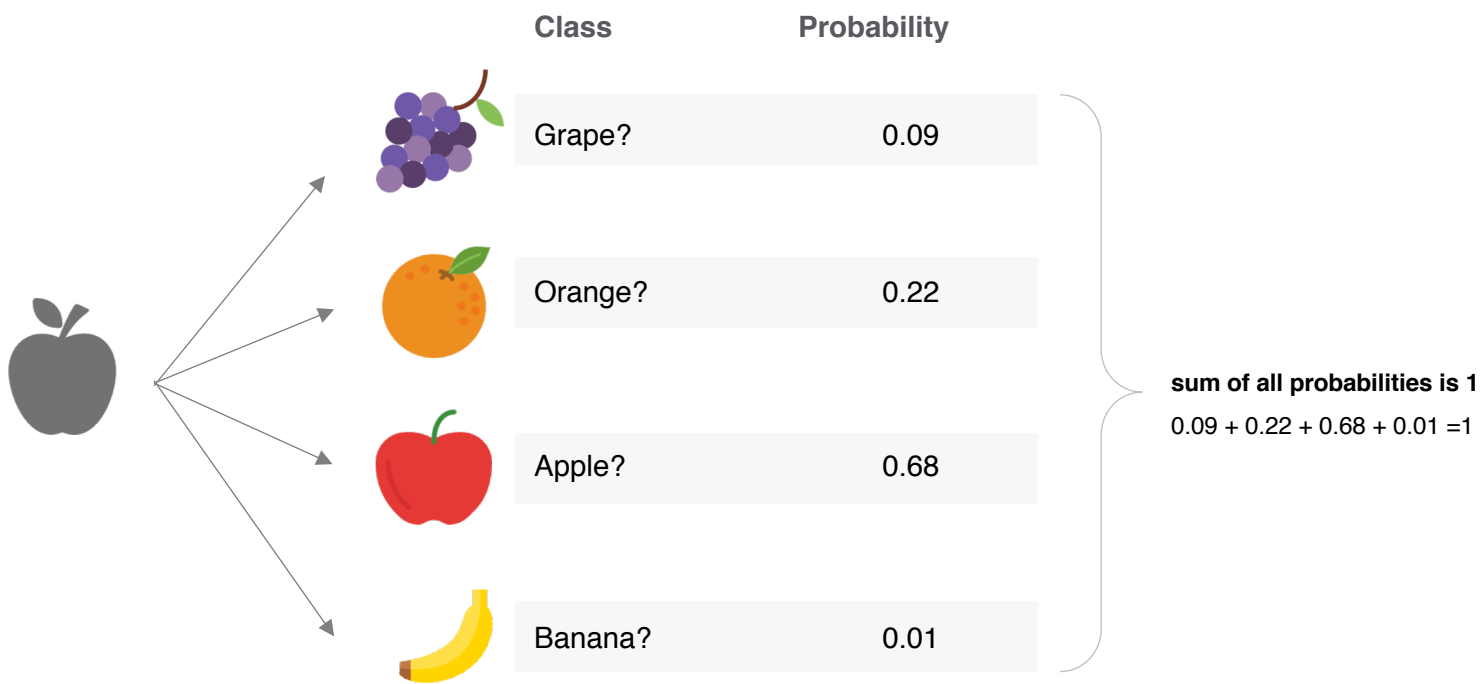


:: Neural Network

Most popular types of Activation functions– SoftMax (continued)

Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. [1] The softmax function is often used in the final layer of a neural network-based classifier [2]. The Softmax layer must have the same number of nodes as the output layer [3].

Softmax might produce the following likelihoods of an image belonging to a particular class:



1,3 Source https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax
2 Source: https://en.wikipedia.org/wiki/Softmax_function

:: Neural Network

Most popular types of Activation functions– SoftMax (continued)

The Softmax function is a generalization of Logistic Regression. It converts linear inputs to probabilities. Softmax is implemented through a neural network layer just before the output layer [1].

