08_Optimization_Calculus

14 May 2025 09:30

Background:

The sequence:

1. We started by classifying big fish & small fish in a dataset which had two features of these fishes & found that if we plot their "features" on a graph then what we need is to find a straight line that separates these dots.
2. Then we found many such lines & concluded that the line that has highest total distances from these points can act as best separator. So, we created a "Gain function" that represents total distances of points.
3. We then plotted this "Gain function" (or "Loss function" by changing sign of the gain function) and found that we are interested in finding the point on this graph which is either maxima(in case of Gain function) or minima(in case of Loss function) to get the best boundary (straight line)
4. Then we came to know that if we draw tangents at different points on that function curve then the differentiation of the function gives slopes of those tangents at those points. And we are looking for the points (minima or maxima) at which the slope of the tangent is zero (hence the differentiation is 0). For this, we studied differentiation.
5. But now what we observe is that the Gain or Loss function is **not a function of only one variable** (unknown) but it is having two variables (w & w_0). So now we want to learn how to differentiate a function that has multiple variable.
6. So, let's learn how to differentiate a function with more than one variables! (Also known as partial differentiation)

Let $f(x,y) = 2x^3 - 4y^2$

$\frac{d}{dx} f = 6x^2 - 0$ $\qquad$ $\frac{\partial}{\partial y} f(x,y) = 0 - 8y$

$\frac{\partial}{\partial x} f(x,y) = 6x^2$ $\qquad$ $\frac{\partial}{\partial y} f(x,y) = -8y$

Partial Differentiation

"Complete" Differentiation of $f(x,y) = f'(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix}$

↳ Gradient

As we know, this differentiation represents tangent on the function and tangents are also known as "**Gradient**". Hence we want to find the **Gradient of our Loss Function**.

$$L(\vec{\omega}, \omega_0) = -\sum_{i=1}^{n} \frac{\vec{\omega}^T \cdot x_i + \omega_0}{\|\vec{\omega}\|} \cdot y_i$$

where $\vec{w} = [w_1, w_2, w_3, \ldots, w_d]$

To get the total differentiation of L, we need to partially differentiate it with respect to each component of w and also with w_0.

$\frac{\partial}{\partial w_0} L$ = Differentiation of L w.r.t. w_0

$\frac{\partial}{\partial w_1} L$ = " " " w_1

$\frac{\partial}{\partial w_2} L$ = " " " w_2

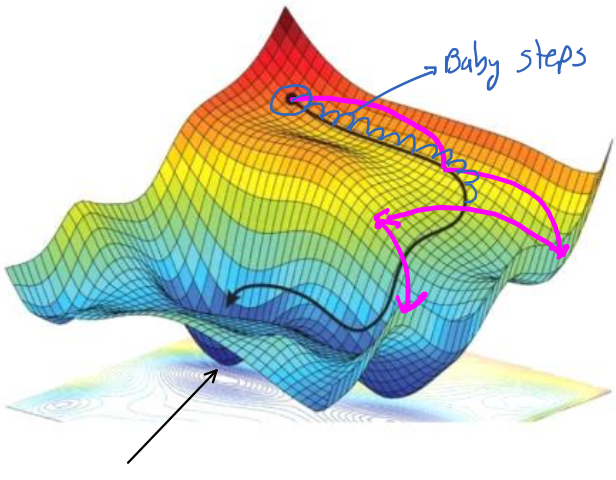
$\vdots$

$\frac{\partial}{\partial w_d} L$ = " " " w_d

∴ The gradient say ∇L, is given by:

$$\nabla L = \begin{bmatrix} \frac{\partial}{\partial w_0} L \\ \frac{\partial}{\partial w_1} L \\ \vdots \\ \frac{\partial}{\partial w_d} L \end{bmatrix}$$

Suppose we consider the following function:



Imagine, we have "guessed" some random values of w & w0. Now we need to make the next guess of w & w_0 based on some logical & mathematical process that takes us closer to the minima. What can be this "process" (formula)?

$^{t+1}w = {}^{t}w -$ size of the "baby steps" $\cdot \nabla L$

The new guess are derived from the current value of w & w0 that's why the size of the step that takes us to the next guess determines how fast or slow we will converge (reach to the correct answer - minima) and that is why this "size of baby steps" is also known as **Learning Rate**

of our algorithm and denoted by **Eta** (η). Replacing this symbol converts our formula into:

$$^{t+1}w = {}^{t}w - \eta \cdot \nabla L$$

What should be the Learning Rate? What happens if it is very high or very low? To avoid complicated 3D graphs, we will understand this concept in a 2D graph. Let our loss function be:

