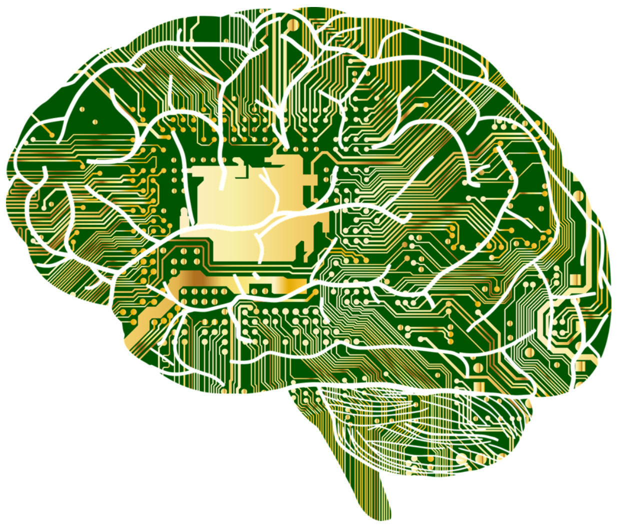
Endterm Report: Machine Learning

Summer of Science 2022

Mentor: Garweet Sresth

Ashwin Abraham

20th July, 2022

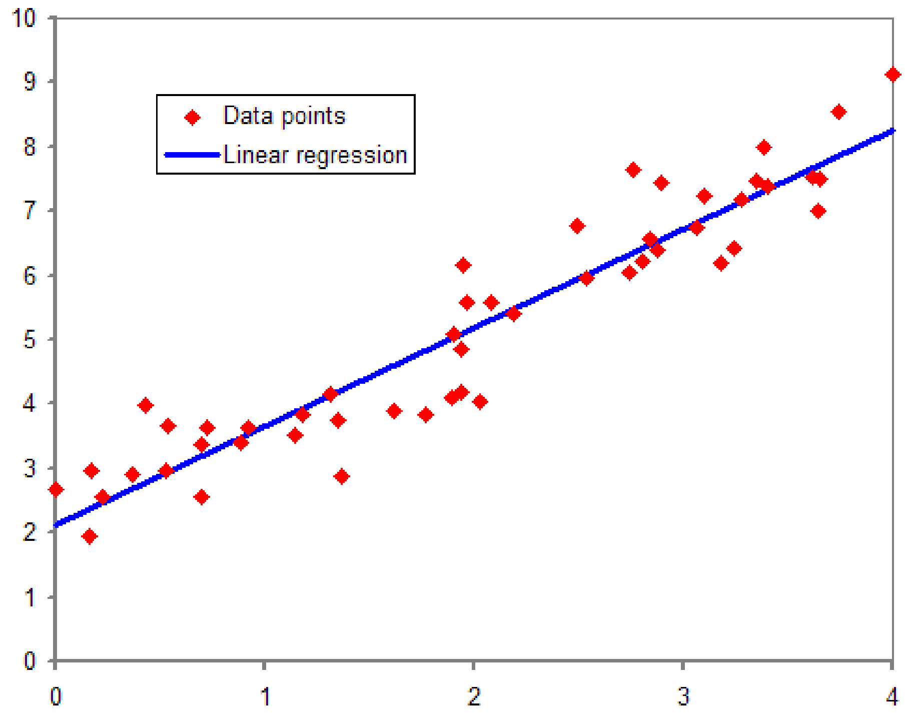


Contents

Chapter 1

Linear and Logistic Regression

Regression is the statistical process to find the relationship between a dependent variable and an independent variable that minimizes some cost function, given a set of datapoints.



The above graph shows an example of Linear Regression with one dependent and one independent variable, where a set of datapoints is given, and we try to find the line that best represents the relationship between the dependent variable and the independent variable.

The notion of what we mean by "best representation" is formalized by the cost function. This is a function of the dataset given to us and the approximate relationship we use to model the relationship between the dependent and independent variables. The best representation of this relation is the one that has the least cost (minimizes the value of the cost function).

1.1 Linear Regression

[1]

In linear regression, we impose the constraint that approximate relationship between the dependent and independent variables should be a linear function. The cost function usually chosen for regression problems is the sum of squares function (leading the method itself to be called the method of least squares).

If there are N datapoints of the form (x_i, y_i) and the relation function is f, then the cost of f is then defined as:

$$C(f) = \sum_{i=1}^{n} ||y_i - f(x_i)||^2 \tag{1.1}$$

Here, if there are more than 1 dependent/independent variable, x and y become vectors.

