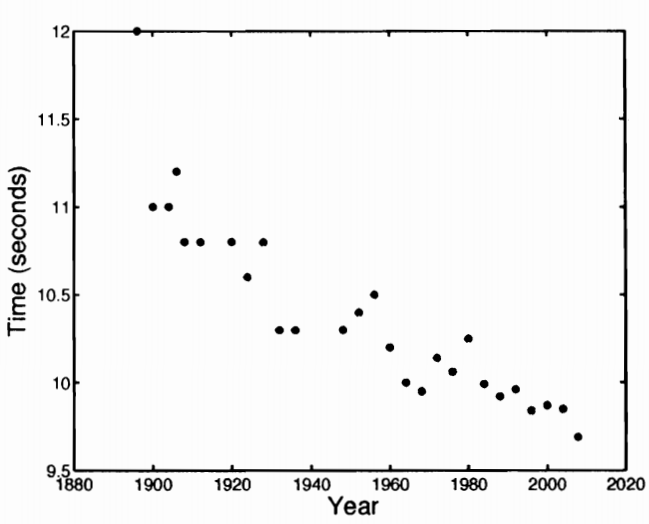
# Linear Regression

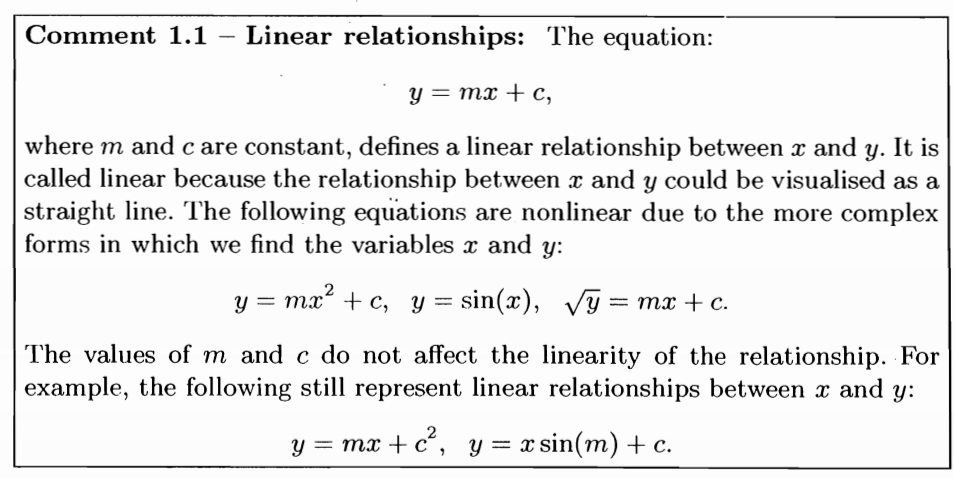
An important and general problem in Machine Learning is to find a functional relationship between a set of input variables and an output value. In other words, given a set of input variables and their output we would like to find out what the function that links them is.

As an example given the following dataset:



there exists a function f(year) such that time = f(year). In other words, the time is a function of the year. What we would like to know is to find what the function f is so that to predict future events and gain an understanding of the phenomenon.

Linear regression consists of finding the best function that fits a set of data points by using a linear model assumption. In other words, it is assumed that it is possible to fit the set of data points with a line or similarly that there exists a linear relationship between the input variables and the output variable.



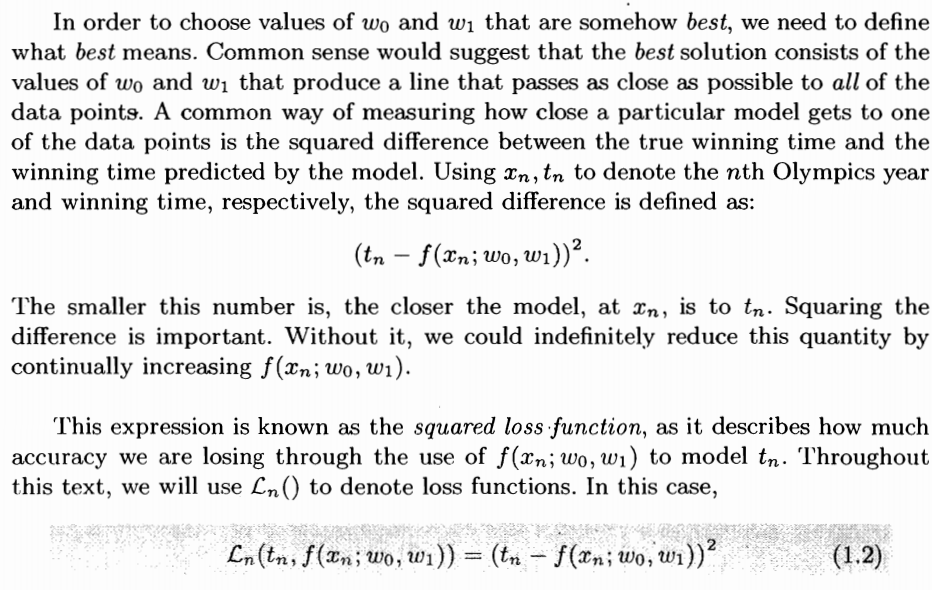
Given that we have a data set of N data points { (x1,y1), (x2,y2),…, (x,yn) }, by using linear regression we firstly assume that there exists a linear relationship between the x (input variables) and the y (output variable). Linear regression aim is to find the straight line that best fits the set of N data points and, thus we need to find a function that does so and we denote this function in the following way:

f(x ; w0,w1) = w1x + w0

such that y = f(x ; w0,w1)

The parameters w0,w1 are also called weights of the function. w0 is called the intercept of the straight line as the line resulting from the function f(x ; w0,w1) will cross the y-axis at w0. w1 is the gradient (slope) of the straight line. In order to obtain a function f(x ; w0,w1) that best fits the data, all we have to do is to tune the values of the weights.

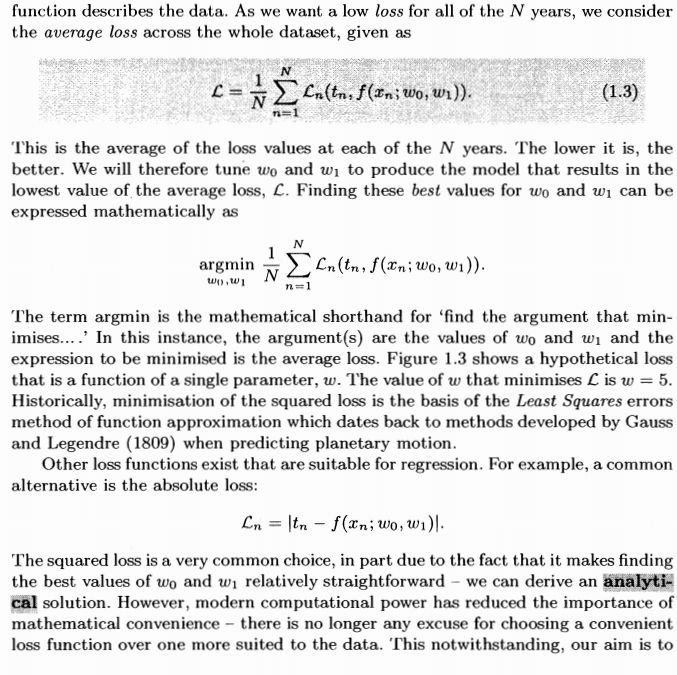
## What is meant with the best function that fits the data



The above picture illustrates that the linear function that best fits a data set is the function that passes as close as possible to all data points. Thus, we need a way to measure the difference between our current prediction and the real value for a given input. In other words, we need to find the difference between yn and f(xn ; w0,w1). Plain difference is not acceptable because if we want to minimise the difference then we could set f(xn ; w0,w1) as high as possible. One possible way is to use the absolute value of their difference or to use the squared loss function as indicated above which is guaranteed to always be positive.