$L = f(w) = w^2 - 30$

Hence the gradient ∇L will become:

$$\frac{\partial L}{\partial w} = 2w \qquad \nabla L = \begin{bmatrix} 2w \end{bmatrix}$$

$$\therefore {}^{t+1}w = {}^{t}w - \eta \cdot \nabla L \Rightarrow \boxed{{}^{t+1}w = {}^{t}w - 2\eta\, {}^{t}w}$$

Let's take w = 3 as our initial guess. What if our Learning Rate is too big? Let's take η = 1.

$$\therefore {}^{t+1}w = {}^{t}w - 2 \cdot {}^{t}w$$

Let's take ${}^{t}w = 3$ as our initial guess:

$$\therefore {}^{t+1}w = 3 - 2(3) \Rightarrow \boxed{{}^{t+1}w = 3 - 6 = -3}$$

∴ with new ${}^{t}w = -3$, we will try to compute new ${}^{t+1}w$ using the same formula.

$${}^{t+1}w = {}^{t}w - 2 \cdot {}^{t}w \Rightarrow {}^{t+1}w = (-3) - 2 \cdot (-3)$$

$$= -3 + 6$$

$$\boxed{{}^{t+1}w = 3}$$

Now let's see what happens when we keep very small learning rate. Let's take η = 0.001 hence the formula for the next guess will become:

the formula for the next guess will become:

$$^{t+1}w = {}^{t}w - 2*(0.001)*{}^{t}w \Rightarrow {}^{t+1}w = {}^{t}w - 0.002*{}^{t}w$$

Let $^{t}w = 3 \Rightarrow {}^{t+1}w = 3 - 0.002*3 = 2.994$

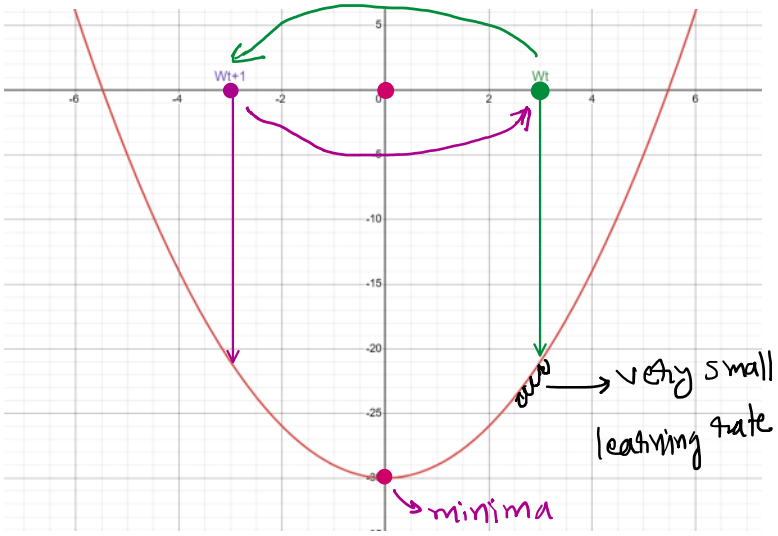
$\therefore {}^{t}w = 2.994 \Rightarrow {}^{t+1}w = 2.994 - 0.002*2.994 = 2.988$

Continuing this process for one more iteration,

$${}^{t}w = 2.988 \Rightarrow {}^{t+1}w = 2.988 - 0.002*2.988 = 2.982$$

As we see, we are moving towards minima very slowly and it will take 100s of iterations to reach to the actual minima.

Let's try to visualize this graphically:



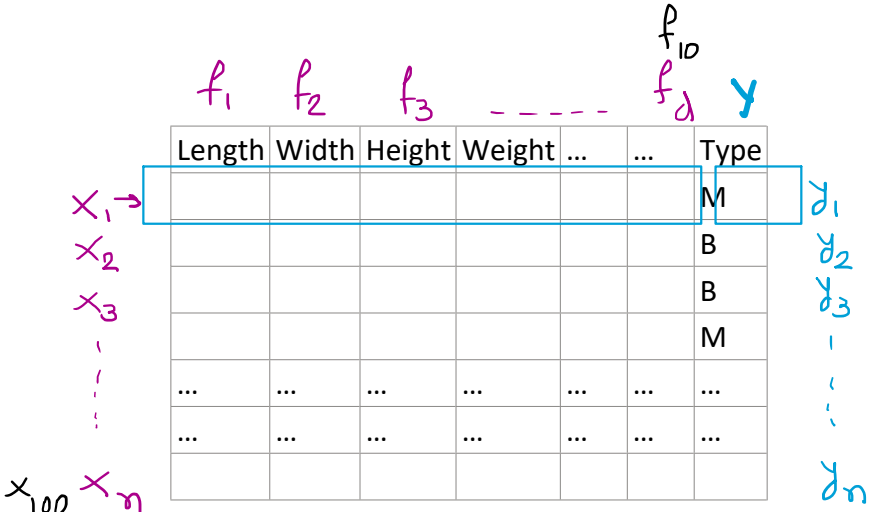
It is clear that η is a deterministic parameter for the performance of our algorithm and it is in our hands to select the right value of η and we must fine-tune to a good value of η that balances both accuracy & speed of the algorithm.