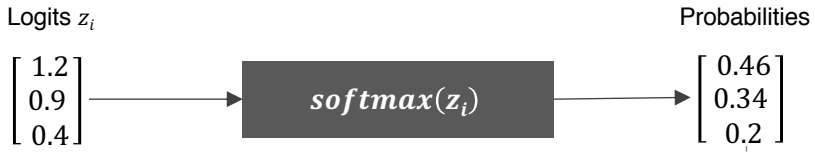
- **Logistic Function**

  It's used for binary classification

$$f(x) = \frac{1}{1 + e^{-x}}$$

- **SoftMax Function**

  It's used in multiclass classification



$$\hat{p}_i = softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

for $i = 1, \ldots, j$

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a Logistic/Sigmoid function.

Unlike logistic regression which is for the binary classification, softmax allows for classification into any number of possible classes. Softmax classifier provides "probabilities" for each class. The total sum of probabilities for each class is equal to 1. In this example, 0.46+0.34+0.2 =1

1 Source: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

:: Neural Network

Most popular types of Activation functions– SoftMax (continued)

Let's see an example on training a network to recognize and classify handwritten digits from images. The network would have ten output units, one for each digit 0 to 9. Then if you fed it an image of a number 9 (see below), the output unit corresponding to the digit 9 would be activated.

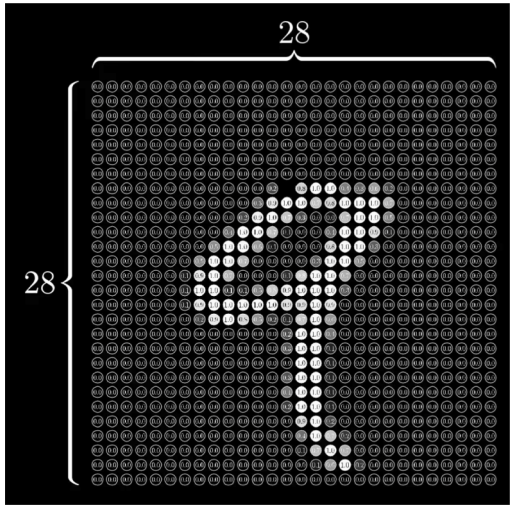
Example

Handwritten digits

Famous MNIST dataset for handwritten digits recognition

Each image is 28 pixels by 28 pixels which has been flattened into 1-D array of size 784.

We generally think of MNIST data as being a collection of 784-dinenbsional vectors



28x28 pixels = 784

:: Neural Network

Most popular types of Activation functions– SoftMax (continued)

