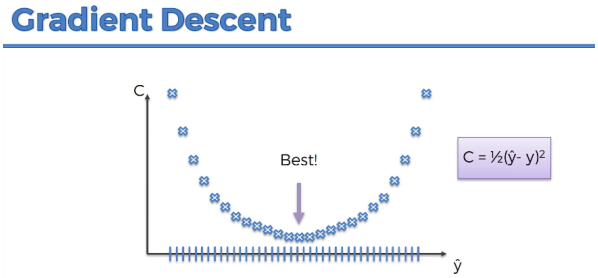
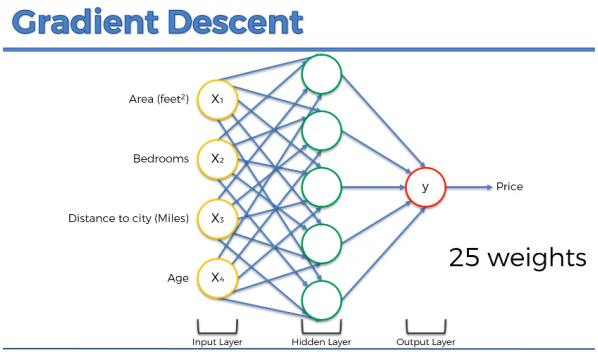
Gradient Descent

Brute Force:

It goes over each possible weight and get the *Cost* function, then selects the global minimum point where the *Cost* function is the lowest.

However, this kind of an approach is not possible in application. In theory, it may work but there is not enough computation power to support it.





The Curse of Dimensionality: In a simple NN, the right side graph, there are 25 weights (5 weights for each input layer to the hidden layer and 5 weights from the hidden layer to the output layer). If we have 10 combinations of weights, then the total number of combination for the NN is 10^{25}. However, if we have 100 combinations, then it goes from $10^{25} \rightarrow 100^{25} = 10^{50}$. Finally, with 1000 combinations, it can become impossible to solve, with the total number of combinations $1000^{25} = 10^{75}$, there is not enough computation power to optimize it.

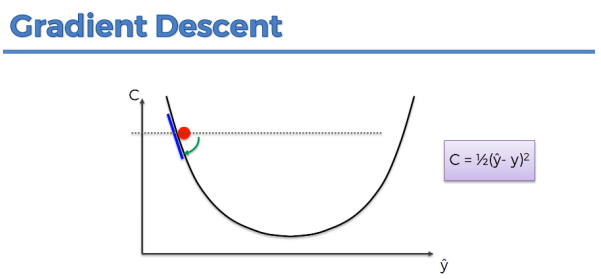
However, what we talk about is a simple NN with a single layer, if we add another layer, or increase the number of perceptrons or input variables, it will go another level by growing exponentially.

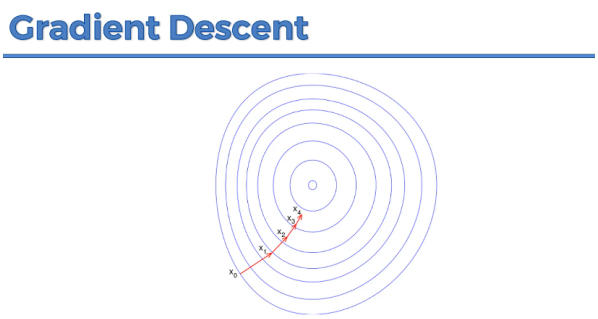
Gradient Descent:

It is called **Gradient Descent**, because we descent to the minimum of the *Cost* function.

The *Gradient Descent* requires a Convex *Cost* function which has a one global minimum.

In the *Gradient Descent* approach, we do not go over each possible combination with brute force, but we use the the slope of the *Cost* function to infer in which direction we have to go in order to reach the minimum. For example, in the graph on the left, we are on the upper left portion of the *Cost* function and we have to go downward, regarding that the *Cost* function has quadratic behavior, in order to reach the minimum point. The graph on the right visualize this behavior on a 2D plot.