Assuming f is linear and there are only 1 dependent and 1 independent variable, we get $f(x) = Ax + B$. Now we must find constants A and B such that $C(f) = C(A, B)$ is minimized. Now,

$$C(A, B) = \sum_{i=1}^{n} (y_i - Ax_i - B)^2 \tag{1.2}$$

Imposing the conditions that $\frac{\partial C}{\partial A} = \frac{\partial C}{\partial B} = 0$, we get

$$\sum_{i=1}^{n} (Ax_i + B - y_i)x_i = 0 \tag{1.3}$$

[1]Primary Source: Wikipedia

and

$$\sum_{i=1}^{n}(Ax_i + B - y_i) = 0 \tag{1.4}$$

Simplifying, we get a system of Linear equations in A and B:

$$A(\sum_{i=1}^{n} x_i^2) + B(\sum_{i=1}^{n} x_i) = \sum_{i=1}^{n} x_i y_i \tag{1.5}$$

$$A(\sum_{i=1}^{n} x_i) + Bn = \sum_{i=1}^{n} y_i \tag{1.6}$$

Note, that since in the RMS-AM inequality $(n(\sum_{i=1}^{n} x_i^2) \geq (\sum_{i=1}^{n} x_i)^2)$, equality occurs only when all x_i are equal (which can never happen with a proper data set), this equation will always have a unique solution, given by:

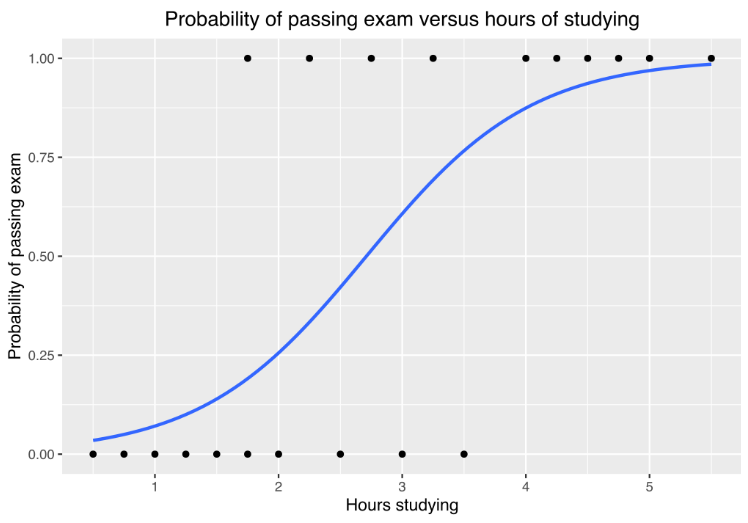
$$A = \frac{n(\sum x_i y_i) - (\sum x_i)(\sum y_i)}{n(\sum x_i^2) - (\sum x_i)^2} \tag{1.7}$$

$$B = \frac{(\sum y_i)(\sum x_i^2) - (\sum x_i y_i)(\sum x_i)}{n(\sum x_i^2) - (\sum x_i)^2} \tag{1.8}$$

1.2 Logistic Regression

[2] In a logistic regression on the other hand, the dependent variable is constrained to be only either 0 or 1. In this case, the best approximation to the relation between the dependent and independent variables will also be the Probability Function of the dependent variable (this is a function of the independent variables that gives the probability that the dependent variable will be 1 for a particular choice of independent variables).

[2]Primary Source: Wikipedia



The above graph shows an example of a Probability function obtained by Logistic Regression.

The cost function for Logistic Regression is given by:

$$C(p) = -\sum_{i=1}^{n} [y_i \ln(p(x_i)) + (1 - y_i) \ln(1 - p(x_i))] \tag{1.9}$$

and we model the relation between the dependent and independent variables by a logistic function, given by:

$$p(x) = \frac{1}{1 + \exp(-(Ax + B))} \tag{1.10}$$

Now, expressing C in terms of A and B and then imposing the condition that $\frac{\partial C}{\partial A} = \frac{\partial C}{\partial B} = 0$, we get the values of A and B. Note that the equations involved may not always have exact solutions and may have to be solved numerically.

1.3 Use of Regression in Machine Learning

A system implementing Machine Learning techniques must be trained with a large amount of data and it must use the data it has been trained with in order to find ways of handling data that it has not come across before. This can be done effectively using regression techniques. The Machine can use regression analysis on the training data to get a relation between the data and the desired outcome. If the outcome is a binary decision, Logistic regression may be used, and if it is a continuous outcome, Linear regression may be used. The relation obtained is applied on the new data to get the outcome required.

