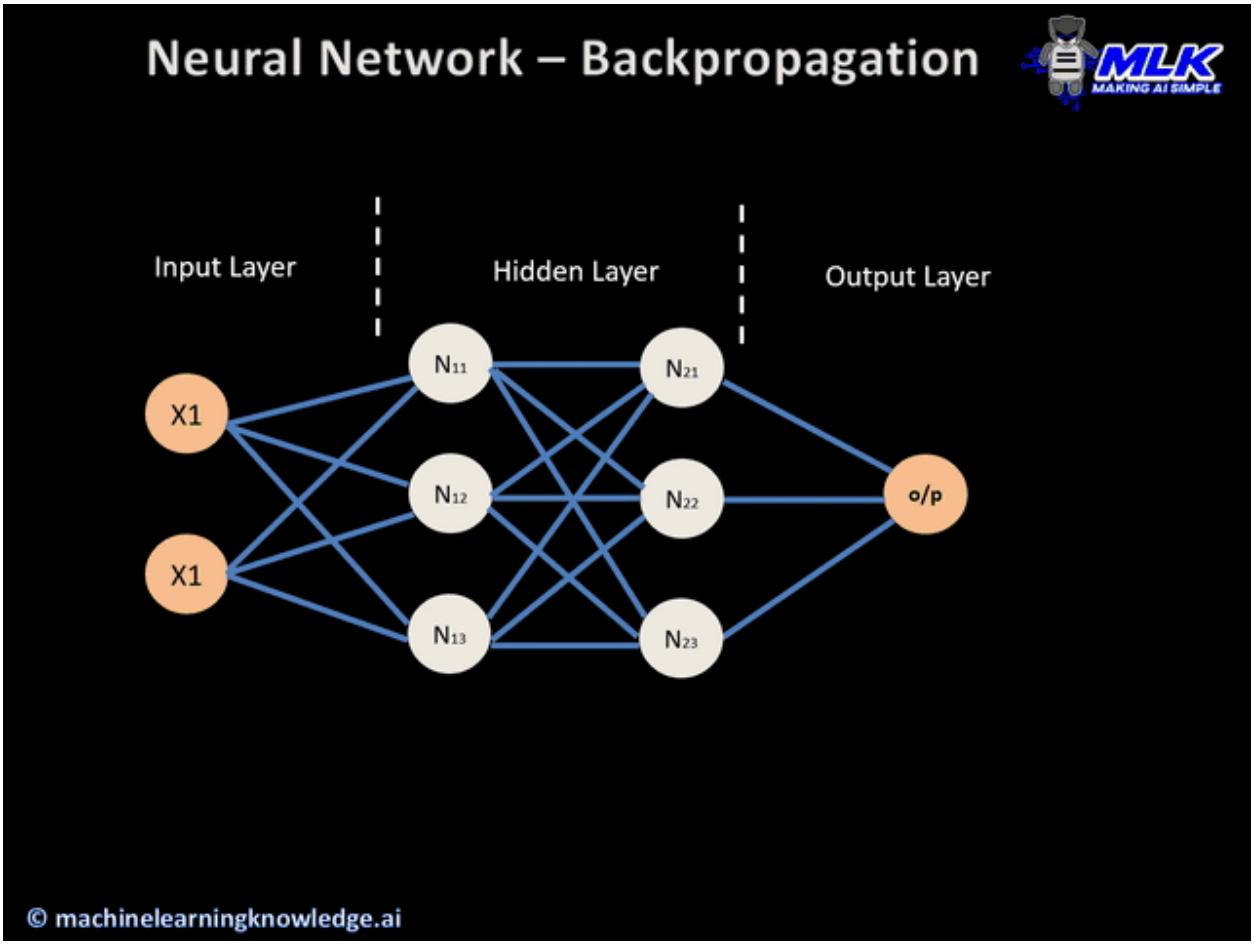
Optimization algorithms!!! Need for Optimizer in Deep Neural Nets ***



courtesy- here

Before our discussion on Optimizer /Optimization algorithm ,let me introduce you to certain terminology…

Forward Propagation:

As the name suggests, the input data is fed in the forward direction through the network. Each hidden layer accepts the input data, processes it as per the activation function and passes to the successive layer.

Loss

'**Loss**' helps us to understand how much the predicted value differ from actual value

Function used to calculate the loss is called as "Loss function"

Loss function is a method of evaluating "*how well your algorithm models your dataset"*. If your predictions are totally off, your loss function will output a higher number. If they're pretty good, it'll output a lower number. As you tune your algorithm to try and improve your model, your loss function will tell you if you're improving or not.

