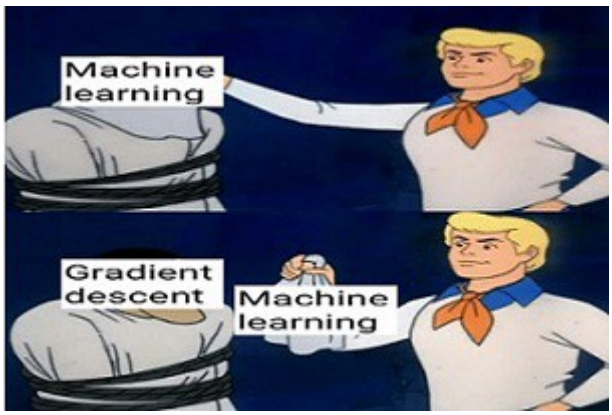
Batch, Mini Batch & Stochastic Gradient Descent

Sushant Patrikar
Oct 1, 2019 · 5 min read

In this era of deep learning, where machines have already surpassed human intelligence it's fascinating to see how these machines are learning just by looking at examples. When we say that we are *training* the model, it's gradient descent behind the scenes who trains it.



Machine Learning behind the scenes (Source: https://me.me/i/machine-learning-gradient-descent-machine-learning-machine-learning-behind-the-ea8fe9fc64054eda89232d7ffc9ba60e)

So let's dive deeper in the deep learning models to have a look at gradient descent and its siblings.

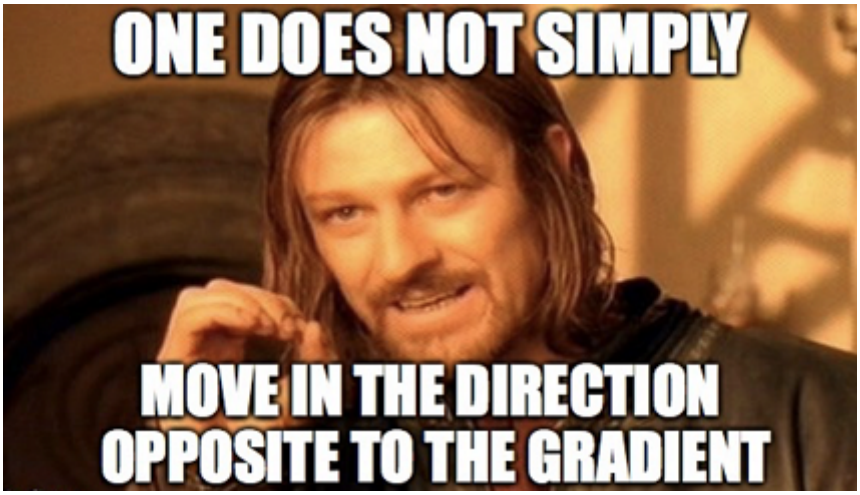
Gradient Descent

This is what Wikipedia has to say on Gradient descent

> ***Gradient descent*** *is a first-order iterative optimization algorithm for finding the minimum of a function*