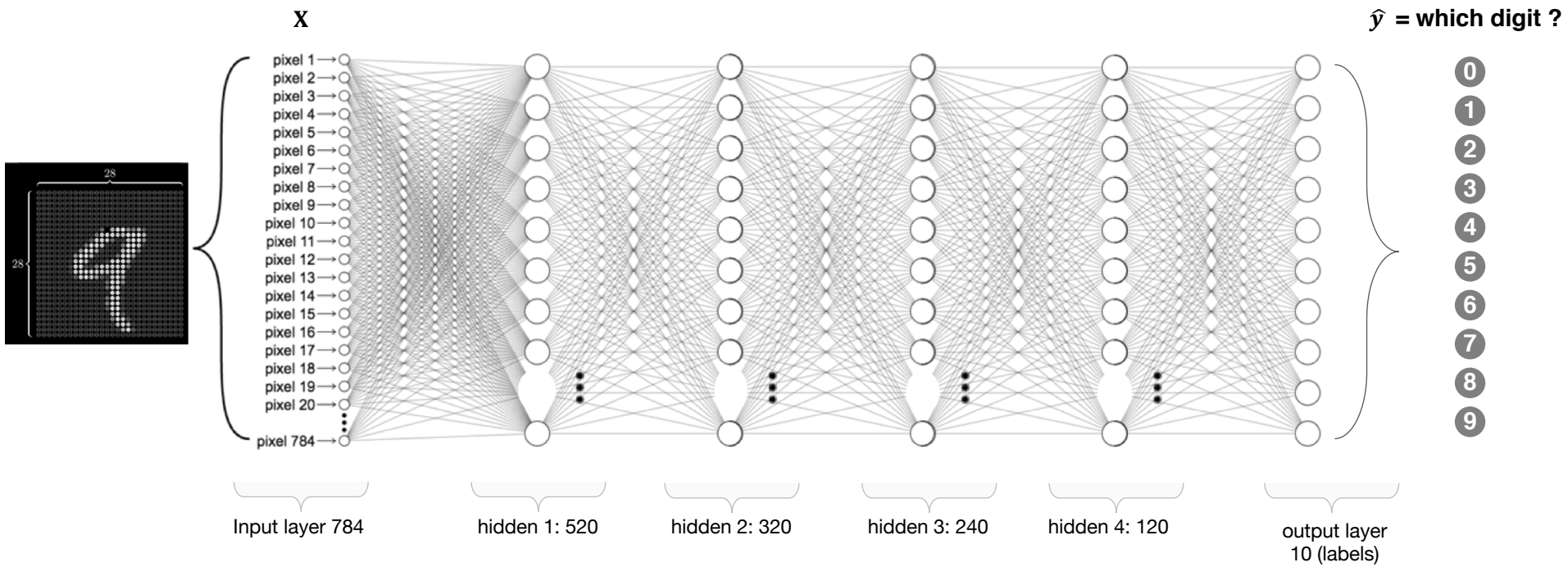
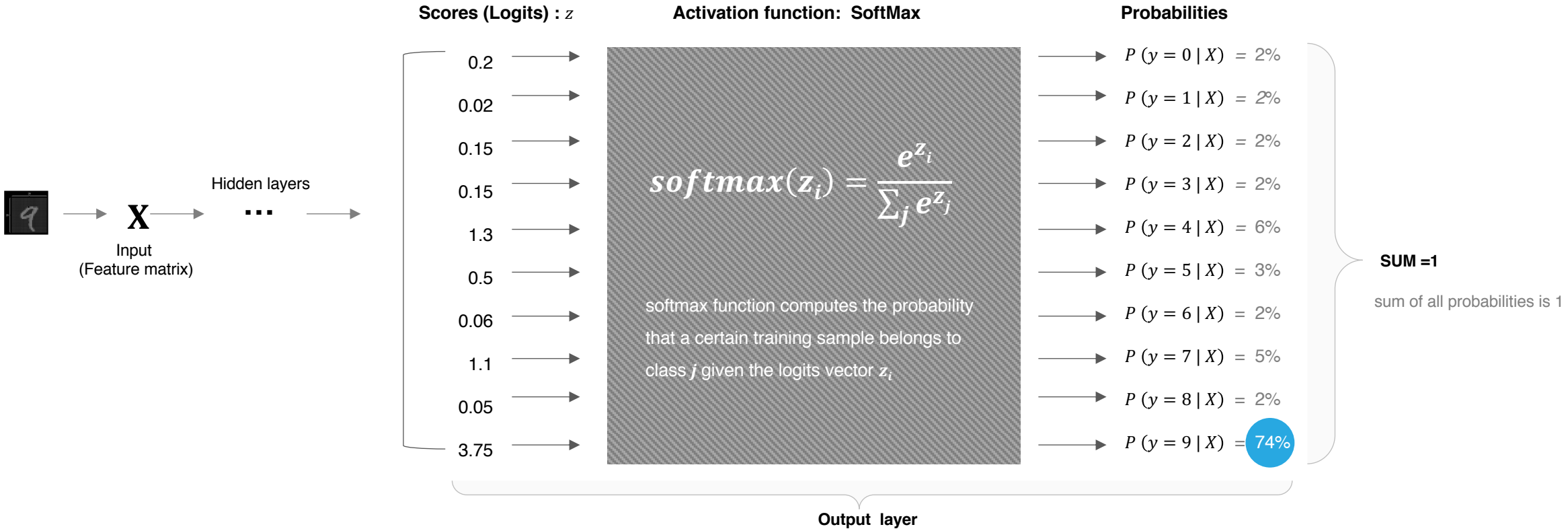


Image: http://bit.ly/PyTorchZeroAll

:: Neural Network

Most popular types of Activation functions– SoftMax (continued)

In order to convert the score matrix Z to probabilities, we use Softmax function.