To read more on loss check *here*.

Now comes our hero optimizer!!!

Optimizing the model

What is an optimizer?

Optimizer helps us to *minimize* an error function(*loss function*)or to *maximize* the *efficiency of production*. Optimizer is simply a mathematical function dependent on the model's learnable parameter [***W's (weights/coefficients),b (bias)***]which is used in computing the target value [***yi's***] from the set of inputs[***xi's].***

How does the optimizer minimize the loss?

It minimize the cost function(loss function)by finding the **optimized value for learnable parameters**. This helps in generalizes the model as well.

Types of Optimizer:

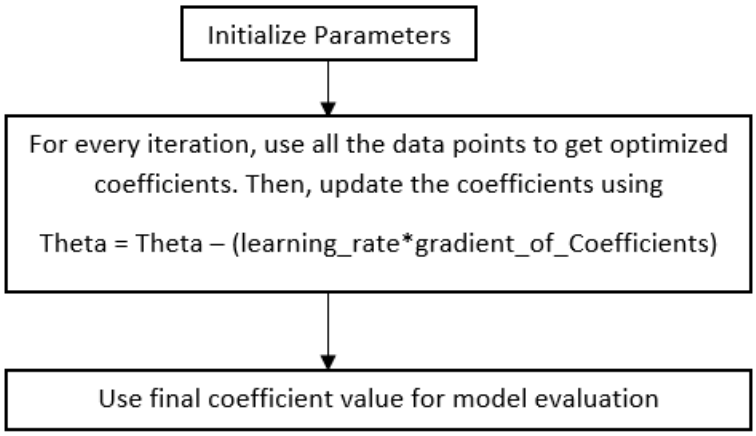
Gradient Descent:

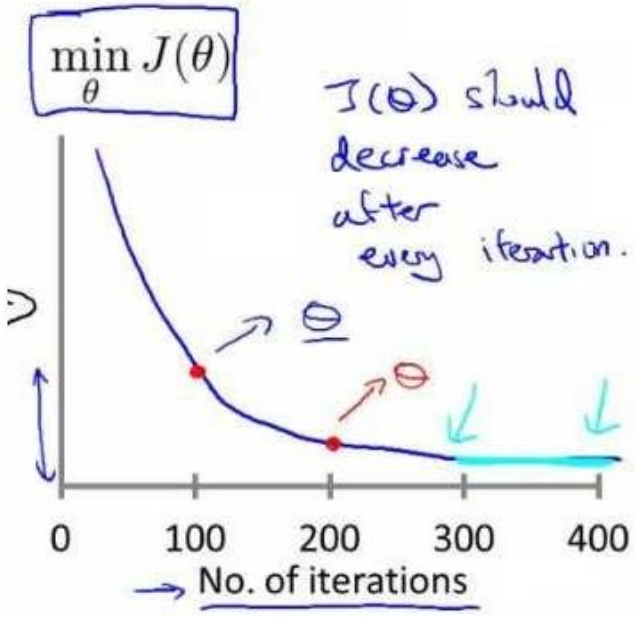
Why do we need Gradient Descent?

In neural network our goal is to train the model to have optimized parameters(w,b) to make better prediction.

we get optimized weights using gradient descent.

Working of Gradient descent:





Cost function for GD

$w_{t+1} = w_t - \eta \nabla w_t$

$b_{t+1} = b_t - \eta \nabla b_t$

where, ∇w_t (gradient of weight)= $\partial L(w, b) / \partial w$,

∇b_t (gradient of bias)= $\partial L(w, b) / \partial b$

"Parameter updates always depends on the gradient"

Note:

When the slope is steep gradient is large.