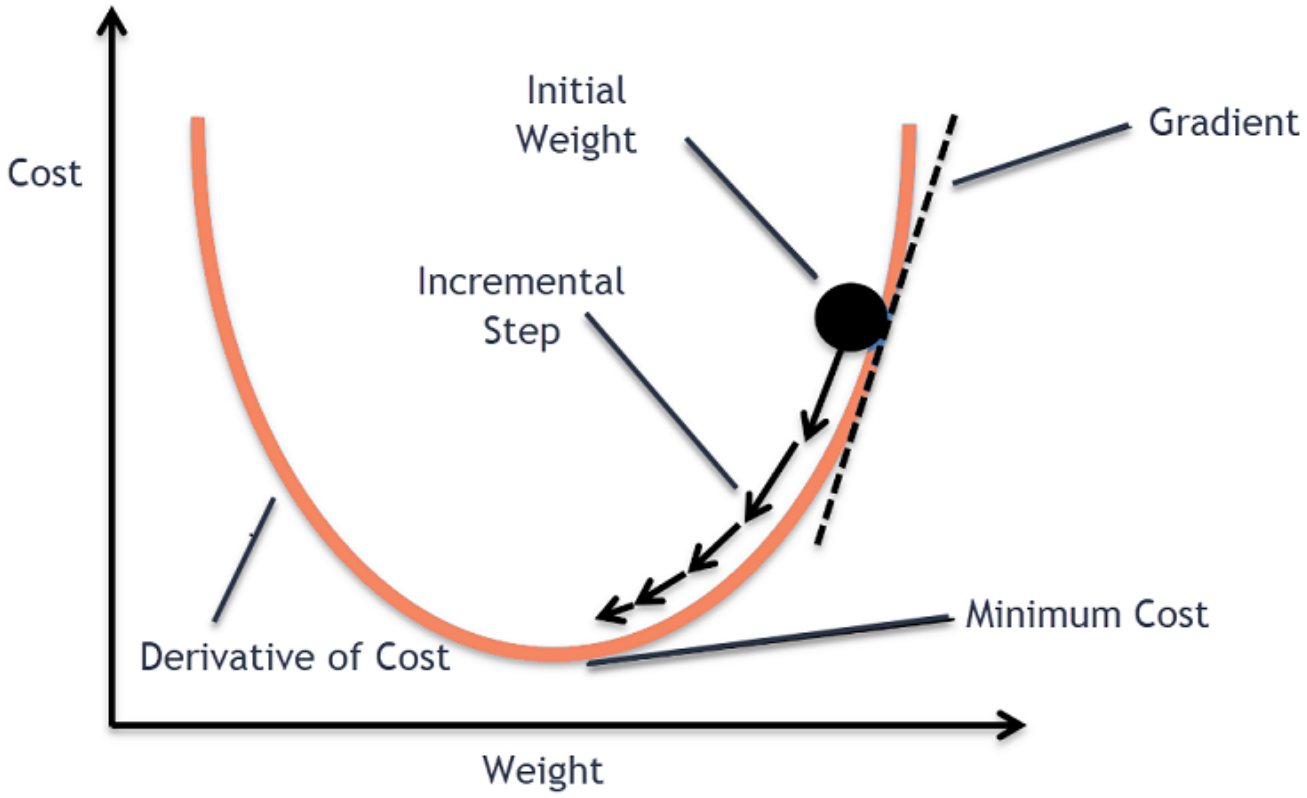
This seems little complicated, so let's break it down.

The goal of the gradient descent is to minimise a given function which, in our case, is the loss function of the neural network. To achieve this goal, it performs two steps iteratively.



Source: https://hackernoon.com/the-reason-behind-moving-in-the-direction-opposite-to-the-gradient-f9566b95370b

1. Compute the slope (gradient) that is the first-order derivative of the function at the current point
2. Move-in the opposite direction of the slope increase from the current point by the computed amount



Gradient descent (Source: https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1)

So, the idea is to pass the training set through the hidden layers of the neural network and then update the parameters of the layers by computing the gradients using the training samples from the training dataset.