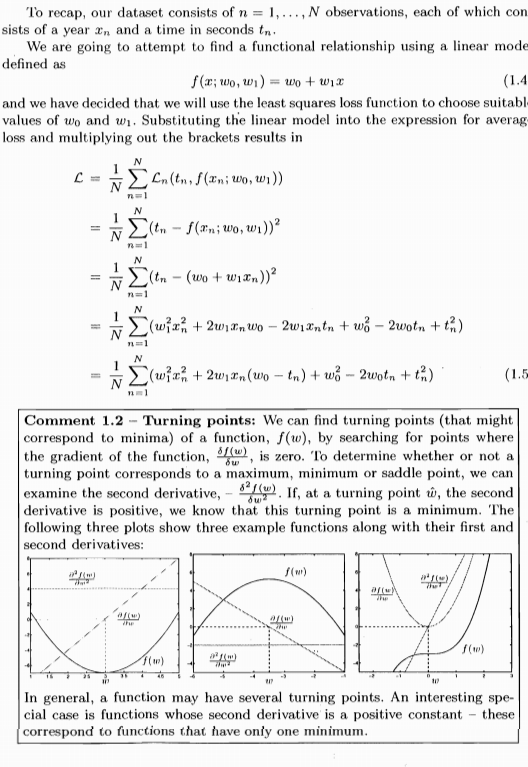
It is important to notice that the book uses tn instead of yn. It is just a different notation.

Naturally, the lower is the loss function for a given data point then the better the prediction of the function f(x ; w0,w1) is.



Therefore, in the end linear regression is just a minimisation problem because all we have to do is to find values of w0 and w1 such that the average loss function is minimised. If the average loss function is minimised then we have found the model with the lowest possible average discrepancy from its predictions and the real values.

The book uses an example for illustrating the linear regression process, as it says that N is the number of years. In general, we must think of N as the number of data points that we have in our data set.



What we obtain with equation 1.5 is a quadratic multivariable function where the variables are w0 and w1. Naturally, this function is differentiable.

By calculus III, we know that if a point is a local extrema and the function is differentiable then all partial derivatives of that point must be equal to 0. In other words, the gradient of the function at that point must be the **0** vector.

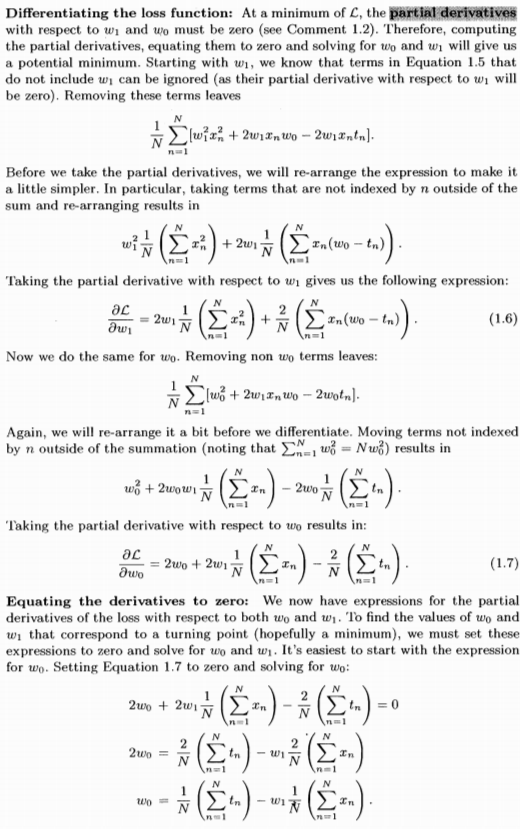
Given that the equation 1.5 tells us that the average loss function is differentiable then we must find at which points the partial derivatives are equal to 0. One of the points where the partial derivatives are equal to 0 will be our local minima.

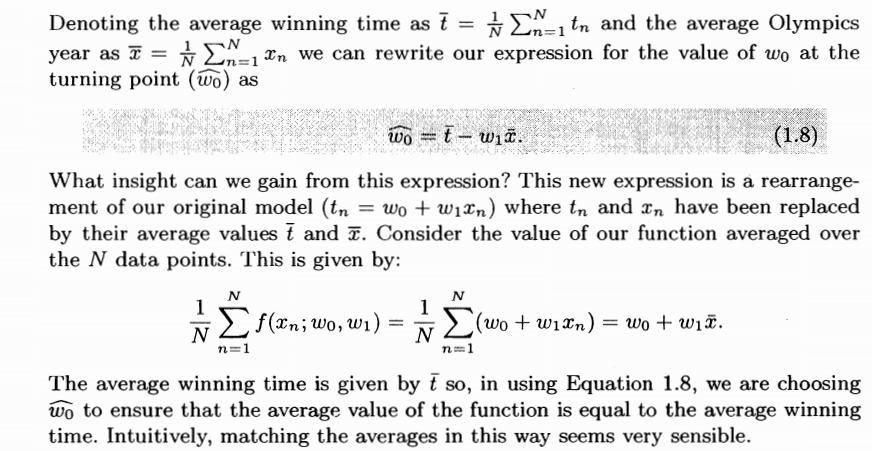
It is easy to see that the average loss function that we obtain at equation 1.5 is actually a 3D parabola as it is a quadratic multivariable function, and so it is convex. Thus, it will have only a local minimum which will also be the global minimum. Finding the local minimum by using the partial derivatives will give us the global minimum and so the best weights for the function f(x ; w0,w1).

Given that the average loss function describes a 3D parabola (convex) then the partial derivatives will be 0 only at the local (global) minimum because the partial derivatives of the average loss function with respect to w0 and w1 will describe each the slope of a 2D parabola where we know that have a slope of 0 only at their vertex. Furthermore, we know that a 3D parabola cannot have a saddle point.

In other words, all we need to know to find the minimum of the average loss function is to find where the partial derivatives are equal to 0, there will be just one point that satisfies that and this point is the global minimum which will give us the best values for the weights.

So, now let’s just find the partial derivatives of the average loss function and set them equal to 0 and let’s find out the values of the weights w0 and w1.



