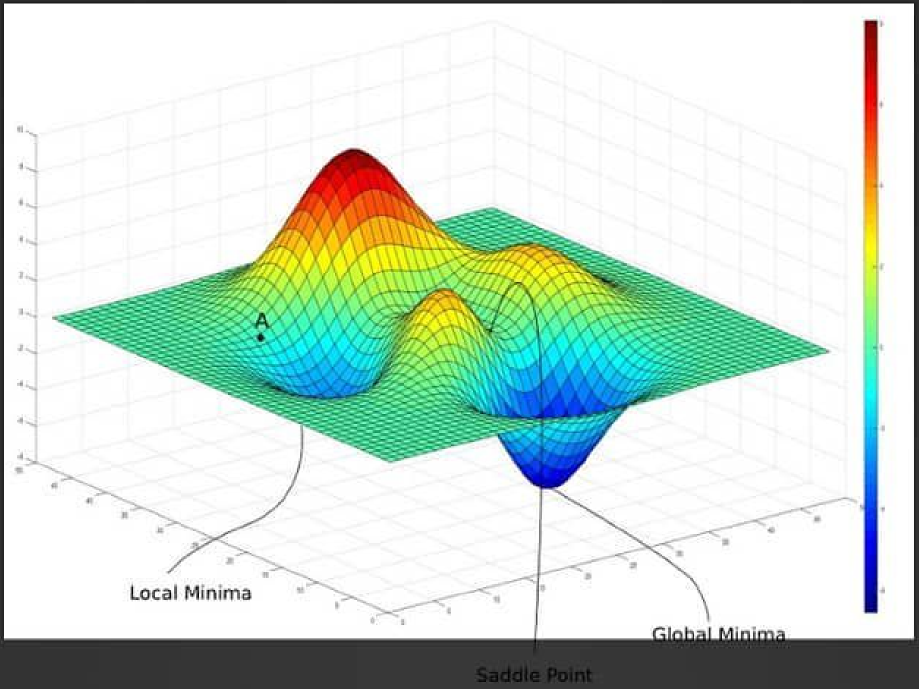
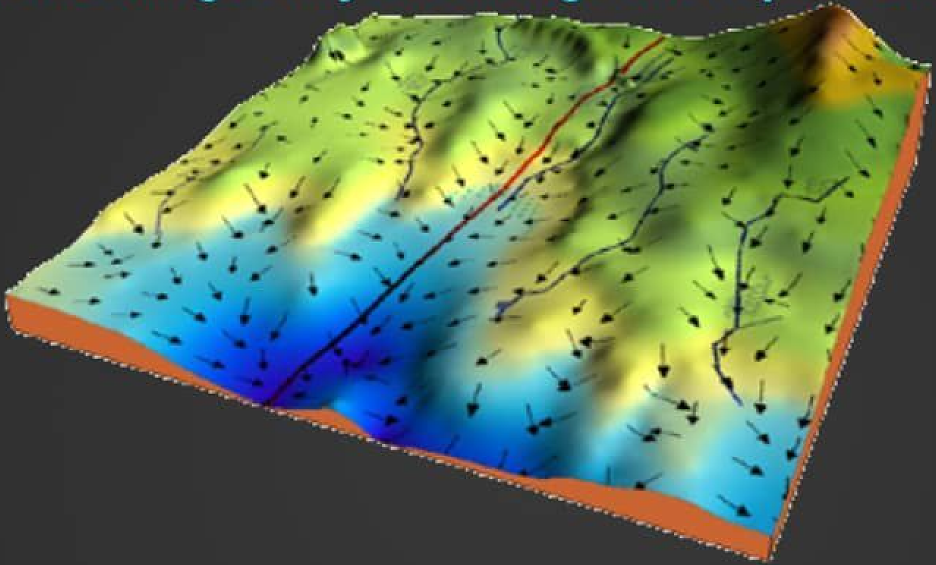
Gradient Descent



- Optimization basically means getting the optimal output for your problem. Generally, while optimizing (other domain problems), we know exactly how our data looks like and what areas we want to improve. But in machine learning, we have no clue how our "new data" looks like, let alone try to optimize on it.
- 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. Parameters refer to coefficients in Linear Regression and weights in neural networks.