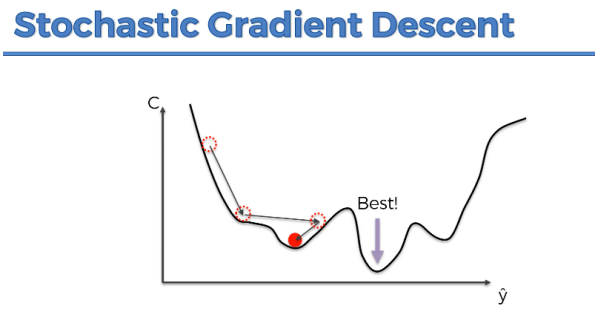
Stochastic Gradient Descent:

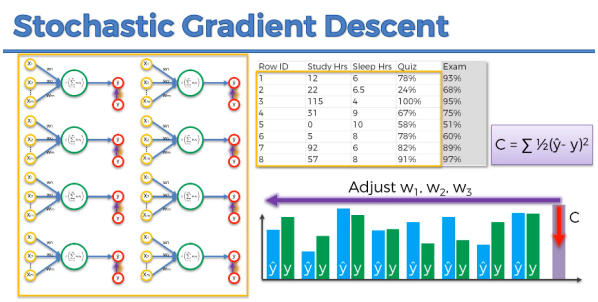
The main difference between *Gradient Descent* and *Stochastic Gradient Descent* is the behavior of the *Cost* function. While, the *Gradient Descent* requires a convex cost function in order to find the global minimum, otherwise it can stuck in a local minimum, the *Stochastic* Gradient Descent does not require the *Cost* function to be convex so that it can reach the global minimum. On the left graph, we see that the *Cost* function is not convex and the NN is not optimized, because it stuck in local minimum.

Another difference between them is how they handle the data. *Gradient Descent* takes a batch of data, data consisting of multiple rows, and update the weights for that batch in order to optimize the NN. However, *Stochastic Gradient Descent* takes the data one row at a time and update weights for each row. By taking one row at a time, *Stochastic Gradient Descent* face more fluctuations so that it is

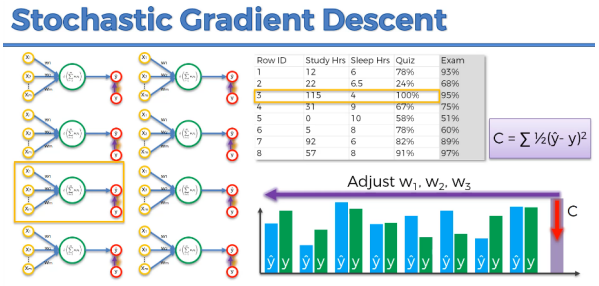
able to move from local minimum to the global minimum. Moreover, it is more faster than *Batch Gradient Descent*, even though it seems counterintuitive.

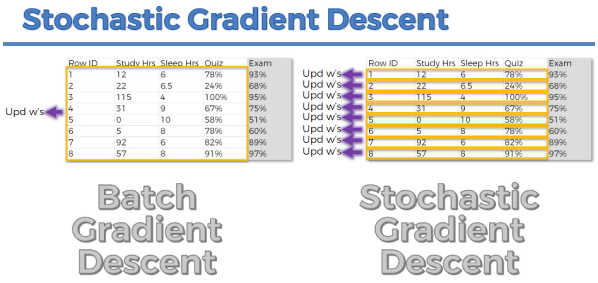
In the below, the left graph shows a representation of a non-convex *Cost* function. On the right, it shows how *Gradient Descent* works with b





In the below, the left graph shows how *Stochastic Gradient Descent* works with one row at a time. On the right, it is possible to see a comparison of *Gradient Descent* with *Stochastic Gradient Descent*.





Mini-Batch Gradient Descent:

The *Mini-Batch Gradient Descent* is the combination of *Gradient Descent* and *Stochastic Gradient Descent* where it takes not whole batch nor single row but a mini-batch may consists of a group of single rows such as 5-10 rows etc.

Resources:

https://iamtrask.github.io/2015/07/27/python-network-part2/

http://www.deeplearningbook.org/

http://neuralnetworksanddeeplearning.com/index.html