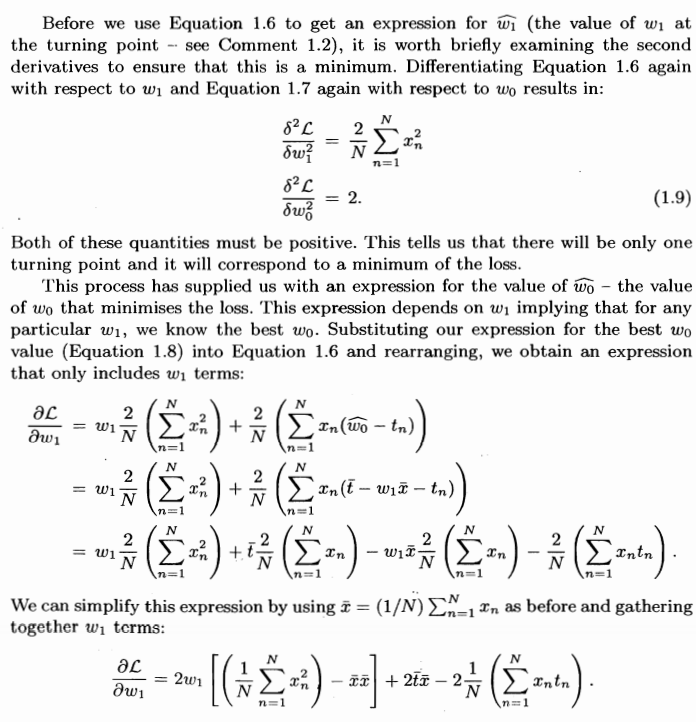


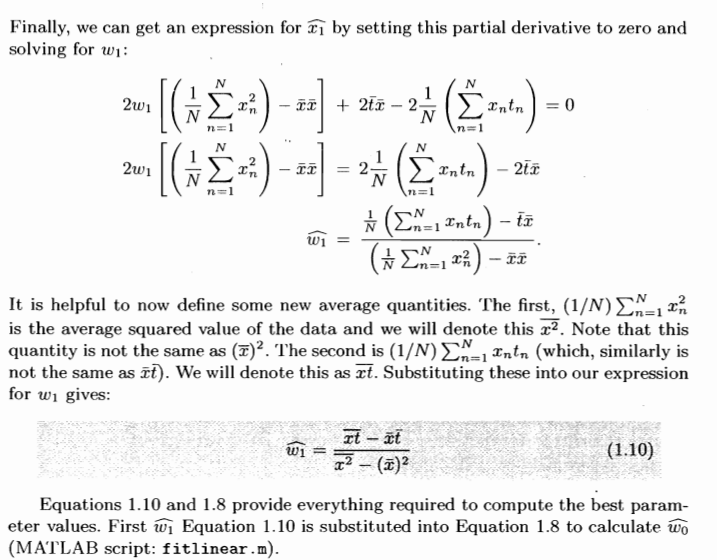
In other words, by equation the partial derivative with respect to w0 to 0, we obtain that w0 must be equal to the average of the real output values – w1 multiplied by the average of the input variable.

This makes a lot of sense because if we rearrange equation 1.8 we obtain the following:

Which says that the average value of the function f(x; w0,w1) must be equal to average value of the output variable.

Given that we want to know for what value of w0 and w1 both partial derivatives are equal to 0 then we are solving a system of two equation in two variables. Thus, having found an expression of w0 in terms of w1, what we are going to do is to replace w0 with its expression in terms of w1 in the partial derivative with respect to w1 and set it equal to 0 so that to find w1 and then consequently find w0.

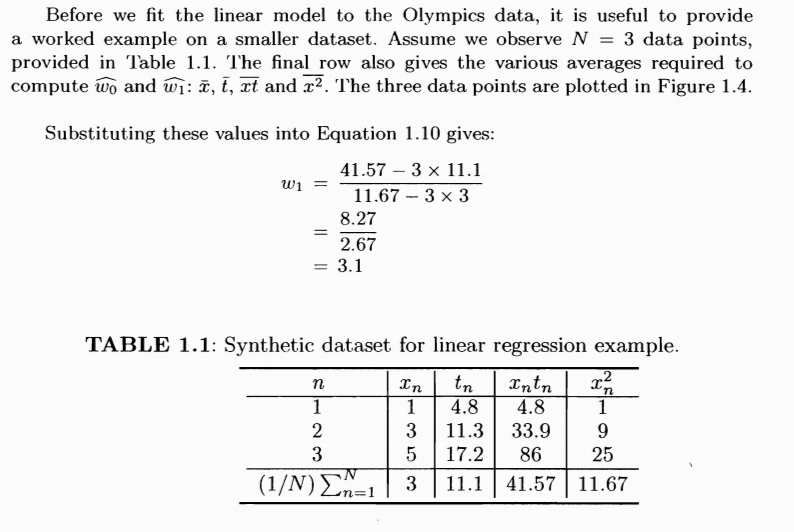


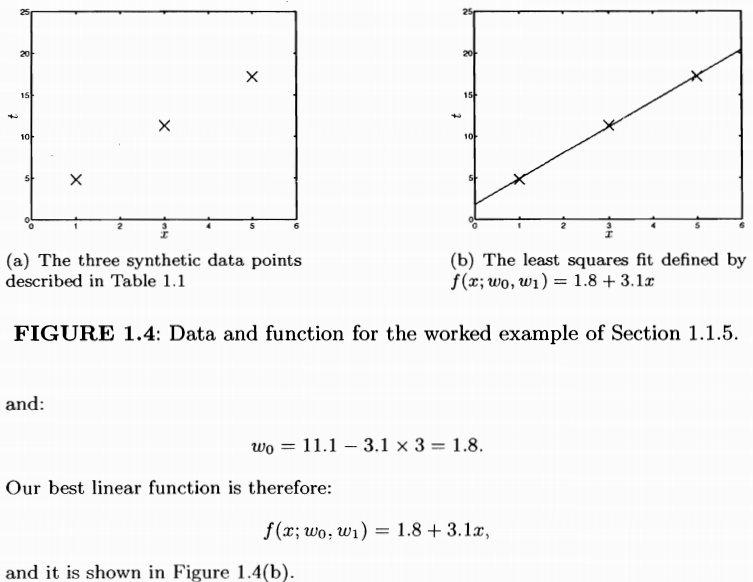


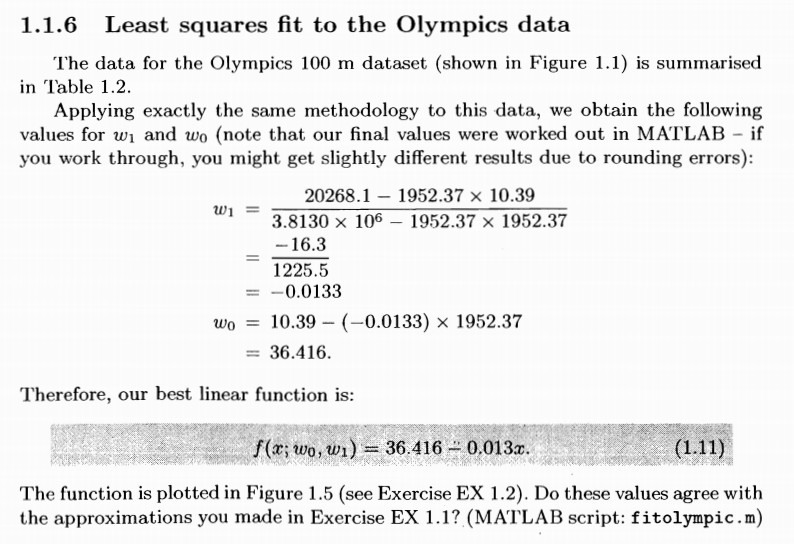
Thus, we have found out for what values of w0 and w1 we minimise the average loss function and so we obtain a straight line f(x ; w0,w1) which as close as possible to all data points.

