Gradient Descent with Large Dataset

We already learned that one of the best way to get a high-performance machine learning system is to take a low bias learning algorithm and train that on lots of data. And sometimes the better performant algorithm is for those who has larger dataset rather than those who has the better algorithm. Larger dataset (like 100,000,000 in normal), the computational cost will be high to calculate the gradient descent. There are some more efficient ways to deal with these problems. Why cannot we choose smaller subset and do the training? Because in some cases if we plot the learning curve for a range of values of m and we get high variance when m is small.

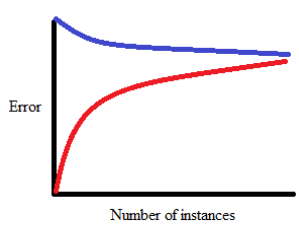


Image #1 (high bias)

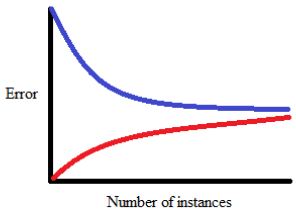


Image #2 (ideal)

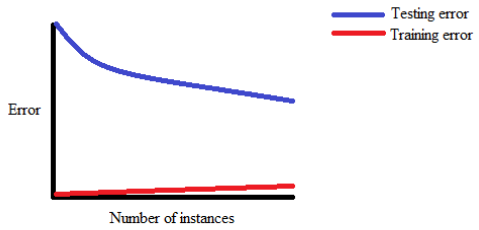


Image #3 (high variance)

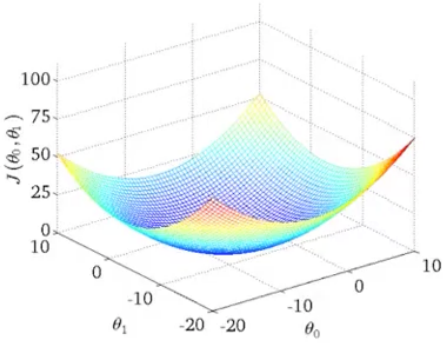
So, if we have a high bias system, adding more training set would not help, and we need to add more features to get a high variance system. Then adding new training dataset will help improving the model.

Stochastic Gradient Descent

When we have large dataset, the gradient descent becomes more computationally expensive method. Suppose, we are training a linear regression model using gradient descent. We remember the hypothesis and the cost function as follow for it:

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$



Using a gradient descent, we update values, using equation below:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$$

Gradient Descent with Large Dataset

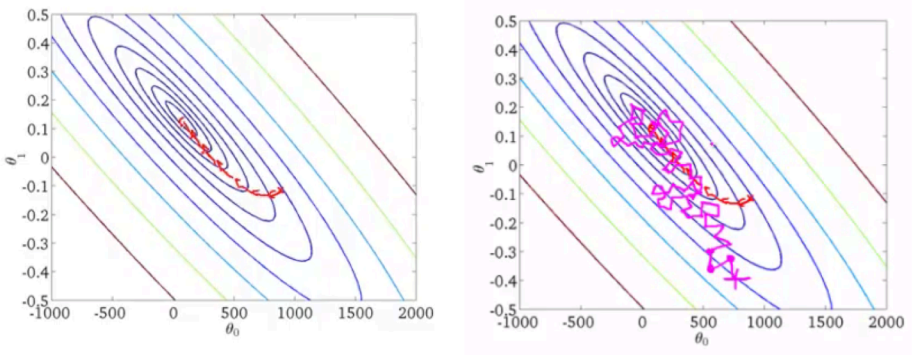
It gives us the global minimum, since the cost function is bell shape. For large n calculating the summation in gradient descent is computationally expensive. We called this type as *batch gradient descent, since we are looking at all training set at a time.* We need to read all the record into computers at computer and then for the next round, do that again and again.

More efficient algorithm is Stochastic algorithm, which only needs to look at single example at a time. *The cost function is defined how well the hypothesis is doing on a single example i.*

Batch Gradient Descent	Stochastic Gradient Descent
$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$ $\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i$ • Repeat θ_j for j=0..., n	$cost\left(\theta,(x^i,y^i)\right) = \frac{1}{2}(h_\theta(x^i) - y^i)^2$ $J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost\left(\theta,(x^i,y^i)\right)$ • Randomly shuffle the dataset • Repeat for i=1..., m and update: $\theta_j := \theta_j - \alpha\frac{1}{m}(h_\theta(x^i) - y^i)x_j^i$ Which has the partial derivate as we had before.

So, in stochastic gradient descent, it does the process for i=1, then for i=2, and so on. The random shuffling ensures that the data is in random order and we do not have any bias movement. In this method, instead of scanning all dataset (like 300,000,00) dataset, we already make a process as we enter every single training set.

Batch gradient descent, takes a reasonably straight-line trajectory to get to global minimum as shown on the left. In contrast in stochastic gradient descent, every iteration is going to be much faster since we are not doing sum all over the training examples. In this method, it's trying to fit better by every single training dataset entering.



