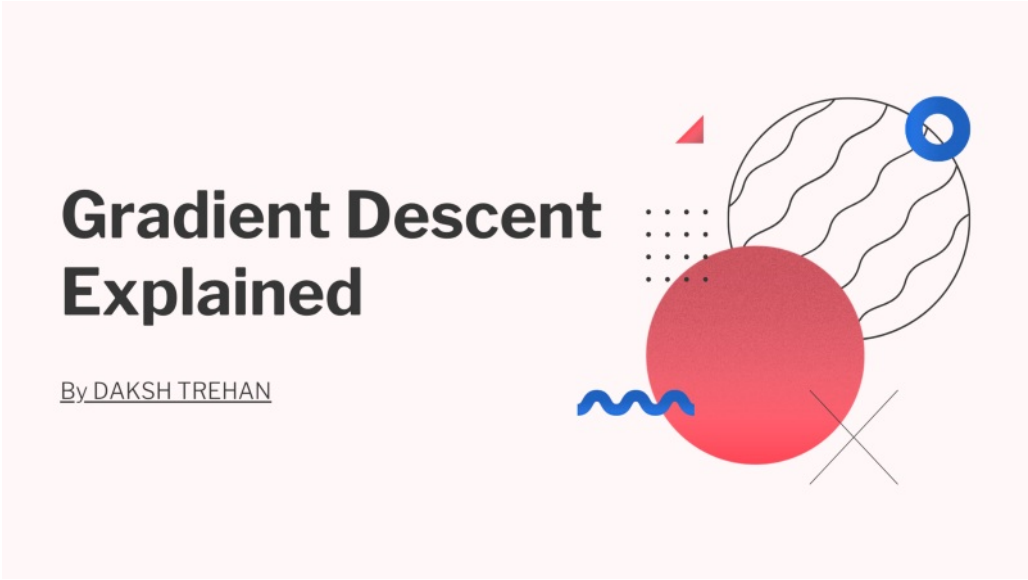
Gradient Descent Explained

A comprehensive guide to Gradient Descent

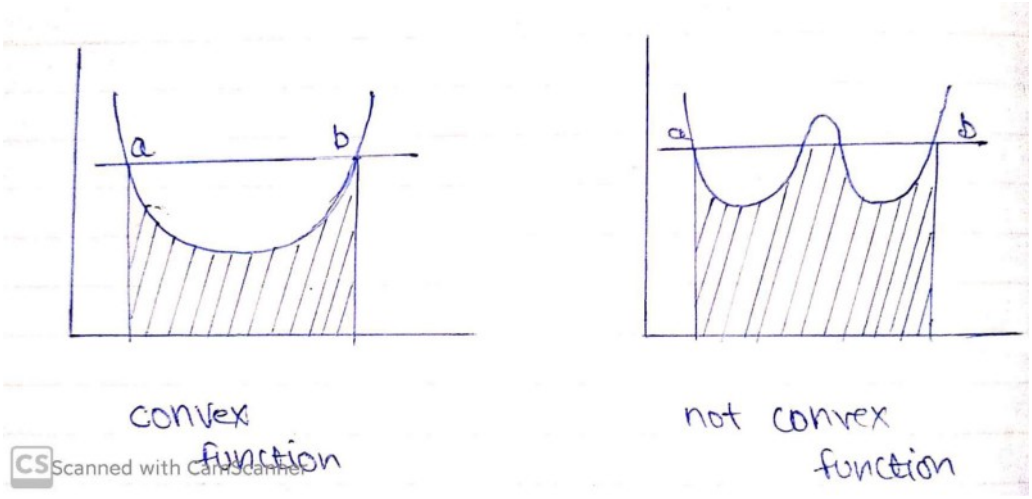


Optimization refers to the task of minimizing/maximizing an objective function f(x) parameterized by x. In machine/deep learning terminology, it's the task of minimizing the cost/loss function J(w) parameterized by the model's parameters w ∈ R^d.
Optimization algorithms (in the case of minimization) have one of the following goals:

1. Find the global minimum of the objective function. This is feasible if the objective function is convex, i.e. any local minimum is a global minimum.
2. Find the lowest possible value of the objective function within its neighborhood. That's usually the case if the objective function is not convex as the case in most deep learning problems.