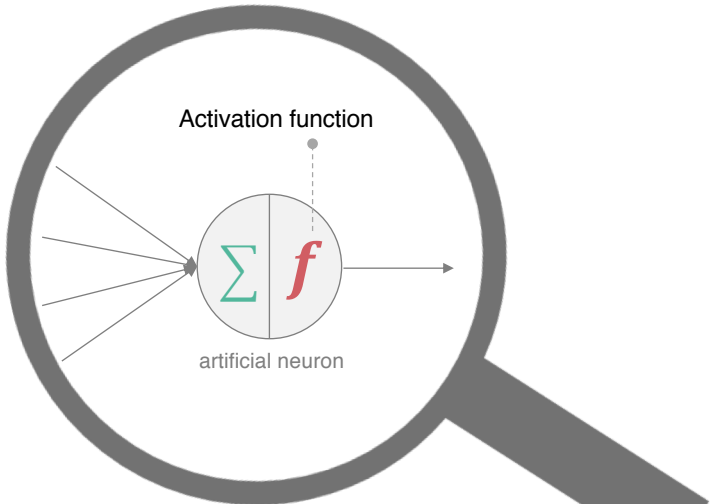


z_i is a vector of the inputs to the output layer (if you have 10 output units, then there are 10 elements in z) [1]. j indexes the output units, so j = 1, 2, ..., i. Softmax function provides us with a vector of probabilities of each class label for a given observation. The image looks the most like the digit 9 because the probability of being 9 is highest

1 source: https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions

:: Neural Network

Which Activation Functions $f(z)$ Should I use ?



$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$f(z) = f(b + \sum_{i=1}^{n} x_i w_i)$$

In Hidden Layer

- By Default use **ReLu** function
- In case that ReLu doesn't provide the optimum results, then try other activations like Leaky Relu or Maxout function
- Always keep in mind that ReLU function should only be used in the hidden layers
- Never use **Sigmoid and TanH** fucntion due to the vanishing gradient problem

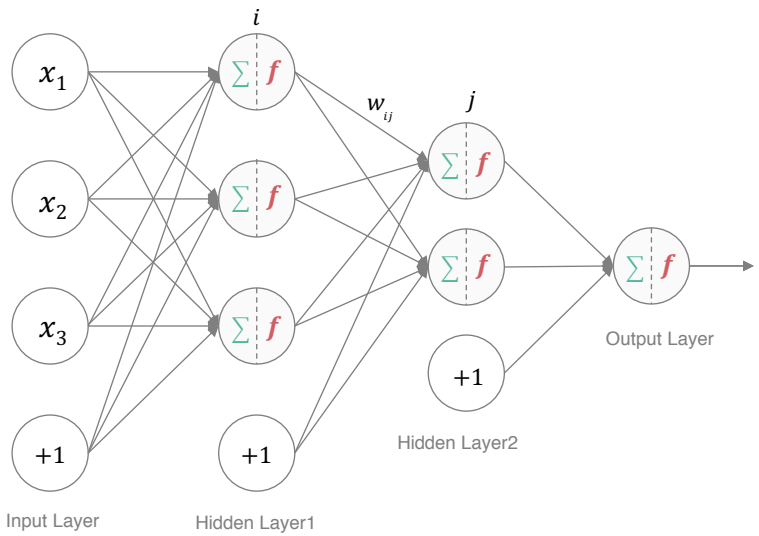
In Output Layer

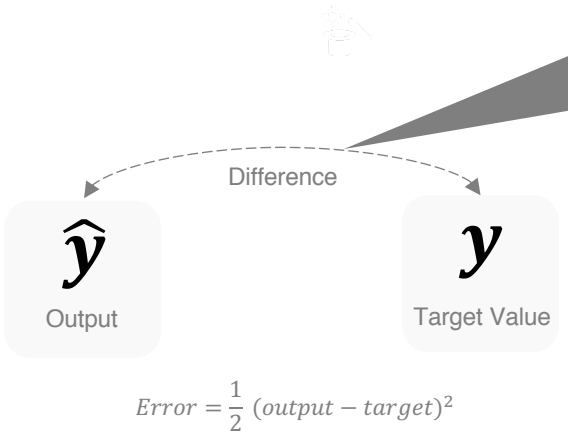
- Use **SoftMax function** for classification problem
- Use **Linear function** for regression problem

:: Neural Network

How does the network learn ?