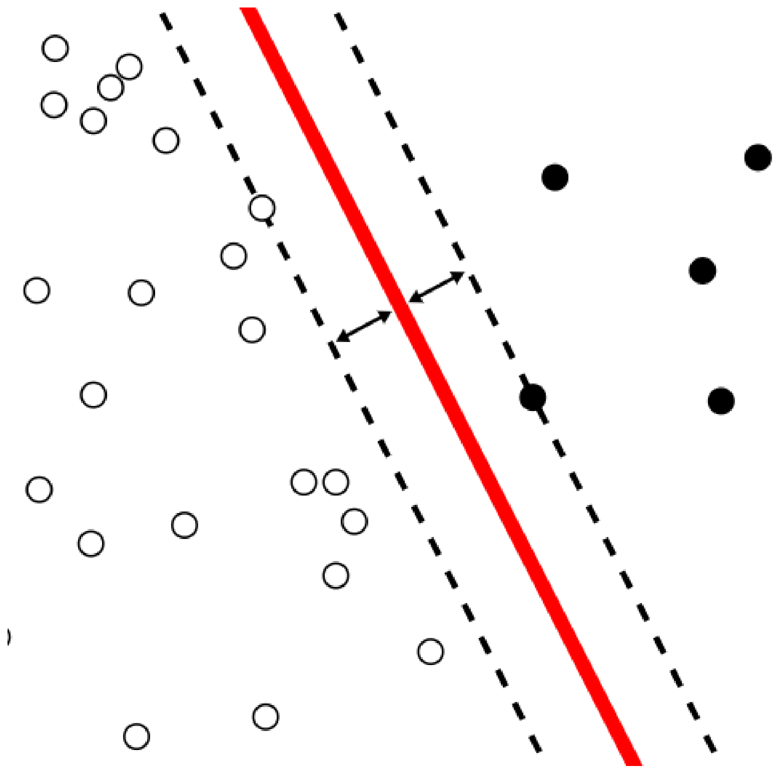
Chapter 2

Support Vector Machines

[1] In Machine Learning, we often need to classify data into multiple groups. If the data can be represented as an n-dimensional vector, then the dataset becomes a set of points in $\mathbb{R}^n$. The easiest way to classify data into different groups is to find a n-dimensional hyperplane dividing the $\mathbb{R}^n$ into two parts, one containing one set of points and another containing the remaining points.

To find the hyperplane, we use a set of points that have already been preclassified into two groups and try to find a hyperplane that separates the two groups into two halves of $\mathbb{R}^n$. If such a hyperplane exists, the dataset (along with the grouping) is said to be Linearly Separable. If a hyperplane exists, then (assuming that the number of points is finite), then an infinite number of such hyperplanes exist. We therefore, choose the hyperplane with the maximum margin, where the margin is defined as the max(minimum distance to a point on one side, minimum distance to a point on the other), as illustrated in the following figure:

[1]Primary Source: Wikipedia



For the training set of points $\boldsymbol{x_1}$, $\boldsymbol{x_2}$ $\cdots$, $\boldsymbol{x_n}$, where $\boldsymbol{x_i} \in \mathbb{R}^m$, consider the function y defined as:

$$y(\boldsymbol{x}) = \begin{cases} 1 & , \boldsymbol{x} \in \text{Group } 1 \\ -1 & , \boldsymbol{x} \in \text{Group } 2 \end{cases} \tag{2.1}$$

We can write the equation of the separating plane with the maximum margin as $\boldsymbol{w}^T\boldsymbol{x} = b$ where $\boldsymbol{w}$ is an arbitrary non-null vector in $\mathbb{R}^m$ and b is an arbitrary real number.

2.1 Linearly Separable Data

If the dataset is linearly separable, then there exist two parallel margin planes passing through the closest points to the separating plane. The seperating plane will also be parallel to this plane and will be exactly between the two planes. Therefore, we can have appropriate $\boldsymbol{a}$ and b such that the two margin planes are given by $\boldsymbol{a}^T\boldsymbol{x} - b = 1$ for the separating plane passing through the closest point in Group 1 and $\boldsymbol{a}^T\boldsymbol{x} - b = -1$ for the separating plane passing through the closest point in Group 2.