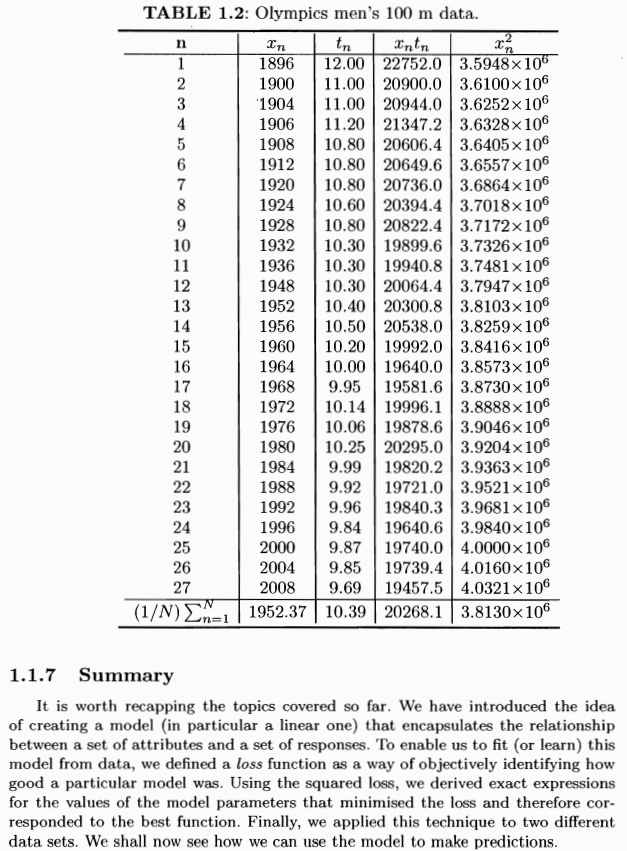
Therefore, when having a linear regression problem which involves only one input variable x all we have to do is to find w0 and w1 and we can firstly find w1 using equation 1.10 and then substitute the value obtained of w1 into equation 1.8 so that to find w0 and we have magically found the line that best fits our data set.

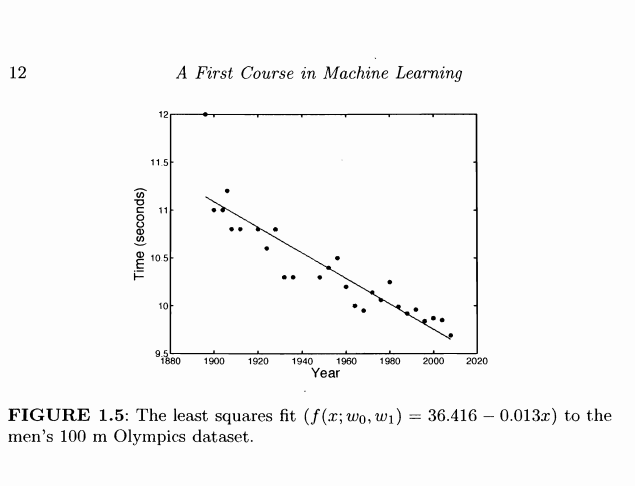
Now we are going to see some worked examples so that to gain an understanding of how to use equation 1.10 and 1.8 in practice.



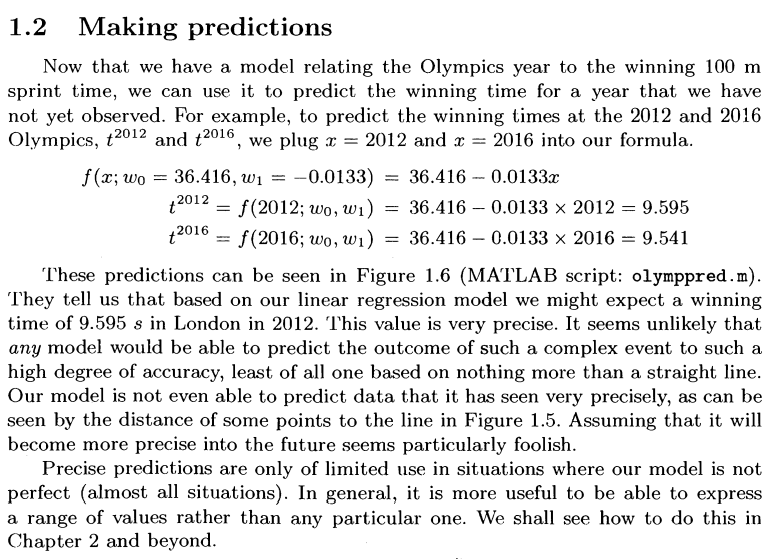








## How to make predictions given a linear model



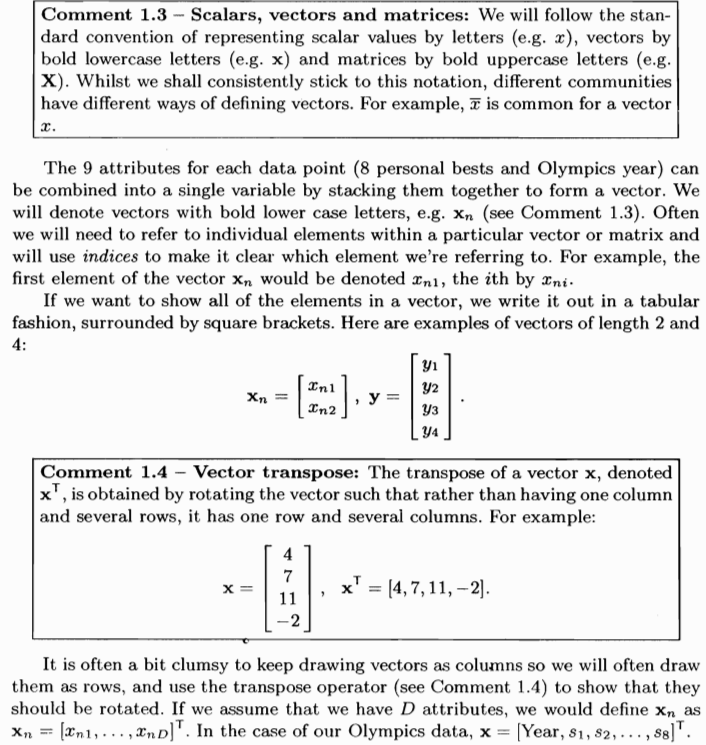
## Linear regression with multiple input variables



