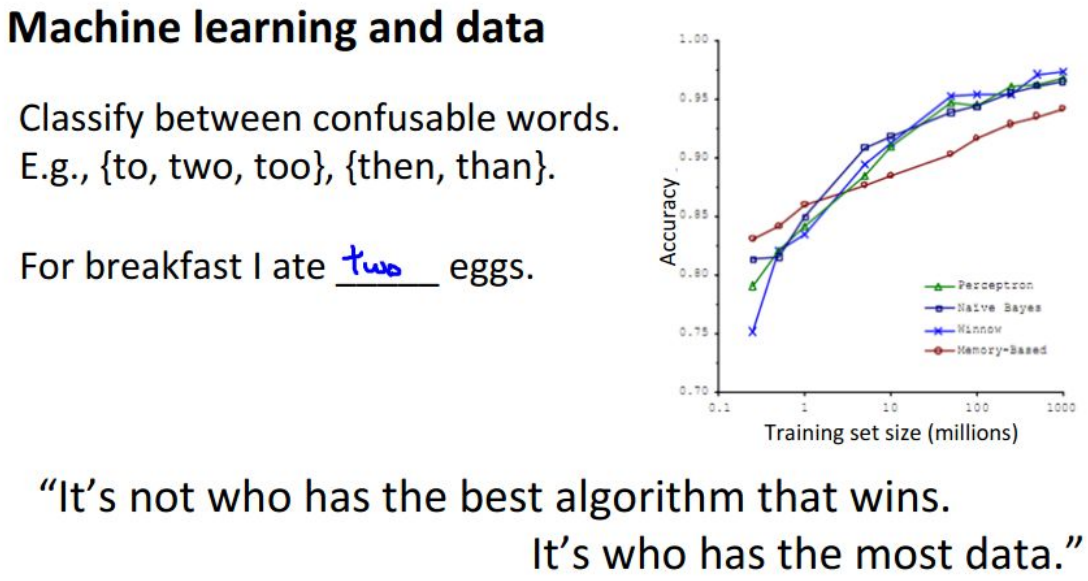
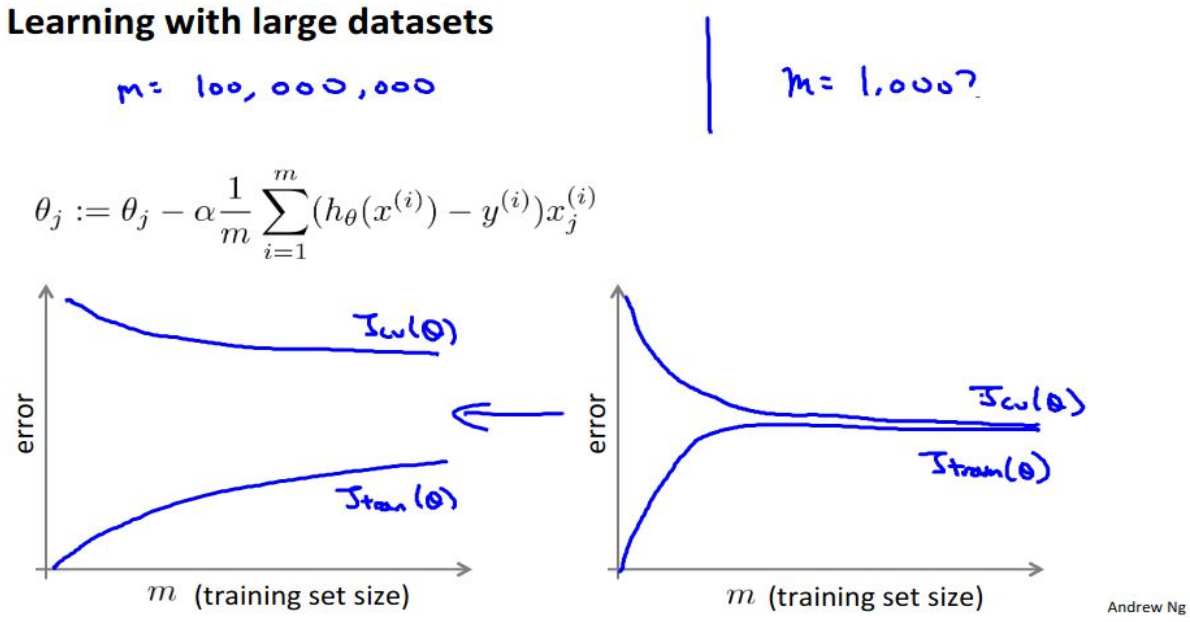
Week : 10

Learning with large datasets :

If you have a large size of data then it can help you even with a bad algorithm.