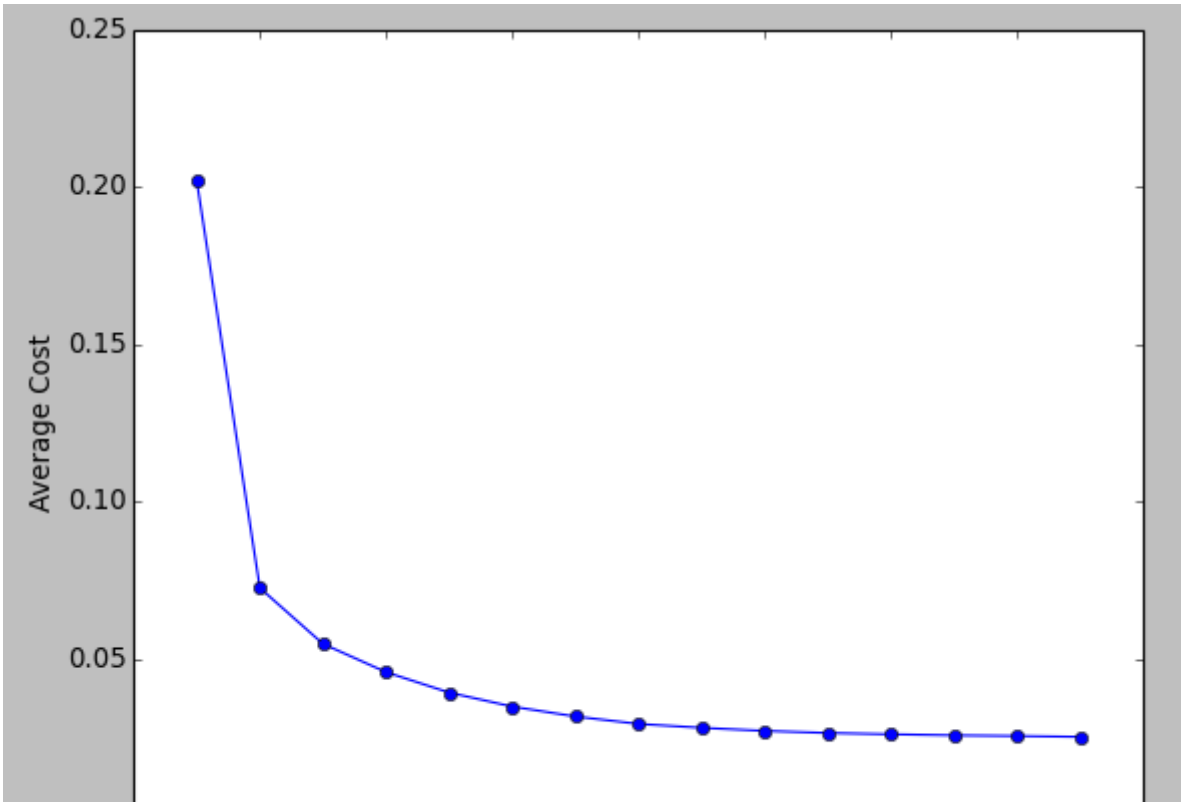
Think of it like this. Suppose a man is at top of the valley and he wants to get to the bottom of the valley. So he goes down the slope. He decides his next position based on his current position and stops when he gets to the bottom of the valley which was his goal.

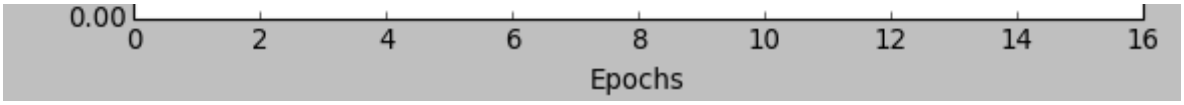
There are different ways in which that man (weights) can go down the slope. Let's look into them one by one.

Batch Gradient Descent

In Batch Gradient Descent, all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters. So that's just one step of gradient descent in one epoch.

Batch Gradient Descent is great for convex or relatively smooth error manifolds. In this case, we move somewhat directly towards an optimum solution.





Cost vs Epochs (Source: https://www.bogotobogo.com/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php)

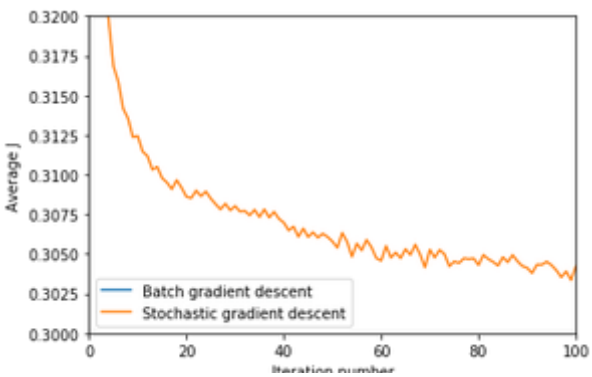
The graph of cost vs epochs is also quite smooth because we are averaging over all the gradients of training data for a single step. The cost keeps on decreasing over the epochs.

Stochastic Gradient Descent

In Batch Gradient Descent we were considering all the examples for every step of Gradient Descent. But what if our dataset is very huge. Deep learning models crave for data. The more the data the more chances of a model to be good. Suppose our dataset has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples. This does not seem an efficient way. To tackle this problem we have Stochastic Gradient Descent. In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a single step. We do the following steps in **one epoch** for SGD:

1. Take an example
2. Feed it to Neural Network
3. Calculate it's gradient
4. Use the gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the examples in training dataset

Since we are considering just one example at a time the cost will fluctuate over the training examples and it will **not** necessarily decrease. But in the long run, you will see the cost decreasing with fluctuations.