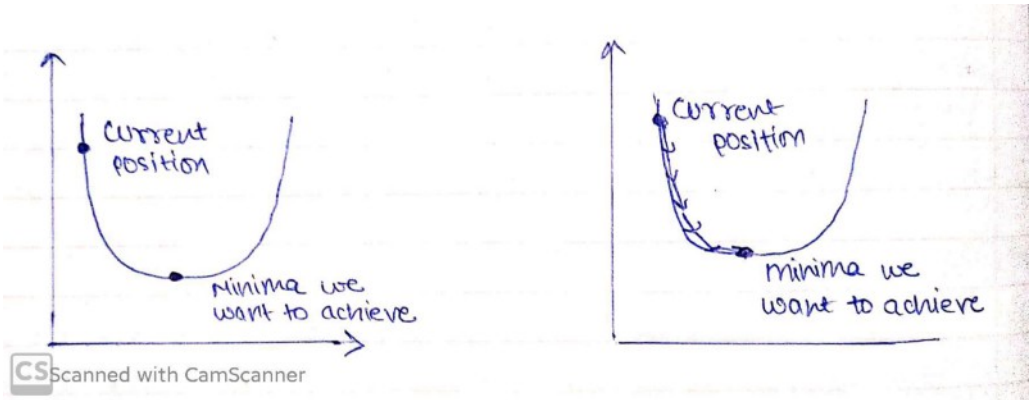
Gradient Descent

Gradient Descent is an optimizing algorithm used in Machine/ Deep Learning algorithms. The goal of Gradient Descent is to minimize the objective convex function f(x) using iteration.



Convex function v/s Not Convex function

Let's see how Gradient Descent actually works by implementing it on a cost function.



Intuition behind Gradient Descent

For ease, let's take a simple linear model.

Error = Y(Predicted)-Y(Actual)

A machine learning model always wants low error with maximum accuracy, in order to decrease error we will intuit our algorithm that you're doing something wrong that is needed to be rectified, that would be done through Gradient Descent.

We need to minimize our error, in order to get pointer to minima we need to walk some steps that are known as alpha(learning rate).

Steps to implement Gradient Descent

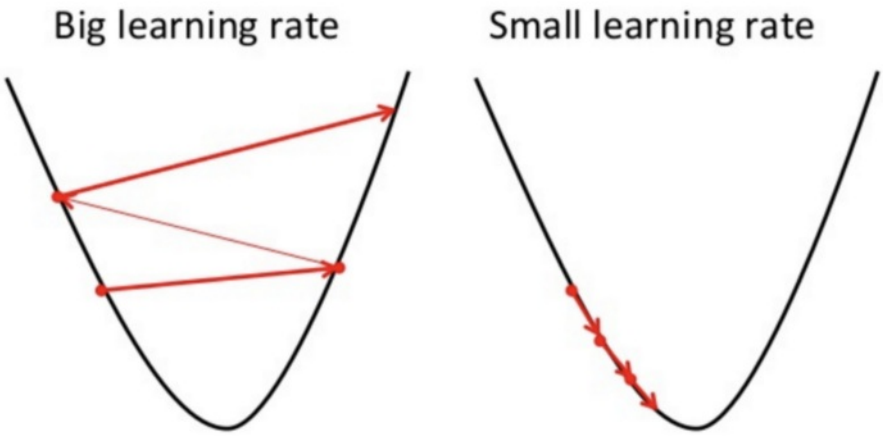
1. Randomly initialize values.
2. Update values.

$$weight^{(new)} = weight^{(old)} - constant \frac{\partial J(\Theta)}{\partial weight}$$