Gradient Descent with Large Dataset

In this method, not always we are heading to global minimum and it might go a bit far, then get closer. Therefore, we have some random move toward the global minimum. *Finally, it gets loop around a point near global minimum and never gets exactly in minimum points.* Which is not problematic, as far as its close enough, that's good for the purpose. Usually we take around *1 to 10 times* the inner loops below depends on size of our training dataset:

{Repeat for i=1..., m and update:

$$\theta_j := \theta_j - \alpha \frac{1}{m}(h_\theta(x^i) - y^i)x_j^i\}$$

But in batch gradient descent, after considering the whole dataset, we have to go through the loops that takes forever!

Mini-Batch Gradient Descent

So far, we learned two types of gradient descent, the mini-batch is defined between those two:

- Batch gradient descent: use all m examples in each iteration
- Stochastic gradient descent: use 1 example in each iteration
- Mini-batch gradient descent: use b examples in each iteration (b is the minj-batch size)

The typical values for b is between 2 to 100. For example, if b=10 then we will have $(x^{(i)}, y^{(i)})$, …, $(x^{(i)+9}, y^{(i)+9})$ and then we update the gradient descent using these 10 examples.

$\theta_j := \theta_j - \alpha \frac{1}{10}\sum_{k=i}^{i+9}(h_\theta(x^k) - y^k)x_j^k$ repeat it for i:=i+10

We improve the progress in modifying the problem every b examples. *Mini-batch allow us to have vectorize implementation and partially parallelize the computations*. If we find a good vector implementation, it can run faster than stochastic gradient descent.

Stochastic Gradient Descent Convergence

How can we tune the learning parameter for stochastic gradient descent? When we were using the batch gradient descent, we plot J as a function of number of iterations of gradient descent to make sure it's converging and making sure it's decreasing on each iteration. In stochastic gradient descent, here is how we proceed to make sure algorithm is working properly:

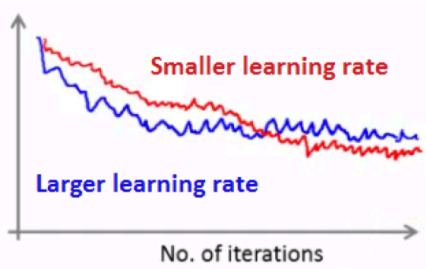
$$cost\left(\theta, (x^i, y^i)\right) = \frac{1}{2}(h_\theta(x^i) - y^i)^2$$

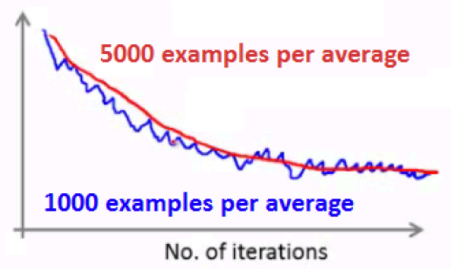
During learning, compute $cost\left(\theta, (x^i, y^i)\right)$ before updating θ using (x^i, y^i)

For convergence check: for every like 1000 iterations, plot $cost\left(\theta, (x^i, y^i)\right)$ averaged over the last 1000 examples proceed by example.

Gradient Descent with Large Dataset

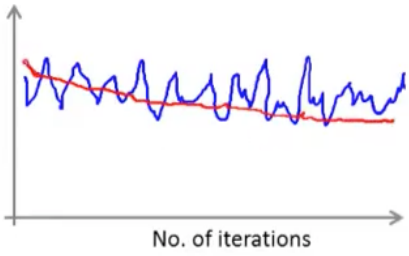
Here are some of the plot's you may encounter during the averaging. Using smaller learning rate cause the plot decreases slowly, and then maybe end up with slightly better solution, because smaller learning rate makes small oscillation. Sometimes, this can be negligible and sometimes can give you better solutions.



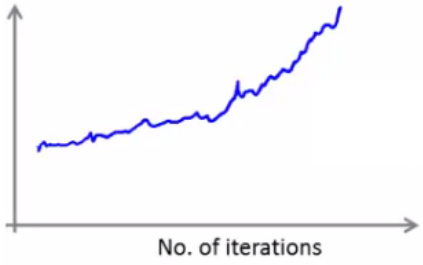


Increasing number averaged examples, can cause smoother plot. The plot above shows when taking average over 5,000 examples gives smoother line than averaging over 1,000 datasets. Disadvantage is, it's slower update on plot.

Sometimes, you may see that algorithm is not learning much and looks flat but after drawing a trend line, you can see the cost is decreasing. If the trend line is flat, it mean's algorithm is not learning and need to change features or leaning rate or etc.



If the cost function is increasing (diverging), it means we need to use smaller learning rate.



In most of gradient descent learning rate is held constant. We can slowly decrease α over time if we want θ to converge. The typical way of doing it is:

$\alpha = \frac{const1}{iterationnumber+sont2}$. But sometimes, people do not want to mess with it since it makes extra parameter to play with. If you decide to do it, in stochastic gradient descent, we get the global minimum instead of loop around it's region.