Any parameter that determines the performance of a machine learning algorithm and which needs to be fine-tuned by us is known as a **Hyper-parameter** and **Hyper-parameter Tunning** is a major part of any machine learning algorithm.

Therefore, η is such a hyper-parameter that we need to fine-tune.

Naming conventions that we will follow:

Example dataset: Dataset of malignant & benign tumors



$\therefore$ The classifier/separator:

$$w_1 f_1 + w_2 f_2 + \dots w_d f_d + w_0 = 0$$

$$w_1 f_1 + w_2 f_2 + \dots w_{10} f_{10} + w_0 = 0$$

And the loss function:

$$1 = -1 < \vec{w}^T \cdot x_i + w_0 \cdot y_i$$

...	...	...	...	...	...	...

x_{100} x_n y_n y_{100}

$$L = -\frac{1}{n}\sum \frac{\vec{w}^T \cdot x_i + w_0}{\|\vec{w}\|} \cdot y_i$$

$$L = -\frac{1}{100}\left[\frac{\vec{w}^T \cdot x_1 + w_0}{\|\vec{w}\|} \cdot y_1 + \frac{\vec{w}^T \cdot x_2 + w_0}{\|\vec{w}\|} \cdot y_2 + \cdots + \frac{\vec{w}^T x_{100} + w_0}{\|\vec{w}\|} \cdot y_{100}\right]$$

Now we will spend some time also to understand how to deal with Regression Problems (the problem that we kept in mind from the beginning of ML was fish-sorting problem which is a Classification Problem). The reason behind this is that it is easy to create solution of a regression problem and by making some changes in it we can also solve the classification problems.

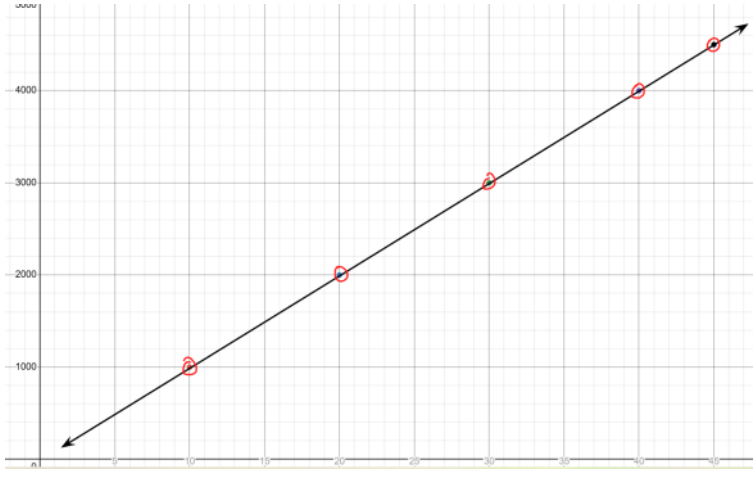
Regression Problems:
Unlike to classification problems where we have fixed number of outcomes (categories) to classify the incoming item, in regression problems the number of outcomes is infinite.

Examples of classification problems:
Fish sorting problem (categories: big fish/small fish)
1. Spam filter that determines whether the incoming mail is spam or not spam
2. Fraudulent transaction detection (categories: fraud/genuine)

Examples of regression problems:
1. House price/car price prediction
2. Weather forecasting
3. Cyclone intensity/route prediction
4. Stock price prediction

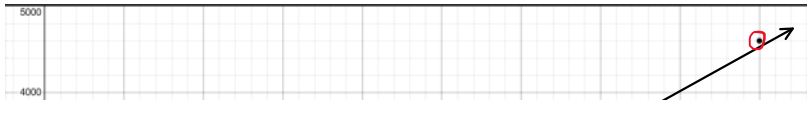
Linear Regression:
Example - Suppose we found that the electricity bill of our home is directly proportional to the average temperature of the city