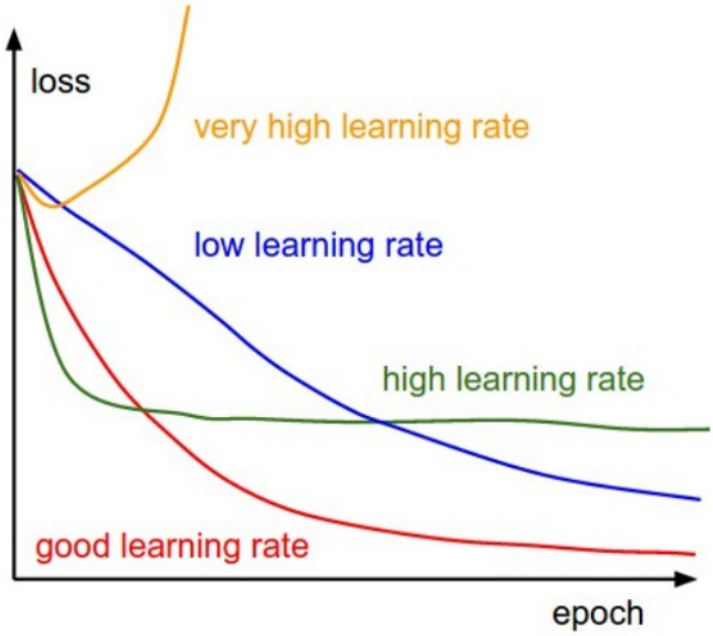
3. Repeat until slope =0

A derivative is a term that comes from calculus and is calculated as the slope of the graph at a particular point. The slope is described by drawing a tangent line to the graph at the point. So, if we are able to compute this tangent line, we might be able to compute the desired direction to reach the minima.

Learning rate must be chosen wisely as:
1. if it is too small, then the model will take some time to learn.
2. if it is too large, model will converge as our pointer will shoot and we'll not be able to get to minima.



Big Learning rate v/s Small Learning rate, Source

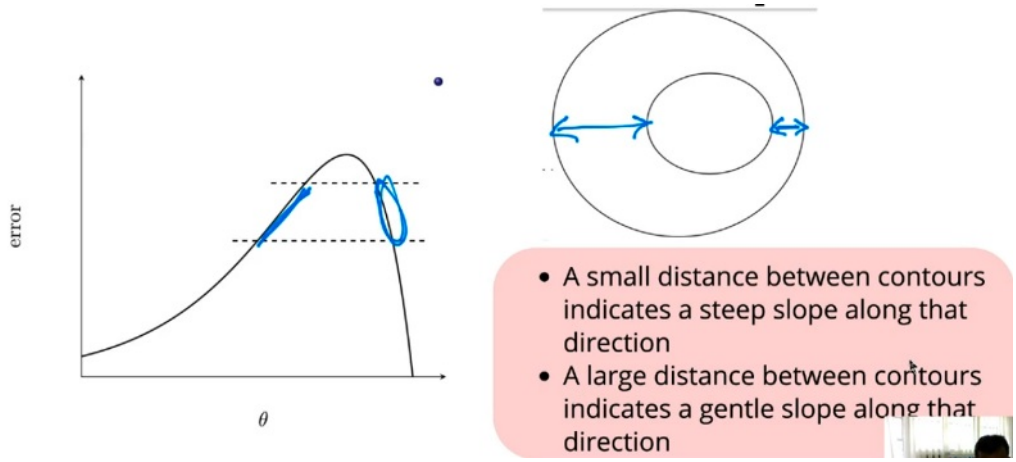


Gradient Descent with different learning rates, Source

Vanilla gradient descent, however, can't guarantee good convergence, due to following reasons:

- Picking an appropriate learning rate can be troublesome. A learning rate that is too low will lead to slow training and a higher learning rate will lead to overshooting of slope.

- Another key hurdle faced by Vanilla Gradient Descent is it avoid getting trapped in local minima; these local minimas are surrounded by hills of same error, which makes it really hard for vanilla Gradient Descent to escape it.