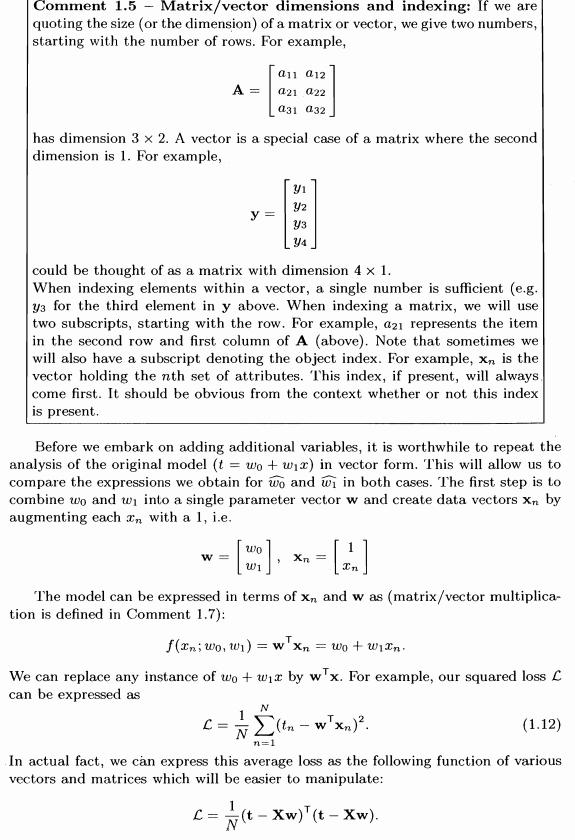
In other words, linear regression with n multiple input variables is carried out at the same way as linear regression with only one input variable. Indeed, since we need to find the weights w0, w1, …, wn all we have to do is to minimise the average loss function which will be a parabola in a n+1 space. Thus, it will always be a convex shape and so we can find the minimum of the average loss function where the partial derivatives with respect to each weight is equal to 0.

However, this is impractical when there are many n input variables as we should find the partial derivative for each of the n+1 weights and set them all equal to 0. In other words, we should solve a system of n+1 equations with n+1 variables (the weights).

Luckily, there exists a more convenient way of doing linear regression with multiple variables which is based on vectors and matrices. It is important to bear in mind that the principle for linear regression is still the same: minimise the average loss function through the use of partial derivatives. However, we will do it by using matrices and vectors.



It is important to notice that vectors by convention are meant to be column vectors. Thus, if **x** is a vector then **x**T is a row vector because by convention **x** is a column vector.





In other words, a function

f (x1,x2,…,xn ; w0,w1,…,wn) = w0 + w1x1 + w2x2 + … + wnxn

can be expressed in a convenient way in a vector form in the following way:

f (**x**;**w**) = **w**T**x**

where **w** is a column vector containing all the weights and **x** is a column vector containing all input variables. It is important to notice that **x** must be augmented by containing a starting 1 as the vector **w** contains an additional element in comparison to **x** as it contains w0.

Thus, we have the following:

* **w** = [w0,w1,…,wn]T
* **x** = [1,x1,x2,…,xn]T

What we do is the dot product between them and so we obtain w0 + w1x1 + w2x2 + … + wnxn.

Therefore,

f (**x**;**w**) = f (x1,x2,…,xn ; w0,w1,…,wn)

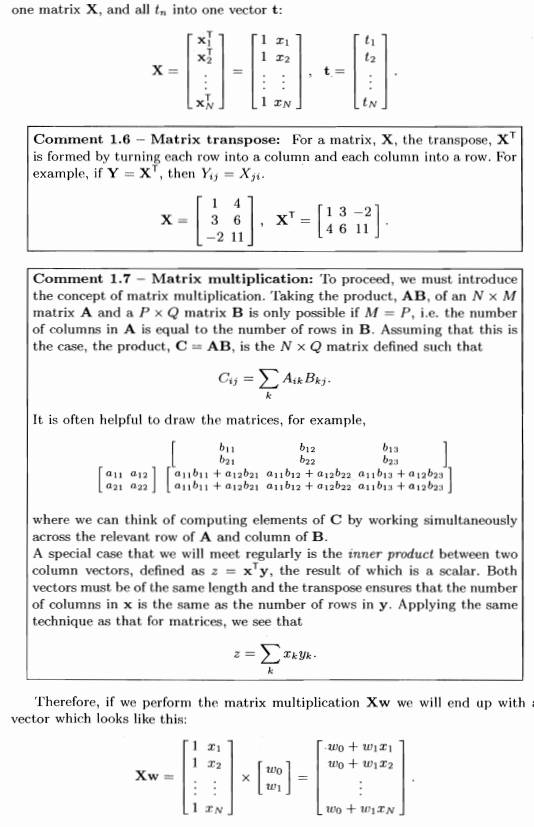
One may wonder why we need to transport **w**. The answer is that by convention the dot product is seen as a matrix by matrix computation and it can be performed only if given matrix A (MxN) and B (QxT) it follows that N = Q and the resulting matrix will be of size (MxT).

Thus, vectors are seen as matrices of size (Mx1). Therefore, given two vectors **w** and **x** of size (Mx1) respectively then in order to compute the dot product between them we need to transpose the left vector.

Therefore, we get **w**T**x** which will get us a vector as expected of size (1x1), therefore it is just a scalar as expected.

Furthermore, it follows from equation 3 that we can express the whole average loss function in terms of matrices and vectors without needing to have a summation series.

Now, we are going to see how that is possible.



In other words we can create a matrix **X** which contains at row i the vector **x**iT. In other words, each row of the matrix **X** represents a set of input values in our data set. We do not need to forget that we need to augment each set of input values with the value 1 because we need to compute the dot product with the vector **w** which contains an additional element: w0

We can create a vector **t** which represents a column vector for all output values in our data set. In other words, row i of **t** contains ti.

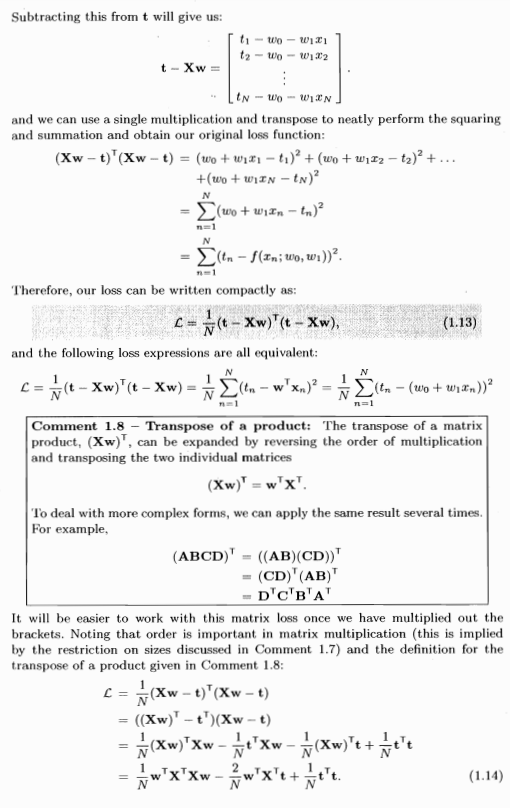
Therefore, let a data set have n input variable this means that we will have n+1 weights.