Sometimes you may have hundreds of millions of data, but if you want to use them all to train your algorithm, it will take a huge time. Instead of doing that we can decide how much bigger data we need to train our algorithm by the help of high vias - high variance graph shown in the picture below. If we ended up with a graph like the left one, that means our graph is in high variance and more training sets will help our data to improve result. But if our graph will be like the right one, then it is suffering from high vias. Adding more training sets won't help to improve our algorithm. We can try adding new features to solve high bias.



Stochastic gradient descent :

Original gradient descent where we count all training sets together, if we had a large dataset then it would be so much slower. As we count all training sets together, another name of this gradient descent is batch gradient descent. In order to solve the slowness we have another gradient descent, stochastic gradient descent. It counts a single dataset at a time rather than the whole dataset. As a result the algorithm is much faster than our original one.

Linear regression with gradient descent

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

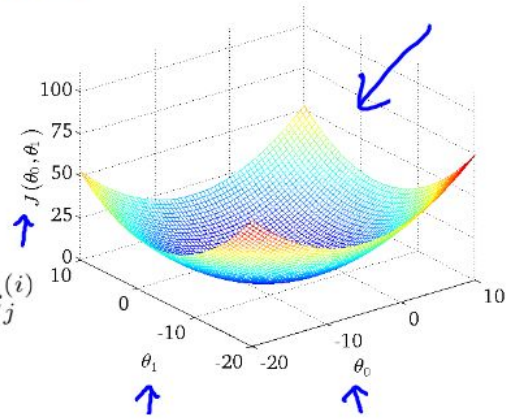
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

(for every $j = 0, \ldots, n$)

}



Linear regression with gradient descent

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

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

Repeat {

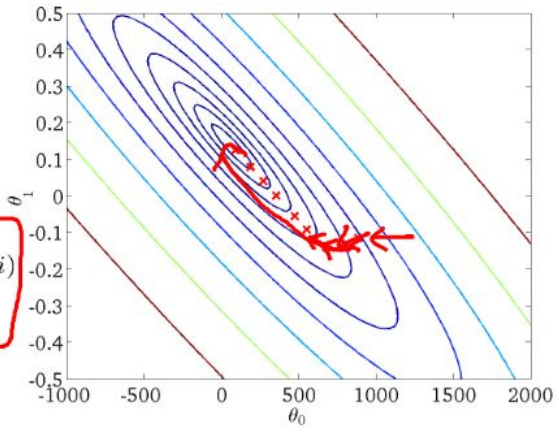
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