Contour maps visualizing gentle and steep region of curve, Source

In simple words, every step we take towards minima tends to decrease our slope, now if we visualize, in steep region of curve derivative is going to be large therefore steps taken by our model too would be large but as we will enter gentle region of slope our derivative will decrease and so will the time to reach minima.

Momentum Based Gradient Descent

If we consider, Simple Gradient Descent completely relies only on calculation i.e. if there are 100 steps, then our model would run Simple Gradient Descent for 100 times.

In laymen language, suppose a man is walking towards his home but he don't know the way so he ask for direction from by passer, now we expect him to walk some distance and then ask for direction but man is asking for direction at every step he takes, that is obviously more time consuming, now compare man with Simple Gradient Descent and his goal with minima.

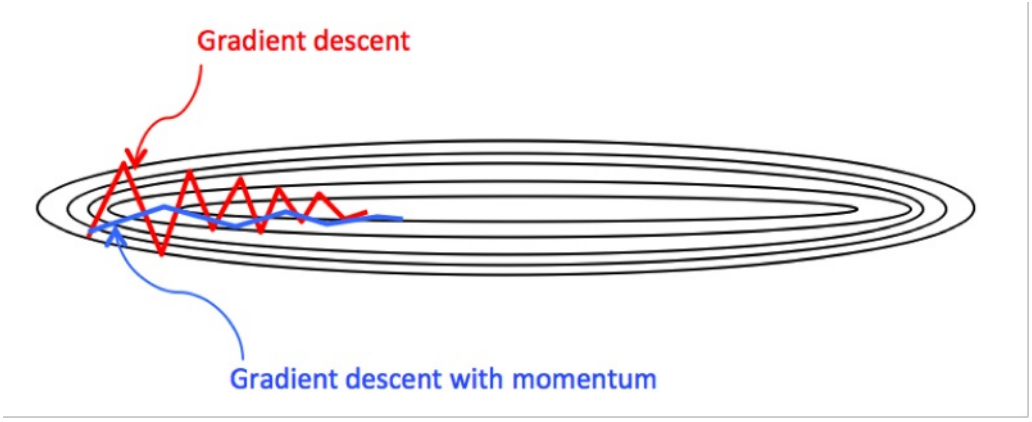
In order to overcome the shortcoming of vanilla Gradient Descent, we introduced momentum based Gradient Descent where the idea is to decrease the computation time and that is achieved when we introduce the concept of experience i.e. the confidence using previous steps.

Pseudocode for momentum based Gradient Descent:

```
update = learning_rate * gradient
velocity = previous_update * momentum
parameter = parameter + velocity – update
```

In this way rather than computing new steps again and again we are

averaging the decay and as decay increases its effect in decision making decreases and thus the older the step less effect on decision making. More the history more bigger steps will be taken. Even in the gentle region momentum based Gradient Descent is able to take large steps because of momentum it carries along. Momentum helps accelerate vanilla Gradient Descent in the relevant direction and dampens oscillations.



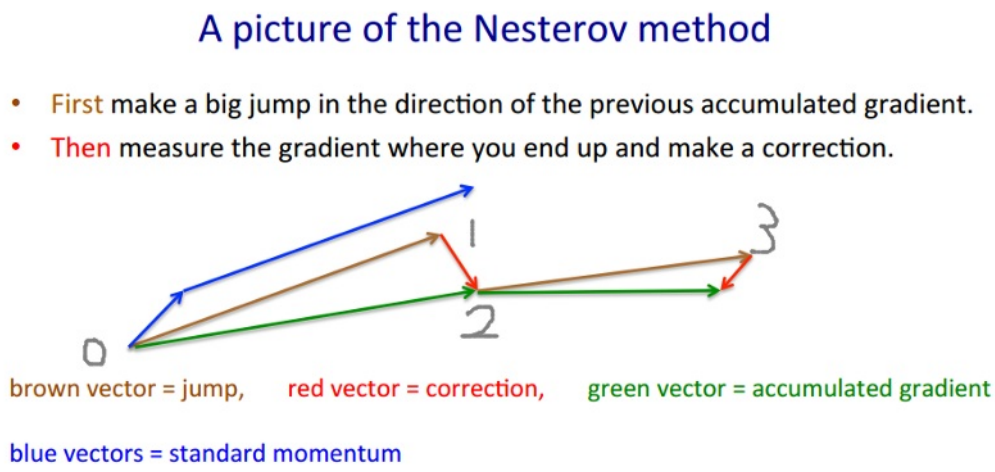
Vanilla Gradient Descent v/s Gradient Descent with Momentum, Source