Gradient Descent

- In machine learning, we perform optimization on the training data and check its performance on new validation data.
- Gradient means it measures how much the output of a function changes if you change the inputs a little bit.



Suppose you are at the top of a mountain, and you have to reach a lake which is at the lowest point of the mountain (a.k.a valley). A twist is that you are blindfolded and you have zero visibility to see where you are headed. How? The best way is to check the ground near you and observe where the land tends to descend. This will give an idea in what direction you should take your first step. If you follow the descending path, it is very likely you would reach the lake.

Gradient Descent

- Starting at the top of the mountain, we take our first step downhill in the direction specified by the negative gradient. Next, we recalculate the negative gradient (passing in the coordinates of our new point) and take another step in the direction it specifies. We continue this process iteratively until we get to the bottom of the hill, or to a point where we can no longer move downhill–a local minimum.

Here is the basic formula of Gradient descent

$$\mathbf{b} = \mathbf{a} - \gamma \, \nabla \, \mathbf{f}(\mathbf{a})$$

So this formula basically tells you the next position where you need to go, which is the direction of the steepest descent. The bottom of the hill is the cost of the best set of coefficients, the minimum of the function. The goal is to continue to try different values for the coefficients, evaluate their cost and select new coefficients that have a slightly better (lower) cost. Repeating this process enough times will lead to the bottom of the hill and you will know the values of the coefficients that result in the minimum cost.

Gradient Descent

- To start with finding the right values we initialize the values of W (coefficients) and B (bias) with some random numbers and Gradient Descent then starts at that point (somewhere around the top of the hill). Then it takes one step after another in the steepest downside direction (e.g. from the top to the bottom of the hill) till it reaches the point where the cost function is as small as possible.
- How big the steps are that Gradient Descent takes into the direction of the local minimum are determined by the so-called learning rate.
- We will learn more about learning rate and cost function (which we need to minimize) in the next post.

Types of Gradient Descent

Let us look at most commonly used gradient descent algorithms and their implementations.

- Vanilla Gradient Descent or Batch Gradient Descent
- Stochastic Gradient Descent
- Mini Batch Gradient Descent

We will learn about every type in next posts

Variants of Gradient Descent

- Gradient Descent with Momentum
- Nesterov Accelerated Gradient
- ADAGRAD
- Adadelta
- RMSProp
- ADAM

We learn more about them when we start deep learning

Gradient Descent

Let me give a very basic formula to understand gradient descent

update = learning_rate * gradient_of_parameters(Coefficients)

parameters(Coefficients) = parameters(Coefficients) - update

Vanilla Gradient Descent

- This is the simplest form of a gradient descent technique. Here, vanilla means pure / without any adulteration. Its main feature is that we take small steps in the direction of the minima by taking a gradient of the cost function.
- Here, we see that we make an update to the parameters by taking the gradient of the parameters. And multiply it by a learning rate, which is essentially a constant number suggesting how fast we want to go the minimum.
- Learning rate is a hyper-parameter and should be treated with care when choosing its value.

Gradient Descent

Vanilla Gradient Descent

- It calculates the error for each example within the training dataset, but only after all training examples have been evaluated, the model gets updated. This whole process is like a cycle and called a training epoch.
- It produces a stable error gradient and a stable convergence.
- Batch Gradient Descent has the disadvantage that the stable error gradient can sometimes result in a state of convergence that isn't the best the model can achieve.
- It also requires that the entire training dataset is in memory and available to the algorithm.
- It is not good for large datasets as it takes a lot of time.