Avg. Temp	Ele. Bill
10	1000
20	2000
30	3000
40	4000
45	4500



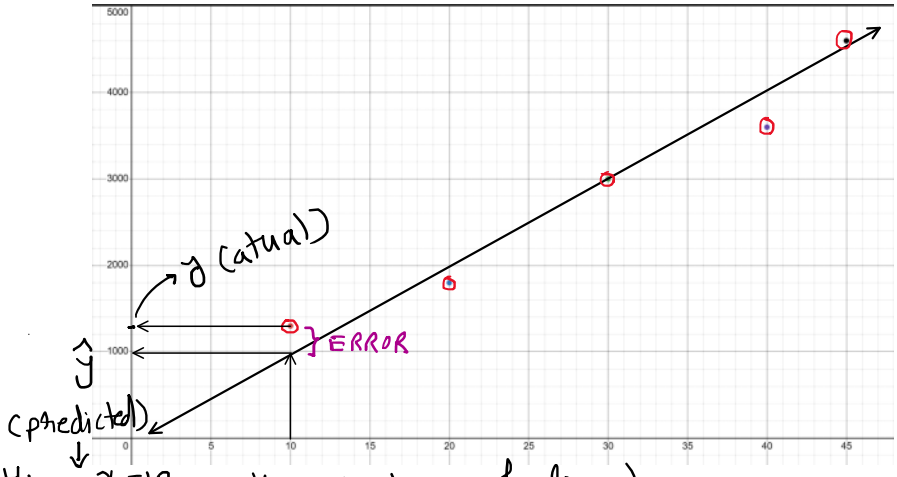
Based on the above table, what do you think what will be our electricity bill when the temperature is 35?
Ans: 3500

A slightly more realistic table can be as below:

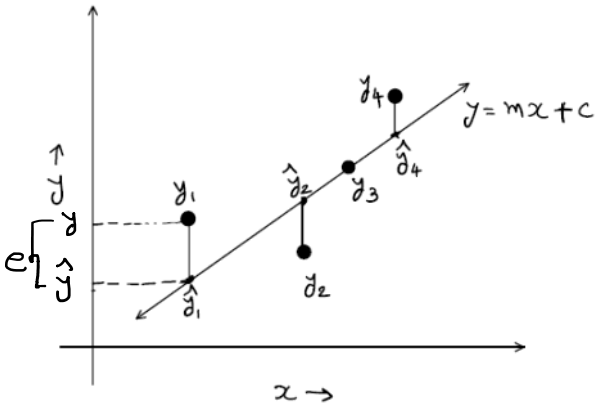


A slightly more realistic table can be as below:

Avg. Temp	Ele. Bill
10	1300
20	1800
30	3000
40	3600
45	4600



Now suppose the dataset is as below:



Here the feature (e.g., temperature) is on X-axis & the target (e.g., electricity bill) is on Y-axis and suppose the best possible line that we get is:
y = mx + c

Therefore, we can predict our target variable "yi" by putting value of "x_i" in this equation which may or may not be exactly same as the actual "y_i". So let's call our prediction as $\hat{y}_i$ & the actual value as y_i

∴ The error in $y_1 = y_1 - \hat{y}_1 > 0$. Error in $y_2 = y_2 - \hat{y}_2 < 0$

This way, the +ve & -ve errors will cancel out each other and even if our is far from every datapoints but still the total error become 0. So we need such an equation for error that will always return +ve error. There are two ways to do this:

① Taking absolute of error (modulus) – M.A.E. (Mean Absolute Error)

② " square of error - M.S.E. (Mean Squared Error)

∴ First we will square the errors & then to calculate total error we will take sum of these squares.

∴ Total errors = $(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \dots + (y_n - \hat{y}_n)^2$

$$= \sum (y_i - \hat{y}_i)^2$$

$$= \sum_{i=1}^{n} (y_i - (m \cdot x_i + c))^2$$

If we want to find average error

$$L = \frac{1}{n} \sum (y_i - (mx_i + c))^2$$ ← This is going to be our Loss function

Now, we differentiate the Loss Function to find the gradient ∇L. In this equation, we want to find 'm' & 'c'. In other words, they are our unknowns. ∴ We will differentiate the Loss function partially w.r.t. them one by one.

$$\frac{\partial L}{\partial m} = \frac{1}{n} \frac{d}{dm} \sum (y_i - (mx_i + c))^2$$

$$= \frac{1}{n} \sum 2(y_i - (mx_i + c)) \cdot \frac{d}{dm} (y_i - (mx_i + c))$$

$$= \frac{2}{n} \sum (y_i - (mx_i + c)) (0 - x_i - 0)$$

$$\boxed{\frac{\partial L}{\partial m} = \frac{-2}{n} \sum (y_i - (mx_i + c)) \cdot x_i}$$

$$\frac{\partial L}{\partial c} = \frac{1}{n} \frac{d}{dc} \sum (y_i - (mx_i + c))^2$$

$$= \frac{1}{n} \sum 2(y_i - (mx_i + c)) \cdot \frac{d}{dc} (y_i - (mx_i + c))$$

$$= \frac{2}{n} \sum (y_i - (mx_i + c)) \cdot (0 - 0 - 1)$$

$$\frac{\partial L}{\partial c} = \frac{-2}{n} \sum (y_i - (mx_i + c))$$

These two equations are the most important formulae for Gradient Descent Algorithm.