Thus, the matrix **X** has size (M x (n+1)).

The vector **w** has size ((n+1) x 1).

If we perform the matrix-vector multiplication **Xw**  we get a (Mx1) vector where each row m represents f (**xm**;**w**) = **x**mT**w.**

Therefore, each row m of the **Xw** vector represents a prediction of our linear model for the mth data point.



If we subtract the vector **t** to **Xw** we obtain a vector which contains a column vector which contains at every row tn – **x**nT**w**. In other words, each row contains the difference between the real output and the prediction of the function.

Since we compute the sum of the squared difference in the average loss function we can compute the sum of the squared difference by simply using the dot product of (**t** - **Xw**) with itself.

In other words, (**t** - **Xw**)T(**t** - **Xw**) is equal to the sum of the squared loss. All we need to do is to compute the average of that sum by simply dividing by the number of the data points in the data set.

Thus, equation 1.13 represents the equation of the average loss function by simply using matrices and vectors.

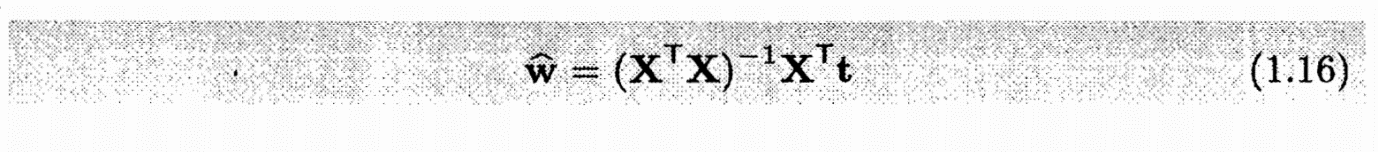
However, we can compute the value of the average loss function given **X** ,**w** and **t** in another way. Indeed, we know that **Xw** – **t** gives the vector of the residuals (difference) between the model prediction and the real value. If we compute || **Xw** – **t** || then by the definition of the norm function we know that || **Xw** – **t** || = . Thus, || **Xw** – **t** || is the square root of the sum of the residuals squared. Since we just want to know the sum of the residuals squared thus we compute (|| **Xw** – **t** ||)2. Therefore, it follows that we can express the average loss function also in the following way:

In order to abbreviate our definition we could call the residual vector **Xw** – **t** => **r.**

Therefore we obtain:

I need to understand how it has been derived but in the following line I will outline for what vector **w** the average loss function in terms of vectors and matrices is minimised. Remember that the average loss function in terms of matrices and vectors is the same as the average loss function not using the matrices and vectors.

In order to compute the vector **w** for witch the average loss function is minimised and so we obtain the function that best fits the data, all we have to do is perform the following computations:

where denotes the vector **w** that minimises the average loss function

Solving a linear regression problem using equation 1.16 is called solving a liner regression problem in **CLOSED-FORM.**

In mathematics, a closed-form expression is a mathematical expression expressed using a finite number of standard operations. As we can see the right-hand size of equation 1.16 is a closed-form expression and since we just resolve (reduce) this expression for solving a linear regression problem then it is said that we solve a liner regression problem in closed-form.

However, solving a liner regression problem in closed-form is not the only way to resolve a regression problem. We must bear in mind that matrix operations are very time consuming and so costly operation. Thus, solving a linear regression problem where the number of data points and input variables is huge will be a very costly operation if solved in closed-form.

Additionally, the matrix (**XTX**) may be singular and so it does not have any inverse matrix which means we cannot solve the right-hand size expression of 1.16.

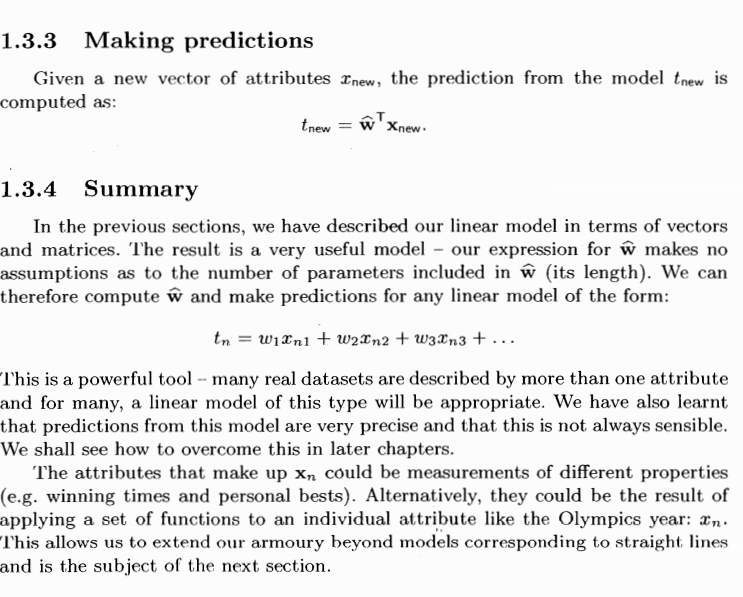
A second way we can solve linear regression problems (but also regression problems in general that are not linear) is using **GRADIENT-DESCENT.**

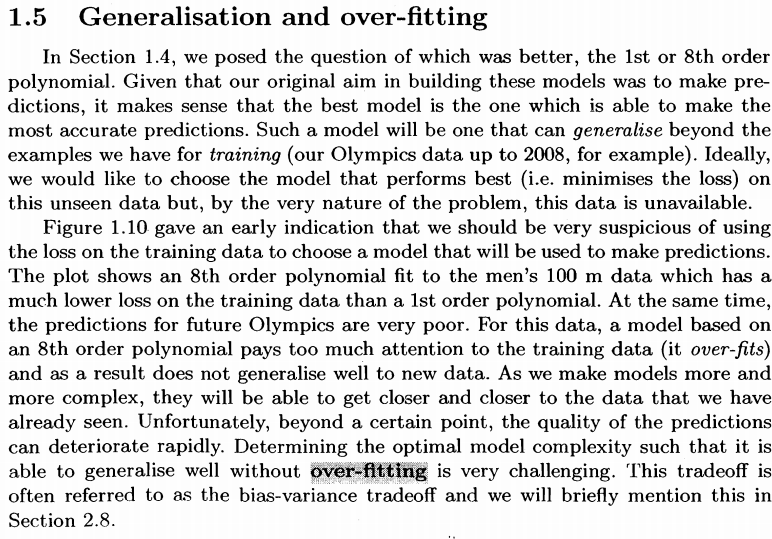
We already know how GRADIENT-DESCENT works, thus all we have to find in order to minimise the average loss function L is its gradient. We start from any point in L (a point is an assignment of the weight parameters) and then iteratively we perform gradient descent by following the steepest descending direction. When we use GRADIENT-DESCENT in the field of Machine Learning, the value of ϒ (step size) is called **LEARNING-RATE.**

Therefore, in linear regression with GRADIENT-DESCENT it is evident that the linear model is learning as it is monotonically updating its weight parameters so that to obtain better results. Given that we use GRADIENT-DESCENT with Linear-Search or we use a very low learning rate then the algorithm will monotonically improve its capabilities as L() > L() > … > L() where is the first choice for the weight parameters and is the last one.