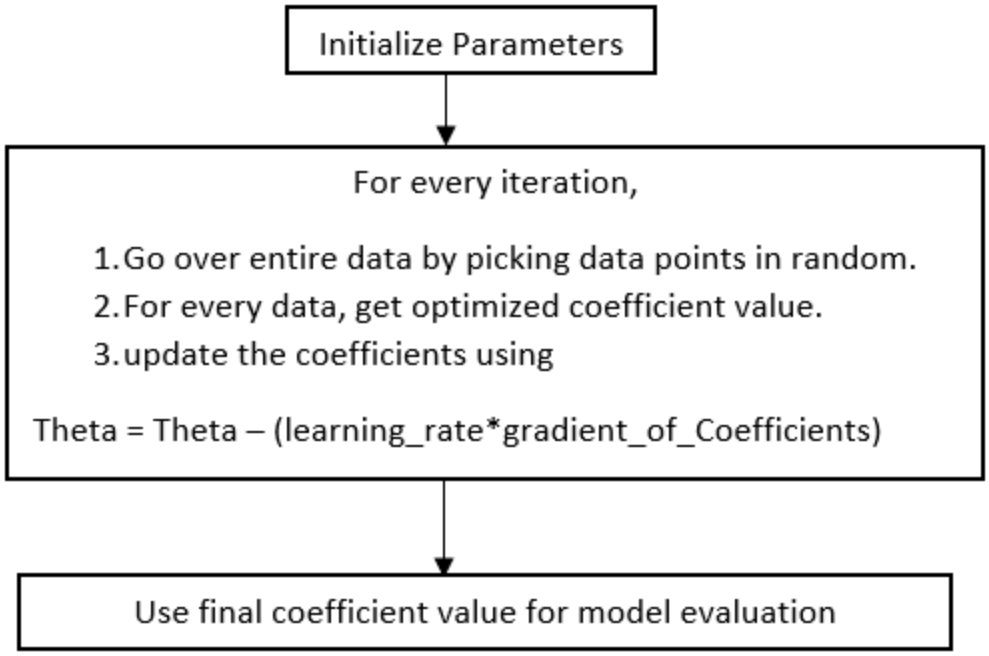
When the slope is steep gradient is small

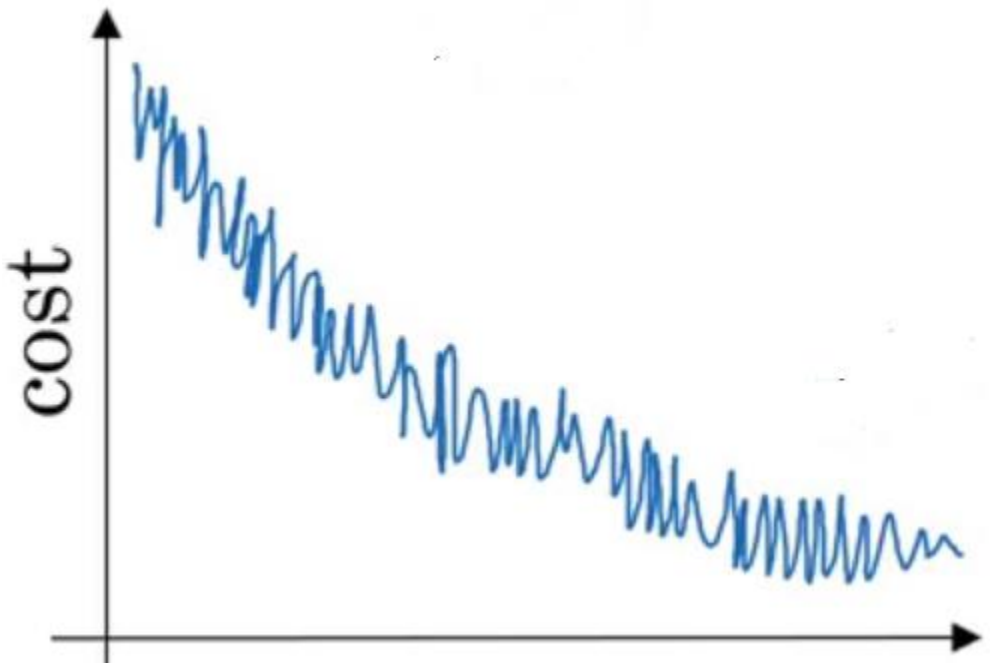
Take away: Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down.

Problem with Gradient descent is *"Slow convergence" and "Computationally heavy".*

Lets discuss the variant of Gradient descent…

Stochastic Gradient Descent





Cost function for the SGD

In stochastic gradient descent we use a single datapoint to calculate the gradient and update the parameter for each iteration.

As the parameters are updated for each single datapoint, update of the parameters and the cost function will be noisy jumping up and down before convergence.*Reason being update of parameters are being localized to single datapoint.*