The distance between the margins here is given by $d = \frac{2}{||\boldsymbol{a}||}$, which is the quantity to be maximized (i.e. $||\boldsymbol{a}||$ must be minimized). Imposing the condition that all the points in group 1 lie on the opposite side of $\boldsymbol{a}^T\boldsymbol{x} - b = 1$ as $\boldsymbol{a}^T\boldsymbol{x} - b$ and similiarly for the points in group 2 and $\boldsymbol{a}^T\boldsymbol{x} - b = -1$.

This condition may be expressed as:

$$y(\boldsymbol{x_i})(\boldsymbol{a}^T\boldsymbol{x_i} - b) \geq 1, \forall i \in \{1 \cdots n\} \tag{2.2}$$

If we know the two closest points to the separating plane (from opposite sides) (say $\boldsymbol{x_a}$ and $\boldsymbol{x_b}$), we can reduce these constrained minimization problem to minimizing $||\boldsymbol{a}||$ based on the constraints:

$$y(\boldsymbol{x})(\boldsymbol{a}^T\boldsymbol{x} - b) = 1, \boldsymbol{x} \in \{\boldsymbol{x_a}, \boldsymbol{x_b}\} \tag{2.3}$$

$\boldsymbol{x_a}$ and $\boldsymbol{x_b}$ are known as support vectors (hence the name Support Vector Machine). $\boldsymbol{a}$ and b obtained are clearly dependent only on the Support Vectors.

2.2 Non-Separable Data

For data that is not linearly separable, a separating plane doesn't exist that clearly divides space into regions containing only points of the same group. However, it is still possible for there to exist a plane that roughly divides space into regions containing points mostly of the same types. We can still use this plane to make predictions about points not in our data.

Instead of minimizing $||\boldsymbol{a}||$, we instead minimize the quantity:

$$k||\boldsymbol{a}||^2 + \frac{1}{n}\sum_{i=1}^{n}[H(\boldsymbol{a}^T\boldsymbol{x_i} - b, y(\boldsymbol{x_i}))] \tag{2.4}$$

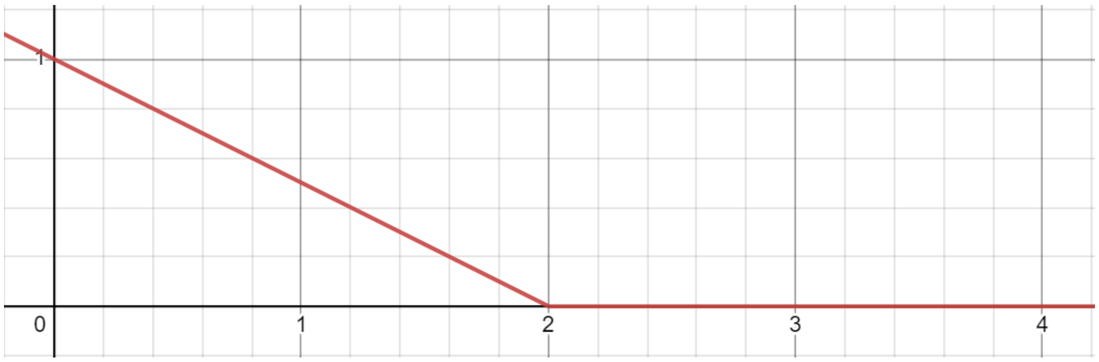
for parameters k and then choose the most appropriate k.

Here H represents the hinge function, defined as:

$$H(p, q) = max(0, 1 - pq) \tag{2.5}$$

Clearly, if $y(\boldsymbol{x_i})(\boldsymbol{a}^T\boldsymbol{x_i} - b) \geq 1$ then $H = 0$, i.e., if the points are on the correct side of the plane, there is no increase in the quantity to be minimized. If they are on the wrong side, then there is an increase, which is **proportional** to the degree in which the points are in the wrong side.