In the case we apply GRADIENT-DESCENT with LINEAR SEARCH in order to minimise the average loss function L, since L is a convex, then we find a point whose gradient is the **0** vector then that point is the global minimum of the function L. By using LINEAR-SEARCH with exact solution, that is we find the global minimum of the function L (), that global minimum is guaranteed to be the global minimum of the function L as well. Therefore, we transform a minimisation problem of n parameters into a minimisation problem of just one parameter ϒ. All this holds because L is convex and actually a kind of parabola in high dimensional space

## How to make predictions in the vectorial defined linear model



i

In other words, given two or more models of regression which one should we use?

It is important to stress out that the aim of regression is to find the function that best describes the relationship between the set of independent variables and the dependent variable so that to make **FUTURE PREDICTIONS.** Thus, the model that we should select when regressing a supervised-learning continuous space problem is the model that **GENERALISES-WELL** with previously unseen data.

In other words, we should select the model which has the lowest average loss function value in previously unseen data.

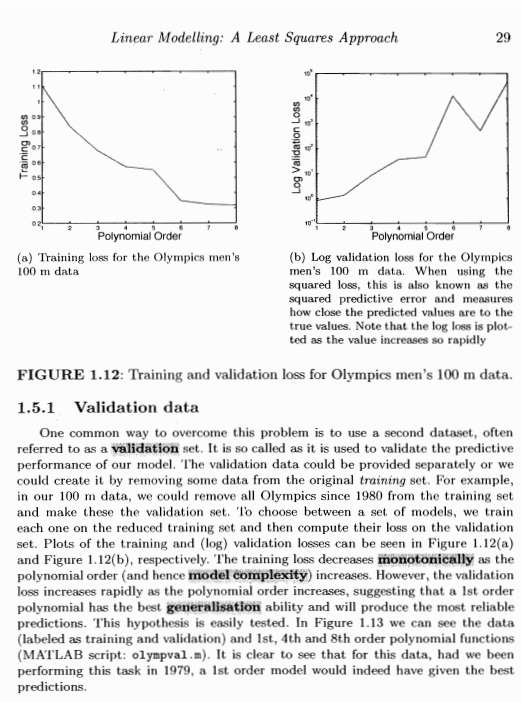
A regression model may have a low average loss function value in the training data but not generalise well with previously unseen data. Such a phenomenon is called OVERFITTING. This occurs when the model has paid too much attention to the training data and has overfitted it.

Determining which model performs well on the training data without overfitting it and so generalise well with previously unseen data is a challenging task.

It is important to notice from what the book says that the more complex a model is the better it will fit the training data but at a certain point selecting more complex models will make the model overfit the data and so generalise poorly with previously unseen data.

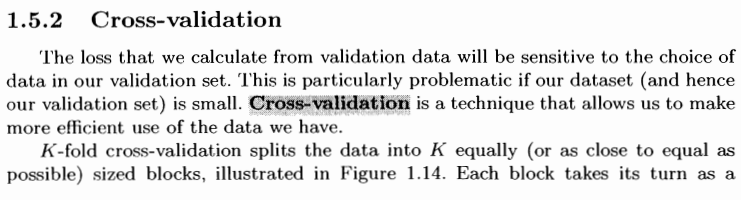
In the case of regression problems, complex models are models that try to fit high order polynomial functions to the data points.

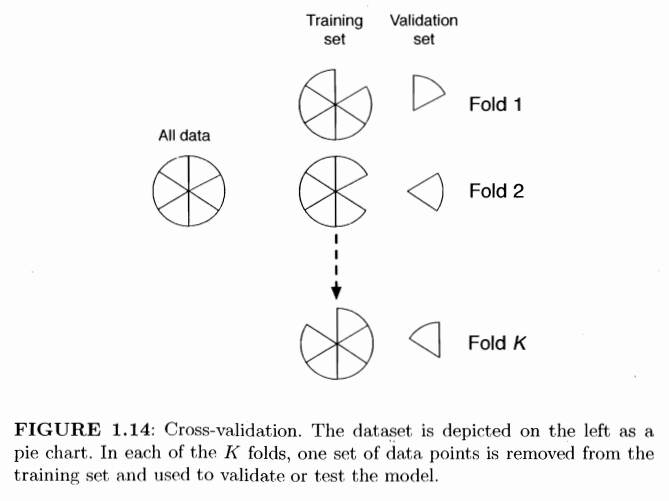
One way to test the prediction capabilities of our model is to use a **VALIDATION SET.**

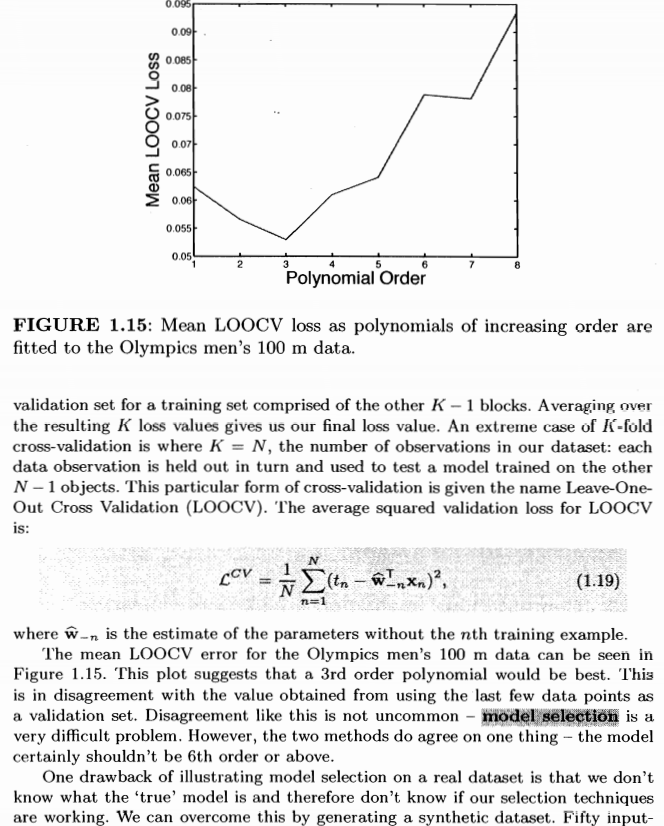


Thus by using a validation (test) set and a training set, we train several different models on the same training set and then evaluate their prediction capabilities on the validation set. We will choose the model which has the highest prediction capabilities and so the model with the lowest average loss function value on the validation set.

