1 Large scale machine learning

1.1 Problem with large scale data

Take linear regression as an example. When we use gradient descent to learn the parameters, what happens to the parameters in each iteration is

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)},\ j = 0, \ldots, n, \tag{1}$$

in which m is the number of examples in the training set. If the scale of the training set is very large, say $m = 10^9$, we will have to do 10^9 additions in each single step of gradient descent, which is a problem that needs to be addressed, otherwise the large amount of computation will make it impossible to solve the problem in reasonable time. One way to avoid such problem is to check if the problem could be solved with a small training set, say $m = 1000$. If the learing curve shows that the problem does not have high variance (overfit) with the small training set, it might be unnecessary to use the large traing set.

1.2 Stochastic gradient descent

The traditional gradient descent (1) is called **batch gradient descent** because it uses all examples at each iteration. Batch gradient descent ensures that the parameters get closer to the optima at each iteration.

In **stochastic gradient descent**, the examples are first shuffled. Then we will repeat

$$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)},\ j = 0, \ldots, n \tag{2}$$

for $i = 1, \ldots, m$. Rather than altering the value of θ_j after summing up items related to all examples as in batch gradient descent, θ_j is altered a little bit for each example. It is not guaranteed that θ_j gets closer to its optimum after each alteration. Actually θ_j ends up wandering around the optimum continually in a small region. In practice, this is enough to ensure the convergence of the algorithm because the parameters are satisfactory so long as they are close enough to the optima. And in general stochastic gradient descent converges much faster than batch gradient descent. Typically all examples need to be passed 1 to 10 times to obtain a satisfactory result.

1.3 Mini batch gradient descent

Batch gradient descent uses all m examples in each iteration, while in stochastic gradient descent, only 1 example is used in each iteration. What mini batch graident descent does it somewhere in between: b examples are used in each iteration:

$$\theta_j := \theta_j - \alpha\frac{1}{b}\sum_{k=i}^{i+b-1}(h_\theta(x^{(k)}) - y^{(k)})x_j^{(k)} \tag{3}$$

b is called the mini batch size. Typically b could be from 2 to 100.

With a judicious choice of b, mini batch gradient descent could run even faster than stochastic gradient descent.

1.4 Convergence

With batch gradient descent, we check the convergence of the algorithm by calculating $J_{train}(\theta)$ after each iteration. This does not make sense for stochastic gradient descent or mini batch gradient descent because the calculation of $J_{train}(\theta)$ leads to the computation load that we are trying to get rid of due to training set size.

What we can do is to calculate $cost(\theta, (x^{(i)}, y^{(i)}) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$ before updating θ using $(x^{(i)}, y^{(i)})$ at each iteration. Then every 1000 iterations (just an example), we can plot $cost(\theta, (x^{(i)}, y^{(i)})$ averaged over the last 1000 examples processed by the algorithm to see the trend of the cost function, and adjust value of α accordingly.

Learning rate α is usually kept constant. We could choose to use a learning rate that gradually diminishes over time, say $\alpha = \frac{const1}{const2 + iterationNumber}$ to make θ_j get closer to the optima in the end, but this will introduce two more paramters for us to tune, and is not always a good idea.

1.5 Online learning

When a continuous stream of data is available, e.g. user searching action on an online shopping website, we can use the mechanism of online learning. Rather than collecting a lot of examples as a traing set and use machine learning algorithms to learning the best parameters, we can adjust the parameters a little bit immediately after obtaining a new example, like we did in stochastic gradient descent, and discard the example without having to store it for further use. Besides facilitating treatment of continuous stream of data, online learning also makes it possible to adapt our model according to changing user preferences.

1.6 Map/reduce and parallelism

The map/reduce approach to large scale machine learning takes advantage of distributed computation. Computation involving addtion of all examples can be divided into a few parts and be sent to a series of machines to conduct the computation, and then the results on all machines are brought together to obtain the final result. Even if we do not have a series of machines available, it is still possible to benefit from the acceleration of parallelism because most modern computers have a few cores that can conduct computation tasks independently.