Gradient Descent

Stochastic Gradient Descent

- In situations when you have large amounts of data, you can use a variation of gradient descent called stochastic gradient descent.
- In this type, the gradient descent procedure described above is run but the update to the coefficients is performed for each training instance, rather than at the end of the batch of instances.
- This can make SGD faster than Batch Gradient Descent, depending on the problem.
- One advantage is that frequent updates allow us to have a pretty detailed rate of improvement.
- Because the coefficients are updated after every training instance, the updates will be noisy jumping all over the place, and so will the corresponding cost function. By mixing up the order for the updates to the coefficients, it harnesses this random walk and avoids it getting distracted or

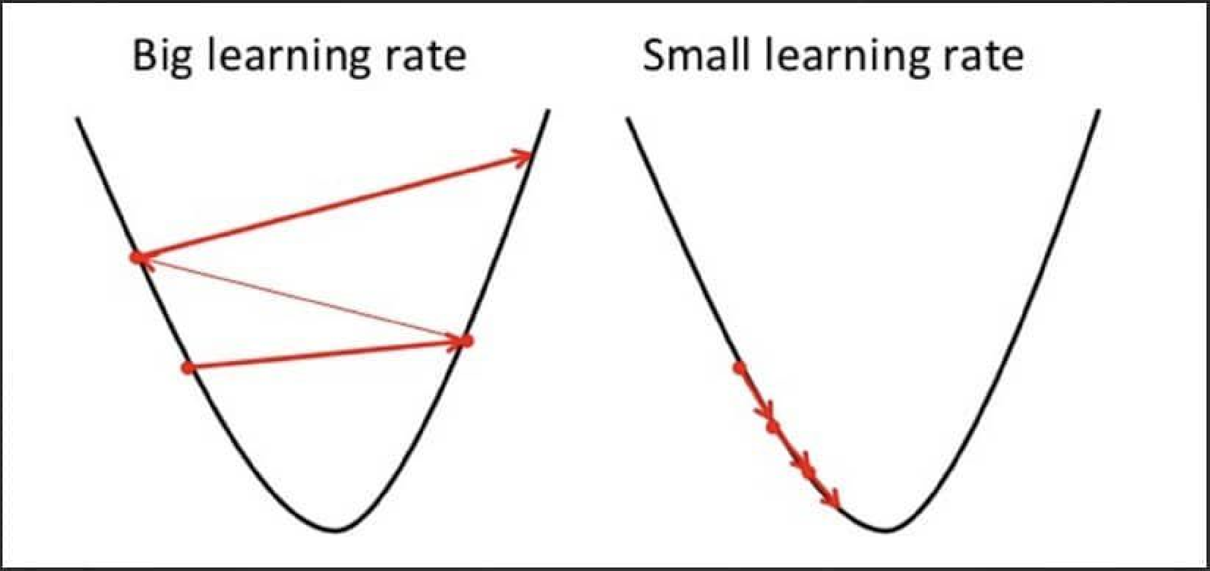
Gradient Descent

Mini Batch Gradient Descent

- Mini-batch Gradient Descent is the go-to method since it's a combination of the concepts of SGD and Batch Gradient Descent. It simply splits the training dataset into small batches and performs an update for each of these batches. Therefore it creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