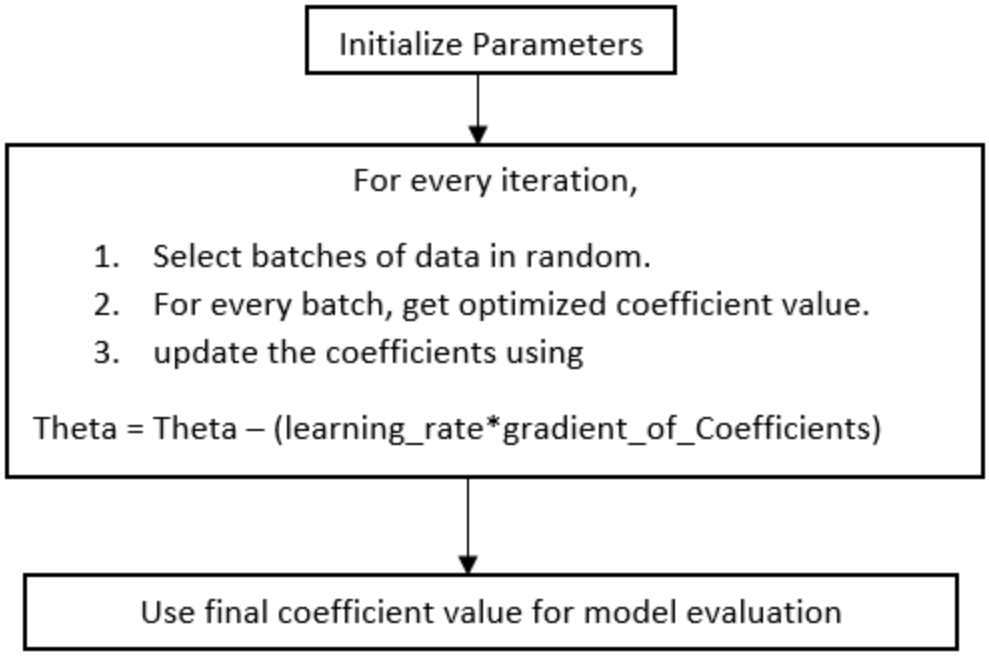
***Take away*: Learning is much faster and convergence is quick for a very large dataset.**

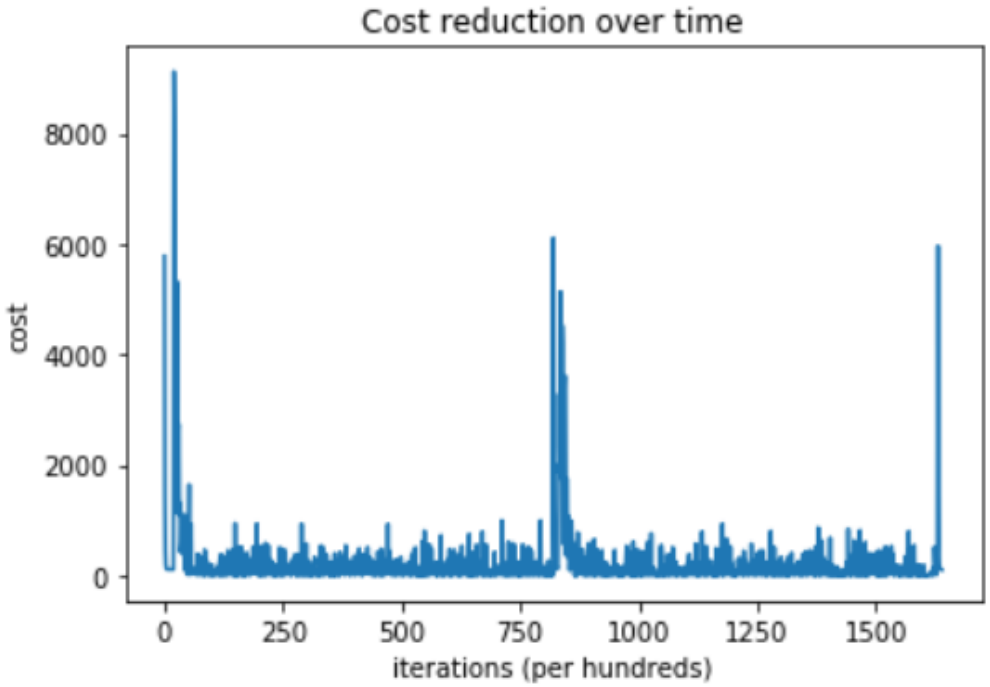
Problem with SGD is that "*as the parameters are frequently updated , cost function fluctuates heavily and take lot of oscillation before convergence"*.

Mini-Batch *Stochastic Gradient Descent:*

Mini- Batch SGD is the combination of concepts of both SGD and Gradient Descent.

Instead of taking single samples for updating the parameter,it will datapoints in batch's.Batch size and number of batch depends upon the dataset.





As we take the small sample of data for updating the parameters,noise is reduced and which helps in faster convergence as well.

Mini-batch SGD is widely used in practice.

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	$\frac{N}{B}$

Something to remember