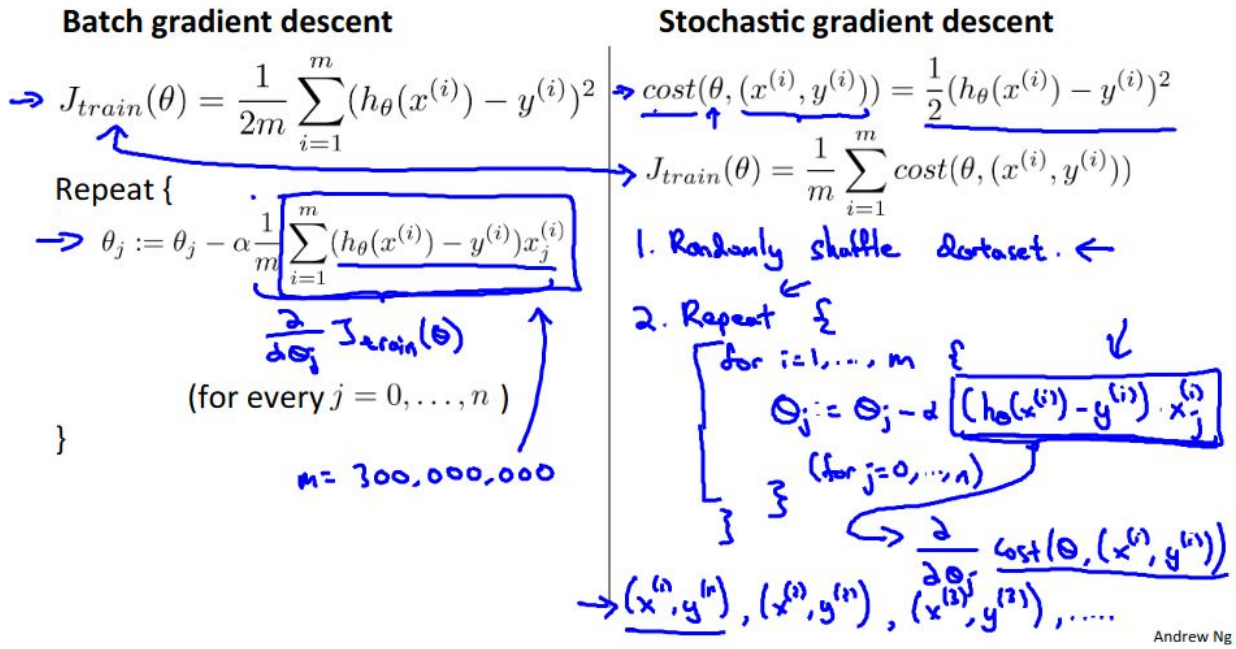
(for every $j = 0, \ldots, n$)