An alternative better solution to a single validation set is K-FOLD CROSS VALIDATION. The prediction capabilities of the different models are highly influenced by the validation set and so we could obtain wrong results because a particular model (not the best one) may perform the best on that particular validation set. In order to decrease the likelihood that this happens then we need to introduce many validation sets where to test the different models and compute the average loss of each model on all validation sets. Additionally, at each round of K-FOLD CROSS VALIDATION, the model will be trained with different training data.



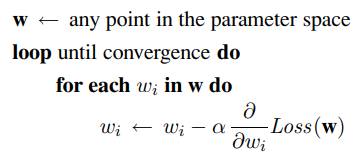




Therefore, an alternative way of performing K-cross validation on a particular model is to perform many rounds of K-cross validation by selecting different k partitions by shuffling the data set. Therefore, the average loss of the model will be equal to the average loss of the K-cross validation rounds which will return each an average loss as well.

Now, I am going to revisit linear regression with gradient descent once again.

Let **w** be a vector belonging to the weight space. In other words **w** is the vector of the weights of the linear function f (**x** ; **w**). Then, we can apply gradient descent algorithm in the following way:





where Loss(**w**) is just the average squared loss function that we have previously called L. In this case α is the learning rate ϒ.

The algorithm above is equivalent to the definition of gradient descent which is the following:

**wn+1**= **wn -** ϒ▽L(**w**)

as instead of simultaneously update all the weights inside the vector **w** we update each of them sequentially by using its corresponding partial derivative.

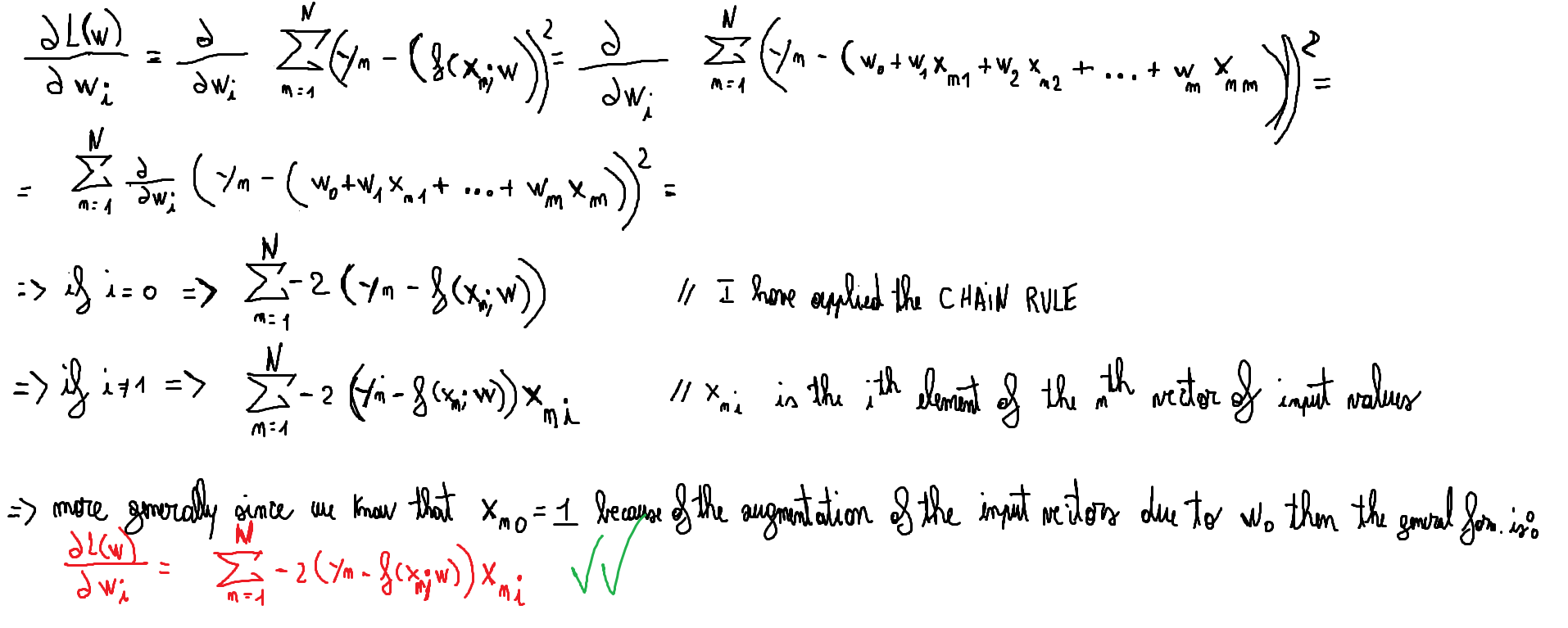
The algorithm says loop until convergence which will be reached when the weights are not updated anymore as we have reached the local (global) minimum. When we reach the local (global) minimum the weights won’t be updated anymore because each partial derivative of the global minimum is equal to 0.

Given that we have a very nice algorithm for gradient descent above all we have to do is to find the partial derivative with respect to each weight of the average loss function and we are done.

Therefore, now we are going to see how to find the partial derivatives in the most general case where there are multiple independent variables.



The above formula is the definition of the partial derivative for each weight.



Thus, we can modify the above algorithm by replacing the definition of the partial derivative just found:



Notice that when replacing the in the forth line the definition of the partial derivative. Notice that I have omitted to insert the factor (-2) and this is pretty ok as anyway we have to multiply everything for the learning rate α and so we can remove the factor -2 and incorporate it with the learning rate α.

Thus, now we know how to implement gradient descent for solving linear regression problems.

The stopping criteria of gradient descent in linear regression problems is when the vector **w** after an iteration has not been updated. In that case, it means that we have found the local minimum.

The algorithm that we have shown above is called **BATCH GRADIENT DESCENT**. The number of iterations that it runs are called EPOCHS. The smaller the factor α is the larger the number of epochs will be needed to converge.

The **BATCH GRADIENT DESCENT** does not have a great performance efficiency as to update at each epoch each weight it must cycle through all data points in order to compute the value of the partial derivative.

There is another version of gradient descent algorithm which is called **STOCHASTIC GRADIENT DESCENT** which works in a different way as it updates the weights of the algorithm by considering a single data point at a time. It will run for all data points in the data set.

The algorithm works in the following way:

**w** = select a random weight vector to start

loop until convergence

loop for each data point n in the data set

for reach wi in **w**

wi = wi + α (yn – f(**x**n ; **w**)) xni

Therefore, in stochastic gradient descent instead of updating the weights with respect to all data points, we update the weights looping for each data point and updating according each weight by only considering the current data point as if there was only this data point in the data set.

Stochastic gradient descent is often faster than batch gradient descent. With a fixed learning rate α, however, it does not guarantee convergence; it can oscillate around the minimum without settling down. In some cases, as we see later, a schedule of decreasing learning rates (as in simulated annealing) does guarantee convergence.