Gradient Descent

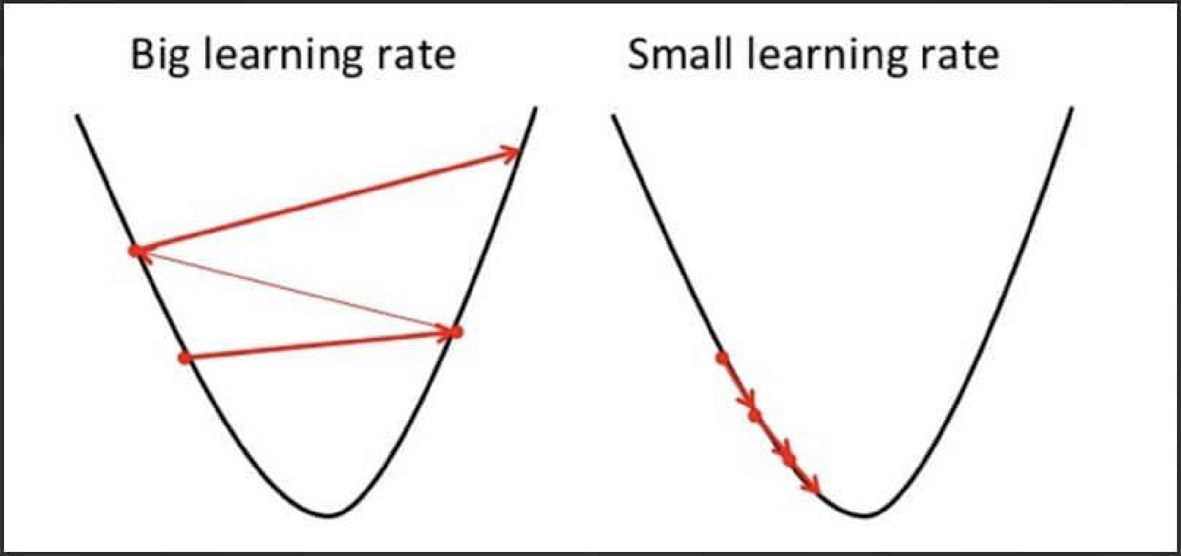
Learning Rate



- Which determines how fast or slow we will move towards the optimal weights (local minima).
- In order for Gradient Descent to reach the local minimum, we have to set the learning rate to an appropriate value, which is neither too low nor too high.
- With a high learning rate, we can cover more ground each step, but we risk overshooting the lowest point since the slope of the hill is constantly changing.

Gradient Descent

Learning Rate

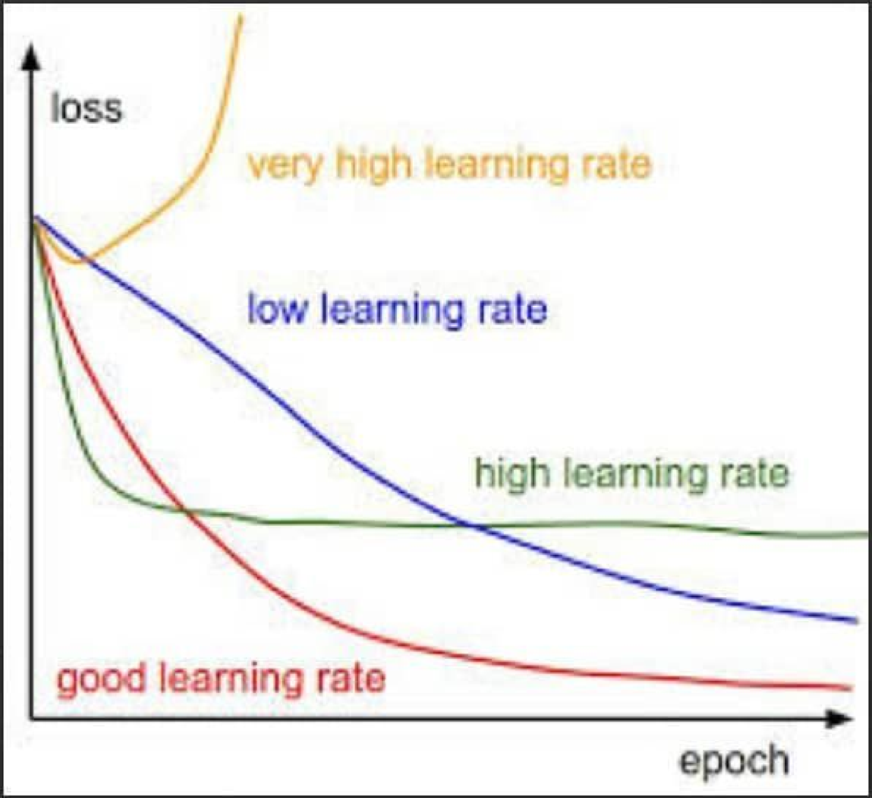


- With a very low learning rate, we can confidently move in the direction of the negative gradient since we are recalculating it so frequently.
- A low learning rate is more precise, but calculating the gradient is time-consuming, so it will take us a very long time to get to the bottom.