}

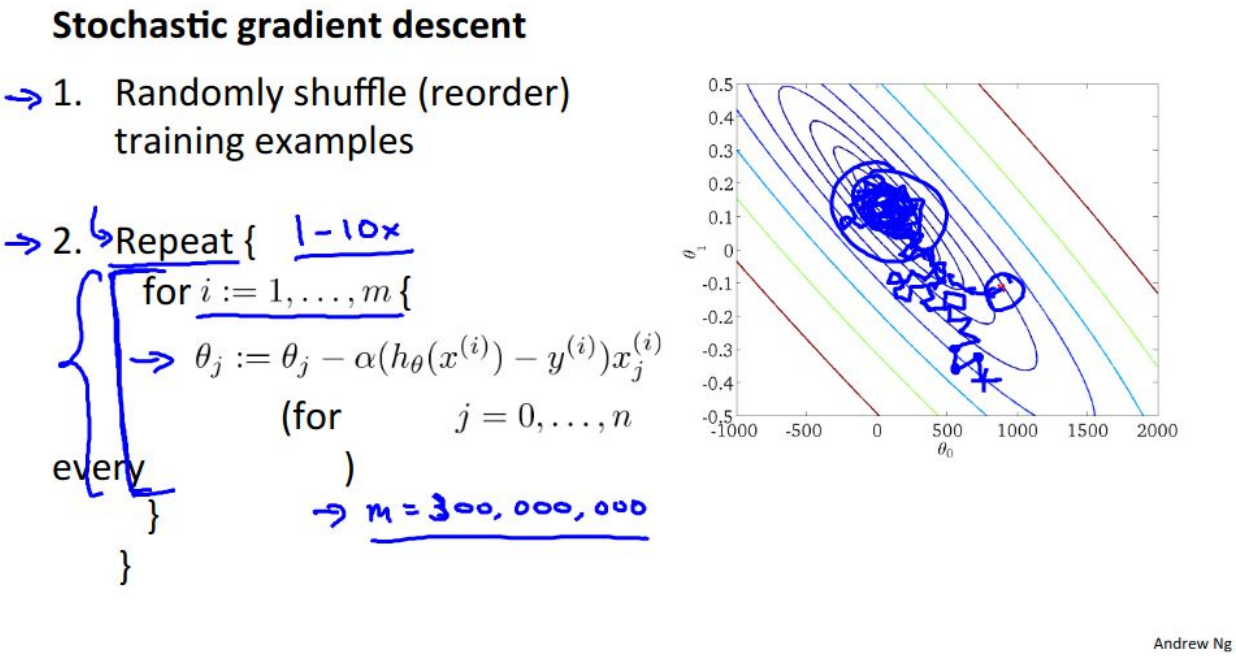
$M = 300,000,000$

Batch gradient descent



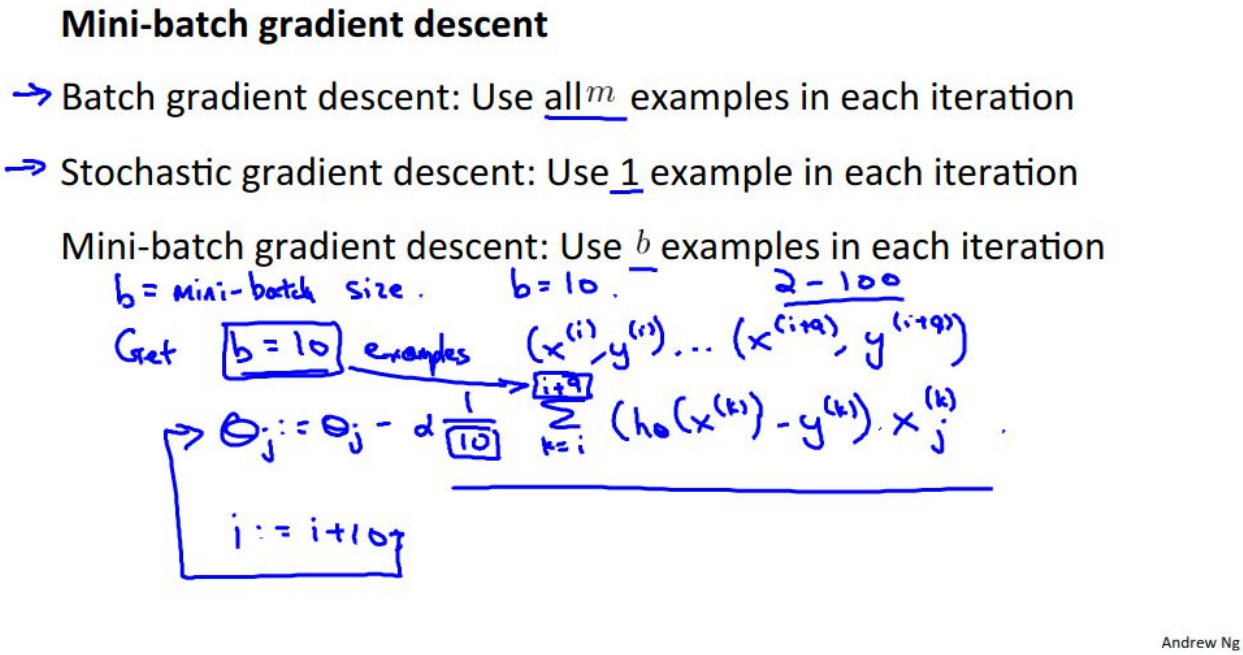


As we use one single data set at a time, our gradient descent will move in a zigzag mood, and it is normal. But it is faster than the convention one and also can achieve our goal.