But due to larger steps it overshoots its goal by longer distance as it oscillate around minima due to steep slope, but despite such hurdles it is faster than vanilla Gradient Descent.

In simple words, suppose a man want to reach destination that is 1200m far and he doesn't know the path, so he decided that after every 250m he will ask for direction, now if he asked direction for 5 times he would've travelled 1250m that's he has already passed his goal and to achieve that goal he would need to trace his steps back. Similar is the case of Momentum based GD where due to high experience our model is taking larger steps that is leading to overshooting and hence missing the goal but to achieve minima model have to trace back its steps.

Nesterov Accelerated Gradient Descent(NAG)

To overcome the problems of momentum based Gradient Descent we use NAG, in this we move first and then compute gradient so that if our oscillations overshoots then it must be insignificant as compared to that of Momentum Based Gradient Descent.



Intuition behind NAG, Source

Nesterov accelerated Gradient(NAG) is a way to provide history to our momentum. We can now adequately look forward by computing the angle not w.r.t. to our present parameters θ.

With momentum:

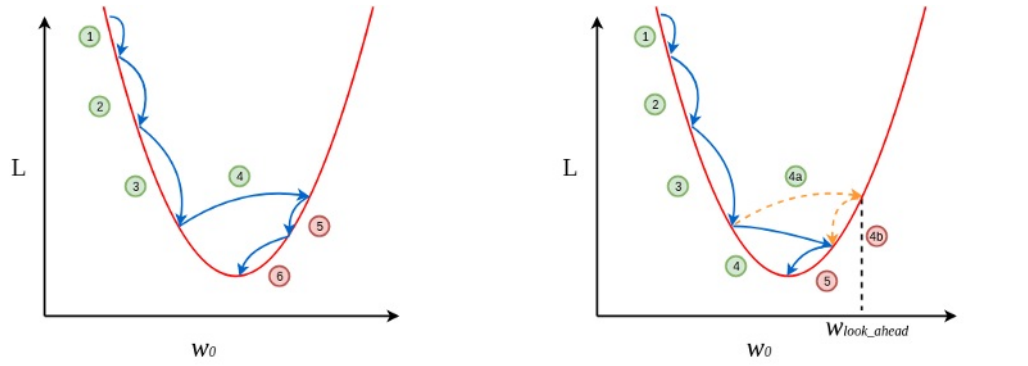
$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

With NAG:

$$w_{look\_ahead} = w_t - \gamma \cdot update_{t-1}$$
$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_{look\_ahead}$$
$$w_{t+1} = w_t - update_t$$

We follow similar update rule for b_t.

Let's see NAG in action!



(a) Momentum-Based Gradient Descent (b) Nesterov Accelerated Gradient Descent

$$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)} \qquad \bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$$

Momentum based Gradient Descent v/s Nesterov Accelerated Gradient Descent, Source

In figure (a), update 1 is positive i.e., the gradient is negative because as ***wo*** increases ***L*** decreases. Even update 2 is positive as well and you can

see that the update is slightly larger than update 1, thanks to momentum. By now, you should be convinced that update 3 will be bigger than both update 1 and 2 simply because of momentum and the positive update history. Update 4 is where things get interesting. In vanilla momentum, due to the positive history, the update overshoots and the our pointer recovers by performing negative updates.