Cost vs Epochs in SGD (Source: https://adventuresinmachinelearning.com/stochastic-gradient-descent/)

Source: https://towardsdatascience.com/optimizers-be-deeps-appetizers-511f3706aa67

Also because the cost is so fluctuating, it will never reach the minima but it will keep dancing around it.

SGD can be used for larger datasets. It converges faster when the dataset is large as it causes updates to the parameters more frequently.

Mini Batch Gradient Descent

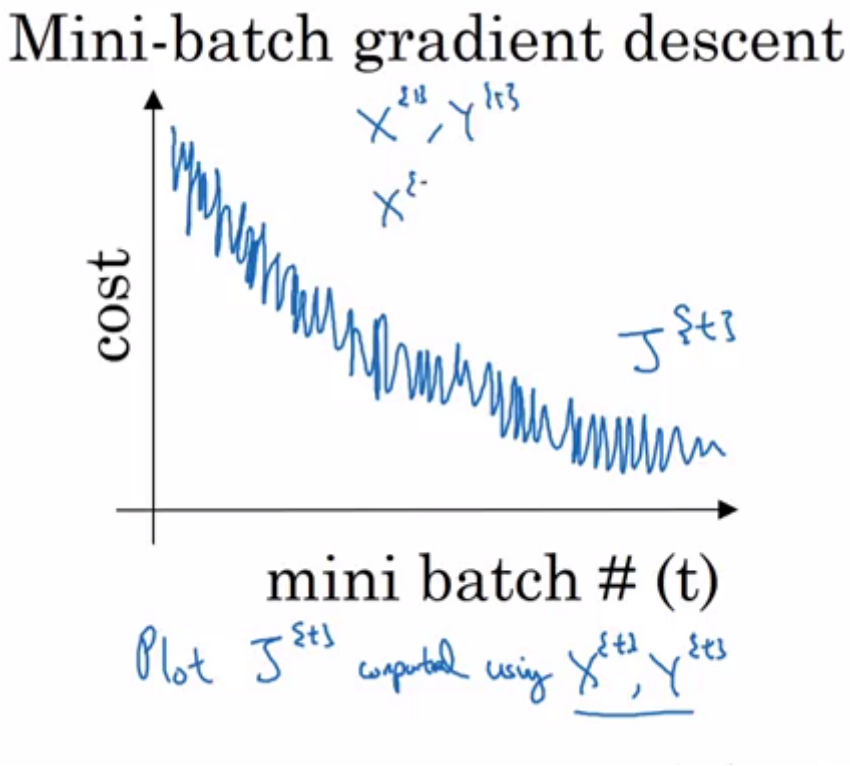
We have seen the Batch Gradient Descent. We have also seen the Stochastic Gradient Descent. Batch Gradient Descent can be used for smoother curves. SGD can be used when the dataset is large. Batch Gradient Descent converges directly to minima. SGD converges faster for larger datasets. But, since in SGD we use only one example at a time, we cannot implement the vectorized implementation on it. This can slow down the computations. To tackle this problem, a mixture of Batch Gradient Descent and SGD is used.

Neither we use all the dataset all at once nor we use the single example at a time. We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch. Doing this helps us achieve the advantages of both the former variants we saw. So, after creating the mini-batches of fixed size, we do the following steps in **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network

3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for the mini-batches we created

Just like SGD, the average cost over the epochs in mini-batch gradient descent fluctuates because we are averaging a small number of examples at a time.



Cost vs no of mini-batch (Source: https://stats.stackexchange.com/questions/310734/why-is-the-mini-batch-gradient-descents-cost-function-graph-noisy)

So, when we are using the mini-batch gradient descent we are updating our parameters frequently as well as we can use vectorized implementation for faster computations.

Conclusion

Just like every other thing in this world, all the three variants we saw have their advantages as well as disadvantages. It's not like the one variant is used frequently over all the others. Every variant is used uniformly depending on the situation and the context of the problem.

. . .

> *Got Questions? Need Help? Contact me!*

Github

Personal Website

LinkedIn

Email: sushantpatrikarml@gmail.com

Machine Learning | Neural Networks | Gradient Descent | Optimization | Data Science