We'll use sum of square errors, the difference between target (known) data and the value estimated by our network, to compute an overall cost and we'll try to minimize it [1]. In a neural network, we're trying to minimize the error with respect to the weights. We want to find the optimal combination of weights and bias that minimize the loss function over the training set.





$$Error = \frac{1}{2}\,(output - target)^2$$

Training a network is to minimize the difference

- The ultimate objective is to update the weights of the model in order to minimize the loss function J.
- The weights are updated using **backPropogation** algorithm

Loss Function / Cost Function / Error Function

$$E = \frac{1}{2}\,(y - \hat{y})^2$$

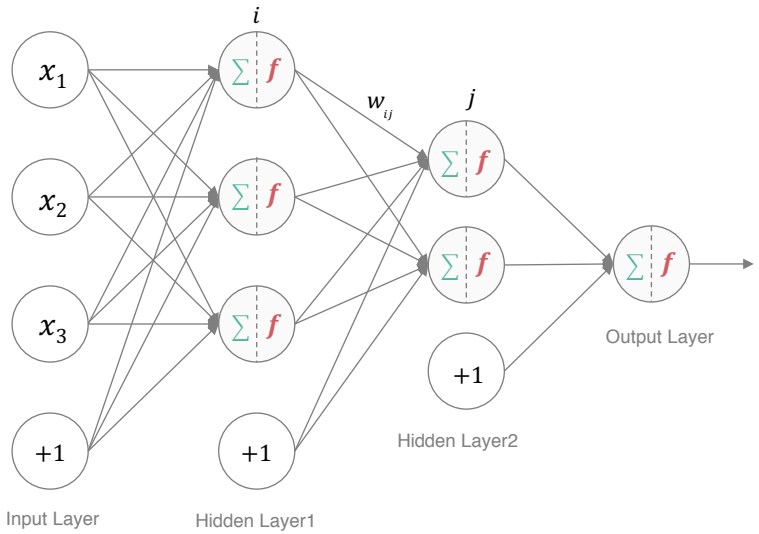
- Loss Function is used to measure exactly how wrong our predictions are.
- The loss is high when the neural network makes a lot of mistakes, and it is low when it makes fewer mistakes[2].

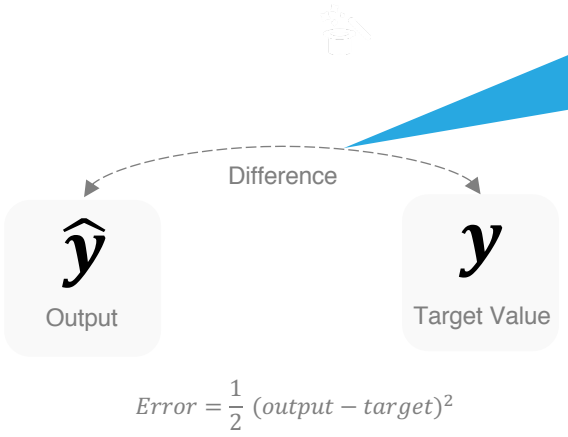
1 Source: http://www.bogotobogo.com/python/scikit-learn/Artificial-Neural-Network-ANN-4-Backpropagation.php
2 Source: https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

:: Neural Network

How does the network learn ?

We're going to make the loss function/error function as small as possible by **finding the optimal weights value**. The weights and bias inside the network get updated after many iterations.





$$Error = \frac{1}{2}(output - target)^2$$

Adjust the weights to minimize the error function

Update the weight parameters iteratively in order to make the error (loss function) as small as possible. The update equation is as follow,

$$w_{ij}^{new} = w_{ij}^{old} - \eta \frac{\partial E}{\partial w_{ij}}$$

η is the learning rate

The choice of learning rate is important, since a high value can cause too strong a change, causing the minimum to be missed, while a too low learning rate slows the training unnecessarily[1].

$\frac{\partial E}{\partial w_{ij}}$ is the partial derivatives

$\frac{\partial E}{\partial w_{ij}}$ is the partial derivatives of the loss function with respect to the weight w_{ij}

1 Source: https://en.wikipedia.org/wiki/Backpropagation