But in NAG's case, every update happens in two steps—first, a partial update, where we get to the *look ahead* point and then the final update, see figure (b). Initial updates of NAG are pretty similar to the momentum-based method as both the updates (partial and final) are positive in those cases. But the real difference becomes apparent during update 4. As usual, each update happens in two stages, the partial update (4a) is positive, but the final update (4b) would be negative as the calculated gradient at ***w_lookahead*** would be negative (convince yourself by observing the graph). This negative final update slightly reduces the overall magnitude of the update, still resulting in an overshoot but a smaller one when compared to the vanilla momentum-based gradient descent. And that my friend, is how NAG helps us in reducing the overshoots, i.e. making us take shorter U-turns.

Since we can adjust our updates to the slope of error function and boost SGD thus, we iterate it for every step.

Gradient Descent Strategies.

Stochastic Gradient Descent

Instead of going through all examples, Stochastic Gradient Descent (SGD) performs the parameters update on each example (x^i,y^i). Therefore, learning happens on every example:

$$w = w - \alpha \nabla_w J(x^i, y^i; w) \tag{7}$$

- Shuffle the training data set to avoid pre-existing order of examples.
- Partition the training data set into m examples.

Advantages : —

a. Easy to fit in memory
b. Computationally fast
c. Efficient for large dataset

Disadvantages :-

a. Due to frequent updates steps taken towards minima are very noisy.
b. Noise can make it large to wait.
c. Frequent updates are computationally expensive.

Batch Gradient Descent

Batch Gradient Descent is a greedy approach, when we add all examples on each iteration when performing the updates to the parameters. Therefore, for each update, we have to sum over all examples.

$$w = w - \alpha \nabla_w J(w) \tag{6}$$

Advantages :-

a. Less noisy steps
b. produces stable GD convergence.
c. Computationally efficient as all resources aren't used for single sample but rather for all training samples

Disadvantages :-

a. Additional memory might be needed.
b. It can take long to process large database.
c. Approximate gradients

Mini Batch Gradient Descent

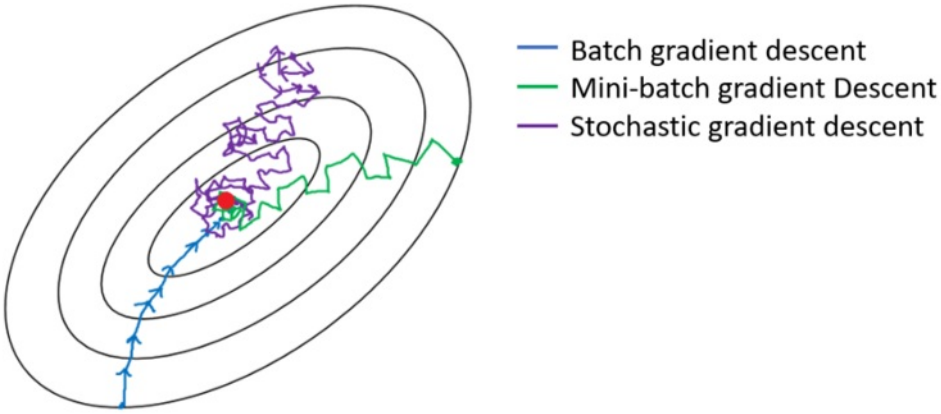
Instead of going over all examples, Mini-batch Gradient Descent sums up over lower number of examples based on the batch size. Therefore, learning happens on each mini-batch of *b* examples.

It is sum of both Batch Gradient Descent and Stochastic Gradient Descent.