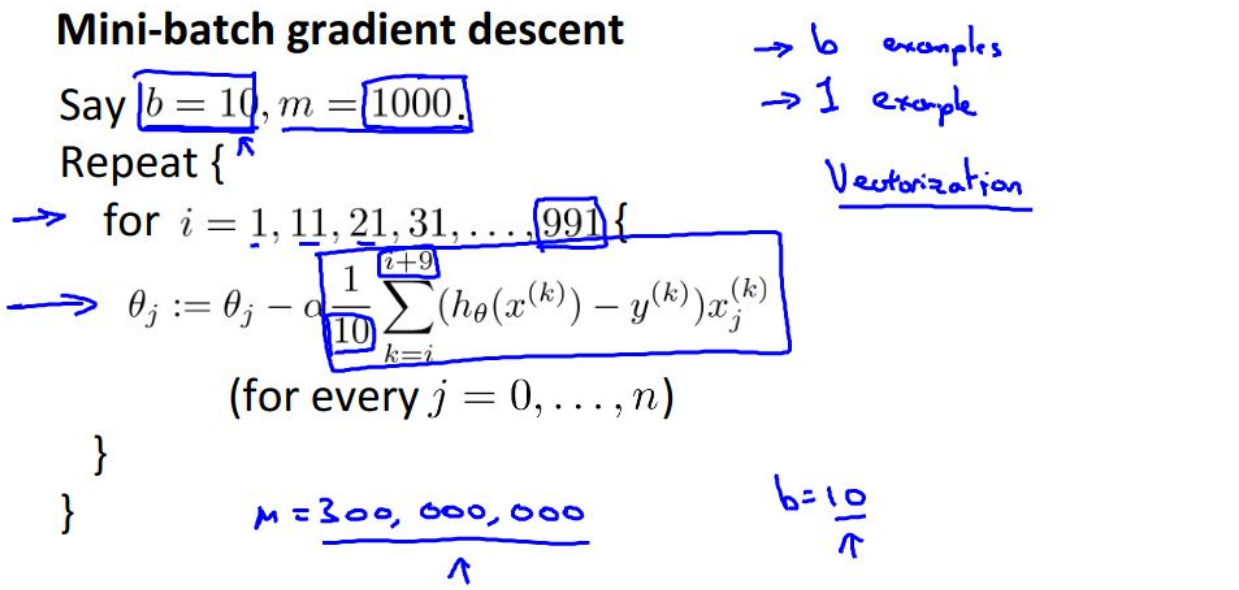


Mini batch gradient descent :

Besides stochastic gradient descent we have another gradient descent named mini batch gradient descent which is also faster than batch gradient descent. In this gradient descent we will take b number of test datasets at-a-time and iterate whole datasets by grouping in b number at a time.



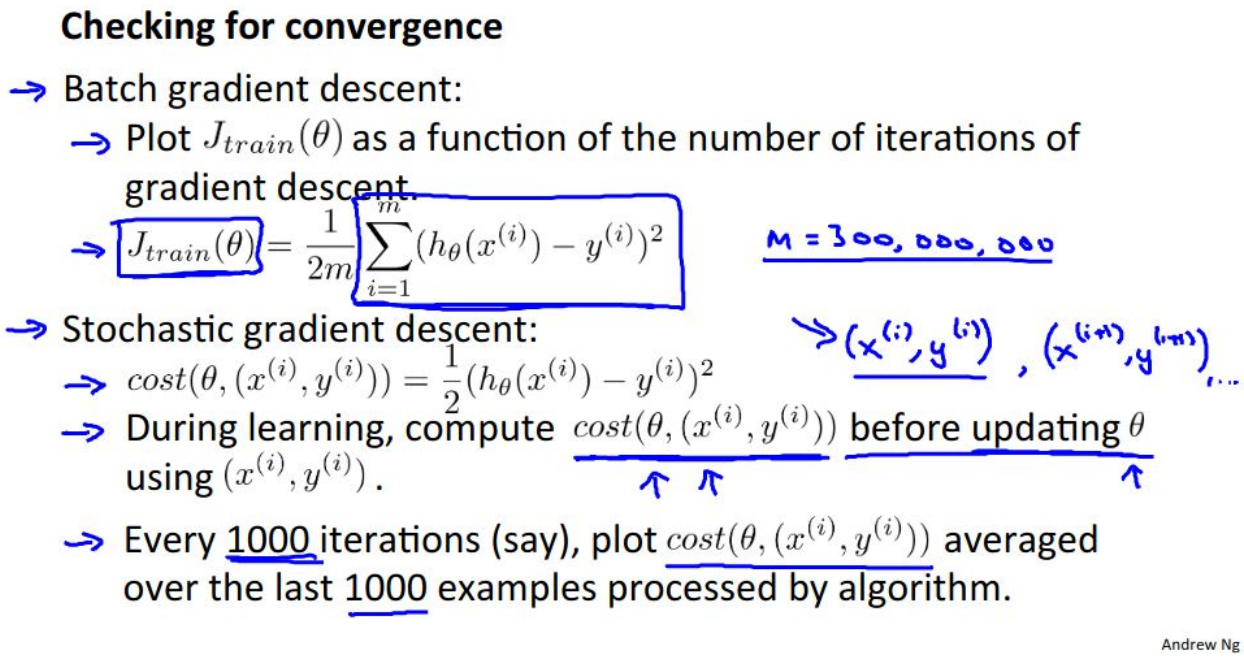
You may be surprised why we should have another gradient descent where stochastic is the fastest. Well, in that case if you can implement a good vectorized implementation then it will be faster than stochastic gradient descent.