The following is a graph of the Hinge function for $q = \frac{1}{2}$:



Chapter 3

Neural Networks

[1] To define a Neural Network, we must first define a Neuron in the context of Machine Learning. A Neuron is defined as an object that takes in inputs (usually from other Neurons in the network) between 0 and 1 and produces an output, also between 0 and 1, according to the rule

$$y = f(\sum_{i=1}^{n} w_i x_i + b) \tag{3.1}$$

where x_i are the inputs to the Neuron, w_i are numbers corresponding to each input known as the weights of the inputs, and b is a number known as the bias of that input. The function f is known as the activation function of the neuron. This function must have it's range as $(0, 1)$. We usually represent the weights and inputs in vector form, as $\boldsymbol{w}$ and $\boldsymbol{x}$ where the components of $\boldsymbol{w}$ and $\boldsymbol{x}$ are w_i and x_i. The activation function used in most Neural Networks used for Machine Learning is the logistic function described earlier:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

Therefore, the output becomes:

$$y = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x} - b)} \tag{3.3}$$

Neurons with this activation function are known as Sigmoidal Neurons. Another common activation function, especially in older Neural Networks, is

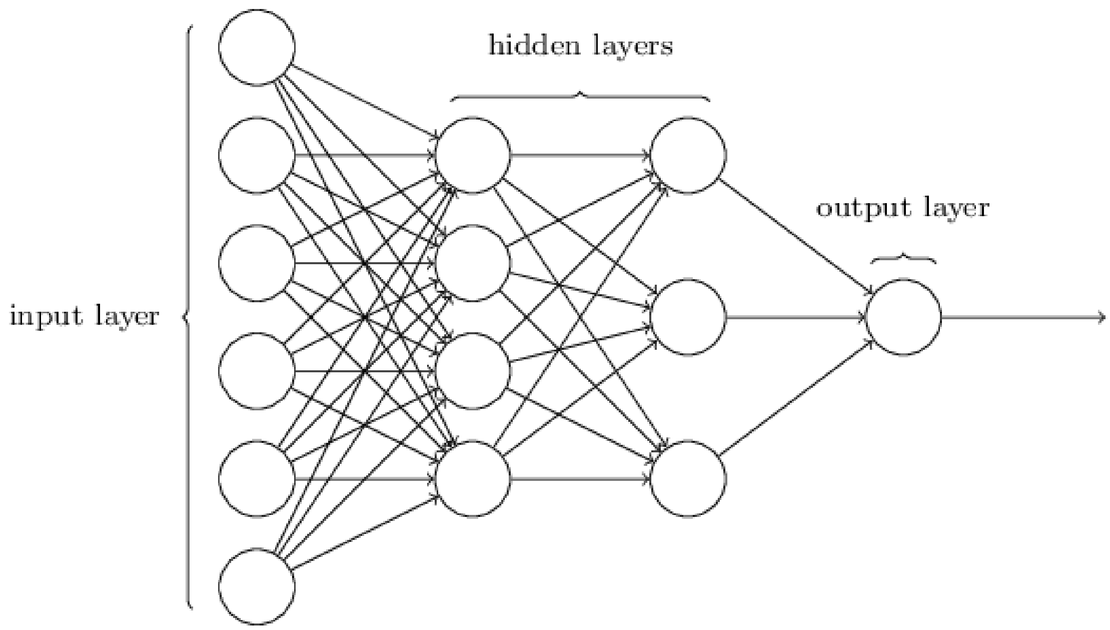
[1]Primary Source: Neural Networks and Deep Learning, Michael Nielsen

the step function:

$$H(x) = \begin{cases} 1 & , x > 0 \\ 0 & , x \leq 0 \end{cases} \tag{3.4}$$

Such Neurons are called Perceptrons and Perceptron Networks were historically the first Neural Networks invented.

A Neural Network is nothing but a network of such Neurons. We can further divide the Neurons into multiple layers, where each neuron in a layer takes input from all the neurons in the preceding layer. The first layer is called the input layer and consists of the input fed into the Neural Network, and the last layer is called the output layer. The other layers in between these are known as hidden layers.



A Neural Network can thus be viewed as a graph where each node represents a neuron and each edge is associated with a weight.

For example, if we consider a Neural Network tasked with learning how to classify handwritten digits, there would be N^2 input neurons, for an $N \times N$ resolution picture of a handwritten digit, and 10 output neurons, representing the probability that the digit is $0, 1, 2 \ldots 9$.

3.1 Gradient Descent

To use a Neural Network, we must first train it. To do this, we need a large, labelled data set. Considering the previous example, we would need a large database of images of handwritten digits, classified according to the digits that they represent. The MNIST Database of Handwritten Digits comes in handy here. We can train the neural network using this database.

