Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses. **Optimizers** help to get results faster.

What people usually mean by Optimization is to find a set of parameters that minimize or maximize a function. In the context of neural functions, this usually means minimizing a cost function by iteratively tuning the trainable parameters.
Perhaps the biggest difference between pure mathematical optimization and optimization in deep learning is that in deep learning we do not optimize for maximum performance directly.

Instead, we use an easier to optimize cost function on a training set and hope that minimizing that would improve the performance on a separate test set.

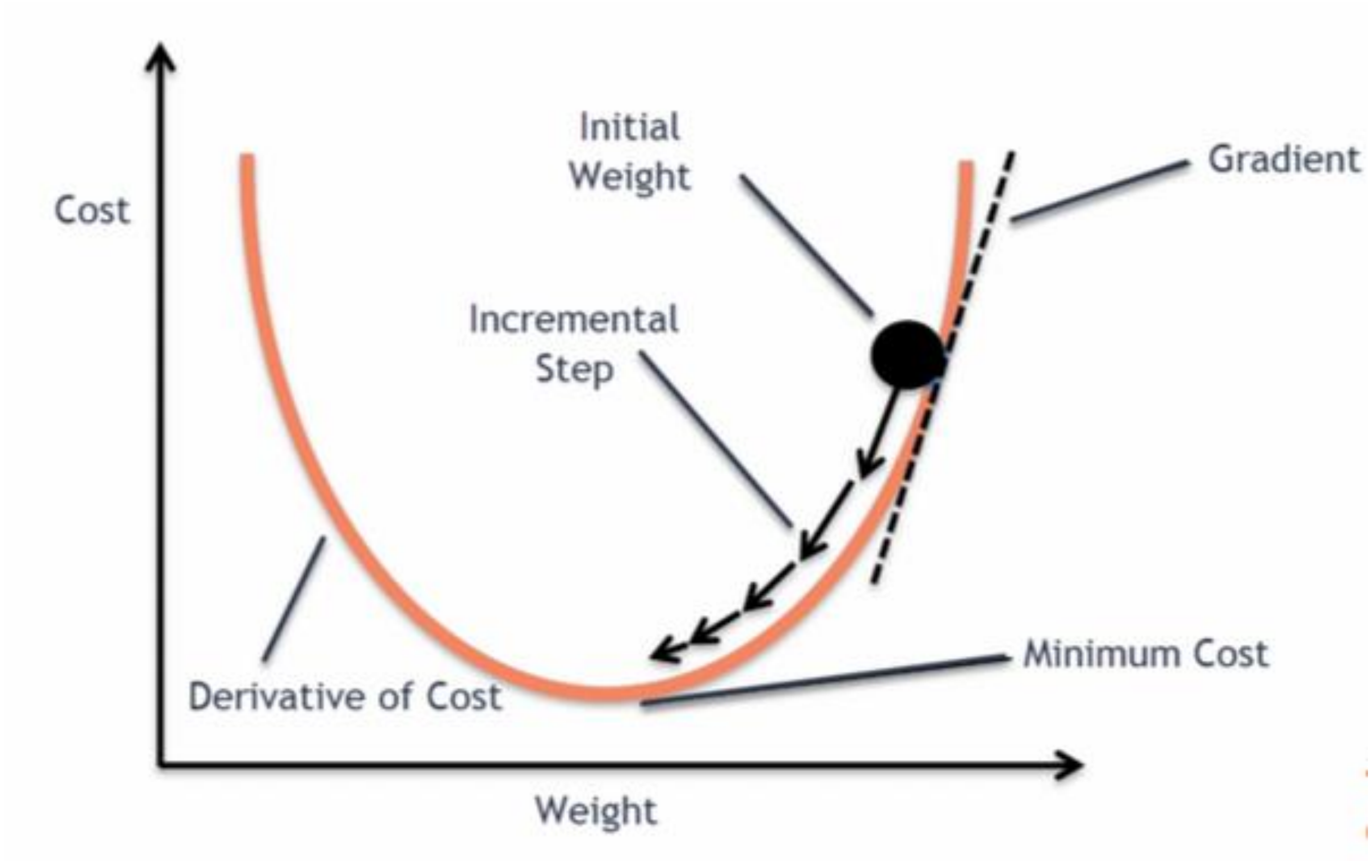
What we want to achieve:
Maximize accuracy = (#correct samples) / (#all samples) on a test set.

What we actually do:
Minimize cross entropy = $\sum p \log(q)$

Where, **Cross-entropy** is a measure of the difference between two probability distributions for a given random variable or set of events. You might recall that information quantifies the number of bits required to encode and transmit an event.