Gradient Descent

Learning Rate

- Try with different learning rates and plot the cost function to the number of iterations. Let's see how the cost function(loss) will be with different learning rates. Epoch is nothing but the number of iterations of complete data



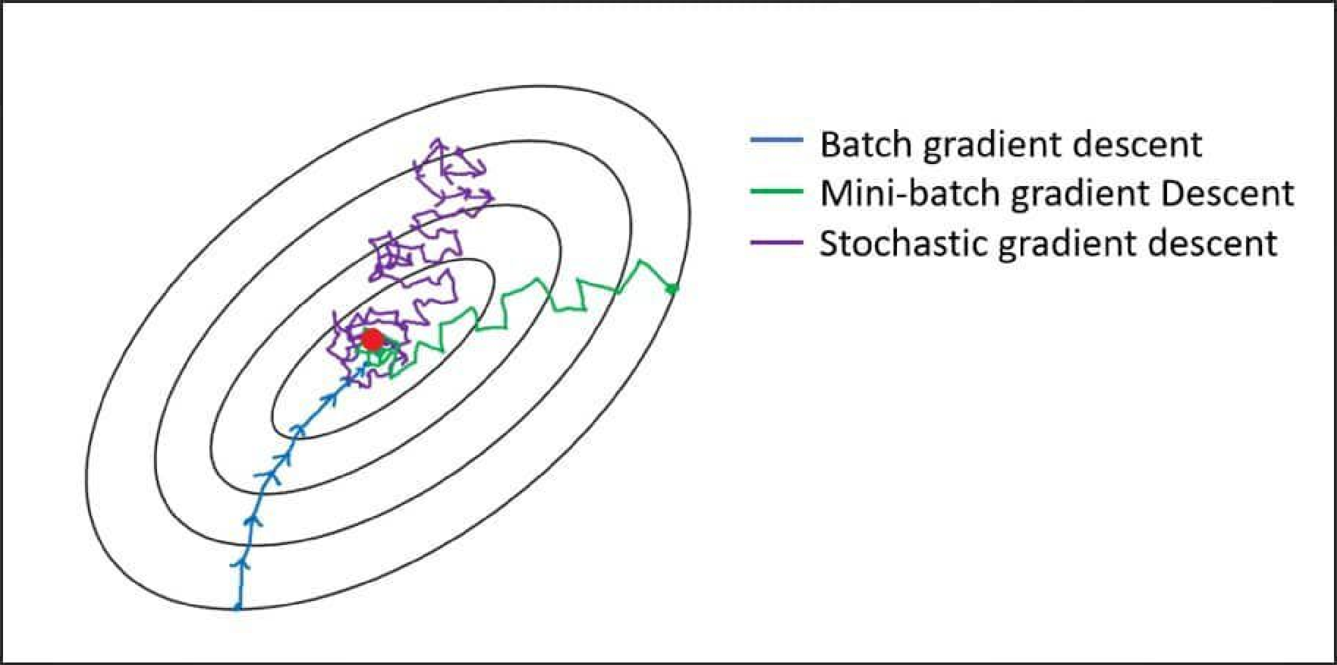
Gradient Descent

Learning Rate

- When Gradient Descent can't decrease the cost-function anymore and remains more or less on the same level, we say it has converged.
- There is no perfect value for a number of iterations. It may take 100 or 1000000 or 100M iterations to converge.
- Another advantage of monitoring Gradient Descent via plots is that you can easily spot if it doesn't work properly, like for example if the cost function is increasing. Mostly the reason for an increasing cost-function, while using Gradient Descent, is that you're learning rate is too high.
- By the way, when you're starting out with gradient descent on a given problem, just simply try 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1 etc. as it's learning rates and look which one performs the best.

Gradient Descent

Lets see how each algorith works through as plot



- Here you can see the GD moves slowly and converges.
- SGD is completely noisy but fast as you can see in the graph because of a single training example to update parameters