To do this, we must develop a cost function, describing how much the prediction of our neural network deviates from the actual answer. The cost function usually used while building a Neural Network is the mean squared error.

$$C(\boldsymbol{w_i}, b_i) = \frac{1}{2n} \sum_{i=1}^{n} ||\boldsymbol{y}(\boldsymbol{x_i}) - \boldsymbol{a}(\boldsymbol{x_i})||^2 \tag{3.5}$$

Here, $\boldsymbol{y}(\boldsymbol{x})$ is the output, represented as a vector whose components are the activations of the neurons in the output layer, and $\boldsymbol{a}(\boldsymbol{x})$ is the output if the neural network were 100% accurate (i.e. if the input was an image of a handwritten 1, $\boldsymbol{a}$ would be $(0, 1, 0, 0, 0, 0, 0, 0, 0, 0)^T$).

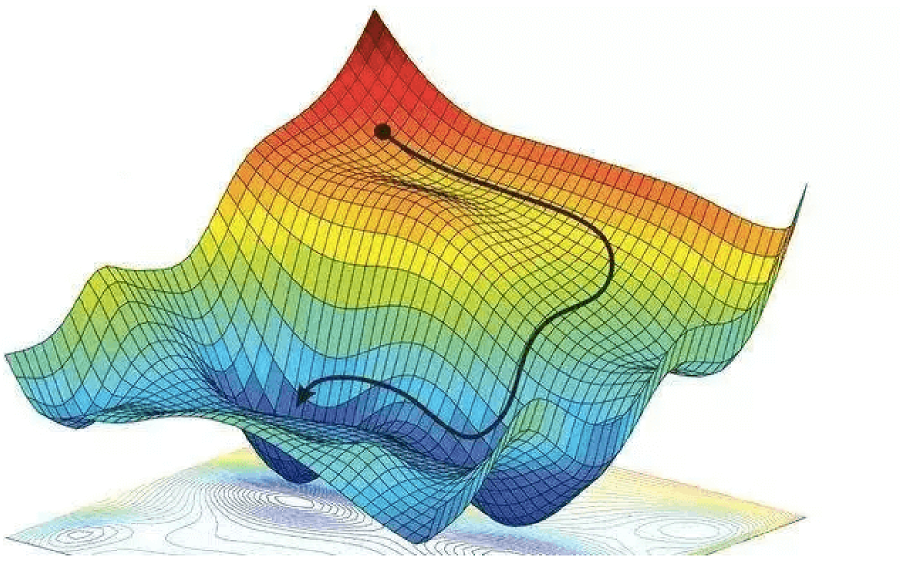
We therefore must choose $\boldsymbol{w_i}$ and b_i such that $C(\boldsymbol{w_i}, b_i)$ is minimized. To do this analytically would be nearly impossible and would be extremely computationally intensive. Therefore, we use a technique known as gradient descent.

Here, instead of calculating the gradient of C at all points and setting it to 0 to find a minimum, we instead take a random point (i.e. random values of $\boldsymbol{w_i}$ and b_i) and find the gradient at that point. Since the gradient points in the direction in which the function increases the most, we can move our random point in the opposite direction to decrease C the most. The amount by which we move must also be proportional to $||\nabla C||$ as when ∇C is small, it means we are close to the minimum.

If we represent the parameters ($\boldsymbol{w_i}$ and b_i) by the vector $\boldsymbol{r}$, the rule:

$$\boldsymbol{r} \leftarrow \boldsymbol{r} - \eta \nabla C(\boldsymbol{r}) \tag{3.6}$$

will eventually lead us to a minimum.



[2] The constant η is related to the learning speed of the Neural Network. A high value of η means that the minimum of C is reached more quickly, however, accuracy of the minimum obtained is less (as high values of η can sometimes lead to overshooting). However, even obtaining the gradient of C at a point can be difficult analytically. Hence it can be approximated by methods such as Stochastic Gradient Descent.

Note that,

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i \tag{3.7}$$

where

$$C_i = \frac{1}{2}||\boldsymbol{y}(\boldsymbol{x_i}) - \boldsymbol{a}(\boldsymbol{x_i})||^2 \tag{3.8}$$

Therefore,

$$\nabla C = \frac{1}{n}\sum_{i=1}^{n} \nabla C_i \tag{3.9}$$

To reduce the computational cost of calculating ∇C, we divide the inputs into smaller batches (say of size m each) and use the approximation:

$$\nabla C = \frac{1}{m}\sum_{j=1}^{m} \nabla C_{X_j} \tag{3.10}$$

[2]Image Credits: easyai.tech

and use the calculated ∇C in the gradient descent. After each step of Gradient Descent, we can use a different batch, so that all the batches are used.

To calculate ∇C_i we use techiques such as backpropagation.