Batch Gradient Descent Algorithm

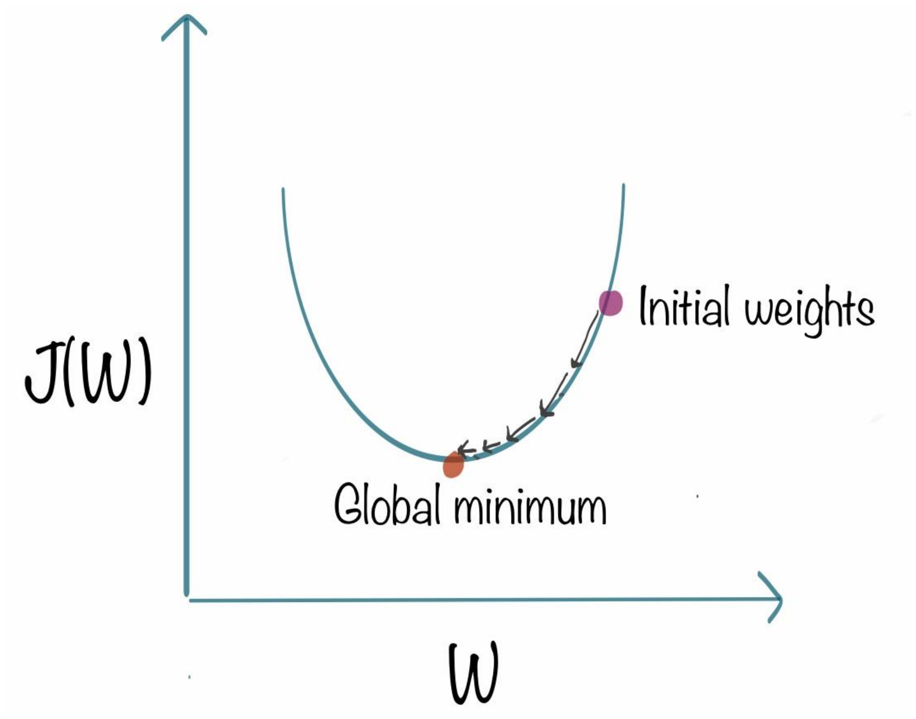
Gradient descent is a first-order iterative optimization **algorithm** for finding a local minimum of a differentiable function. To find a local minimum of a function using **gradient descent**, we take steps proportional to the negative of the **gradient** (or approximate **gradient**) of the function at the current point.



Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to update the parameters of our model.

What algorithms use gradient descent?

Common examples of algorithms with coefficients that can be optimized using gradient descent are **Linear Regression** and **Logistic Regression**.



What is Cost Function?

The terms **cost** and **loss functions** almost refer to the same meaning. But, **loss function** mainly applies for a single training set as compared to the **cost function** which deals with a penalty for a number of training sets or the complete batch. ... The **cost function** is calculated as an average of **loss functions**.

Stochastic Gradient Descent Algorithm

The word ‘***stochastic***‘ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big.

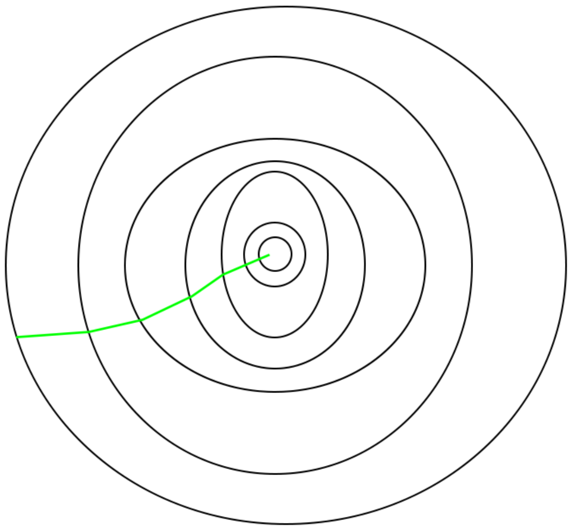
Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

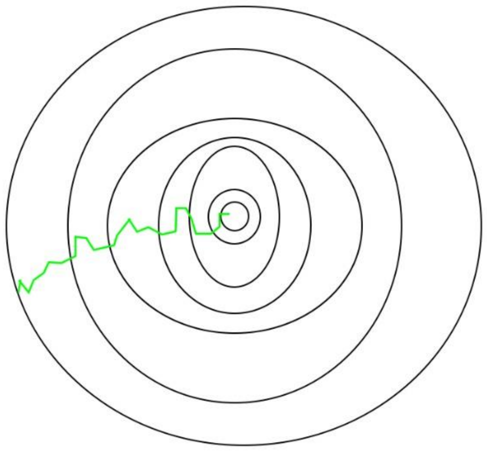
```
$for\ i\ in\ range\ (m):$
$\theta_j = \theta_j - \alpha\,(\hat{y}^i - y^i)\,x_j^i$
```

So, in SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples.
In SGD, since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm. But that doesn't matter all that much because the path taken by the algorithm does not matter, as long as we reach the minima and with significantly shorter training time.

Path taken by Batch Gradient Descent Algorithm



Path taken by Stochastic Gradient Descent Algorithm



```
def SGD(f, theta0, alpha, num_iters):
    """
       Arguments:
       f -- the function to optimize, it takes a single argument
            and yield two outputs, a cost and the gradient
            with respect to the arguments
       theta0 -- the initial point to start SGD from
       num_iters -- total iterations to run SGD for
       Return:
       theta -- the parameter value after SGD finishes
    """
    start_iter = 0
    theta = theta0
    for iter in xrange(start_iter + 1, num_iters + 1):
        _, grad = f(theta)

        # there is NO dot product ! return theta
        theta = theta - (alpha * grad)
```