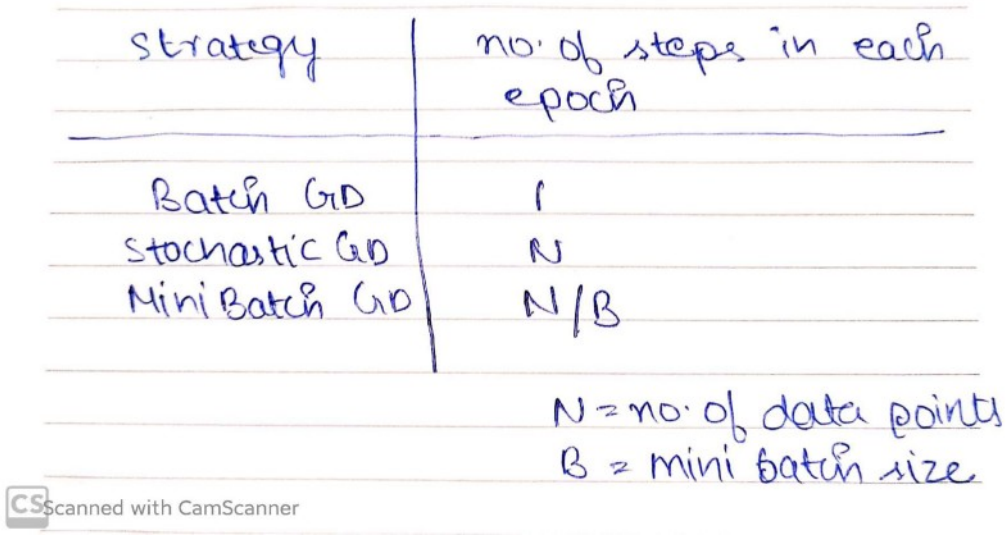
$$w = w - \alpha \nabla_w J(x^{\{i:i+b\}}, y^{\{i:i+b\}}; w) \tag{7}$$

Advantages :-

a. Easy fit in memory.
b. Computationally efficient.
c. Stable error go and convergence.



Batch v/s Stochastic v/s Mini Batch Gradient Descent, Source



What if we cater sparse data to Gradient Descent?

In case of sparse data, we would experience sparse ON(1) features and more frequent OFF(0) features, now, most of the time gradient update will be NULL as derivative is zero in most cases and when it will be one, the steps would be too small to reach minima.

For frequent features we require low learning rate, but for high features we require high learning rate.

So, in order to boost our model for sparse nature data, we need to chose adaptive learning rate.

Conclusion

Hopefully, this article has not only increased your understanding of Gradient Descent but also made you realize machine learning is not difficult and is already happening in your daily life.

As always, thank you so much for reading, and please share this article if you found it useful! :)

To know more about parameters optimization techniques, follow :-

Parameters Optimization Explained
*A brief yet descriptive guide to Gradient Descent, ADAM, ADAGRAD, RMSProp*towardsdatascience.com

---

References:

[1] Gradient Descent Algorithm and Its Variants by Imad Dabbura

[2] Learning Parameters, Part 2: Momentum-Based & Nesterov Accelerated Gradient Descent by Akshay L Chandra

[3] An overview of gradient descent optimization algorithms by Sebastian Ruder

[4] Understanding the Mathematics behind Gradient Descent by Parul Pandey.

[5] Deep Learning (padhAI) by Dr. Mitesh Khapra and Dr. Pratyush Kumar

The cover template is designed by me on canva.com, the source is mentioned on every visual, and the un-mentioned visuals are from my notebook.

Feel free to connect:

Join me at www.dakshtrehan.com

LinkedIN ~ https://www.linkedin.com/in/dakshtrehan/

Instagram ~ https://www.instagram.com/_daksh_trehan_/

Github ~ https://github.com/dakshtrehan

Check my other articles:-

Detecting COVID-19 using Deep Learning.

Logistic Regression Explained

Linear Regression Explained

Determining perfect fit for your ML Model.

Serving Data Science to a Rookie.

Relating Machine Learning techniques to Real Life.

Follow for further Machine Learning/ Deep Learning blogs.

Cheers.

By Daksh Trehan on May 22, 2020.

Canonical link

Exported from Medium on July 15, 2020.