Andrew Ng

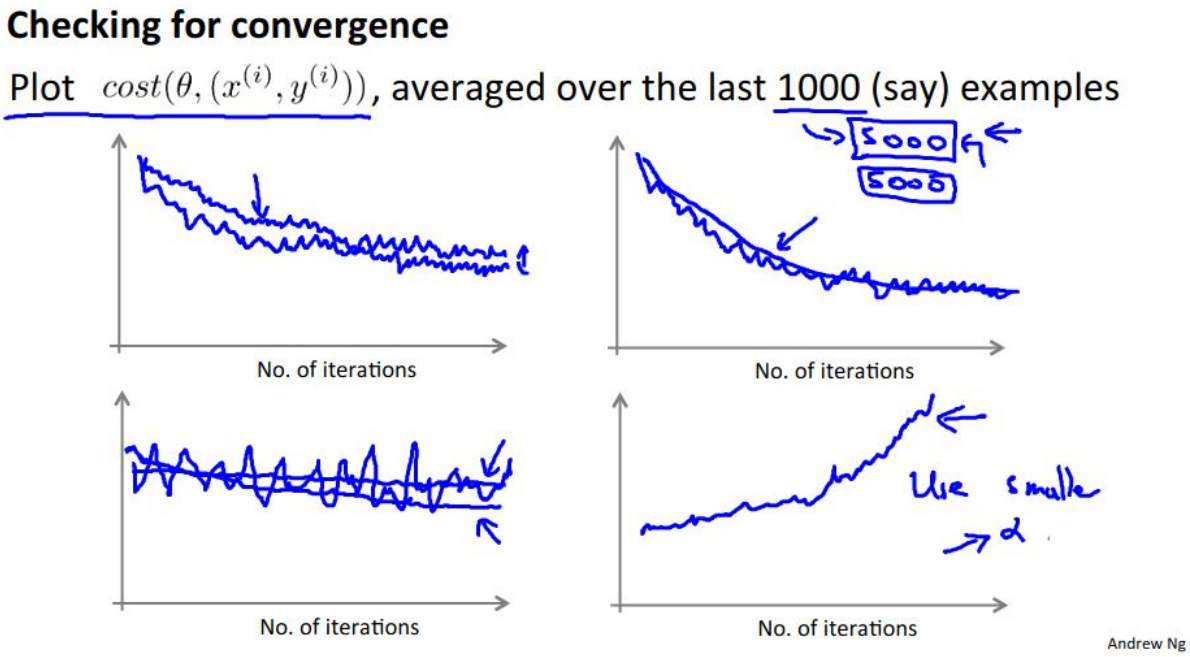
Stochastic gradient descent convergence :

It is important to check if our algorithm is learning well. To do so, we can plot a graph of convergence for every 1k iterations. The number of iterations may vary but the main motivation of this doing is to check if stochastic gradient descent is working well.



Andrew Ng

We may end up with such graphs. If you end up with top like twos, then we can assume that our algorithm is learning well. But if our grap ends up with bottom like twos, then there may be some problem in our algorithm. It may happen because of learning rate alpha. We can set alpha smaller and hope our algorithm learns well.



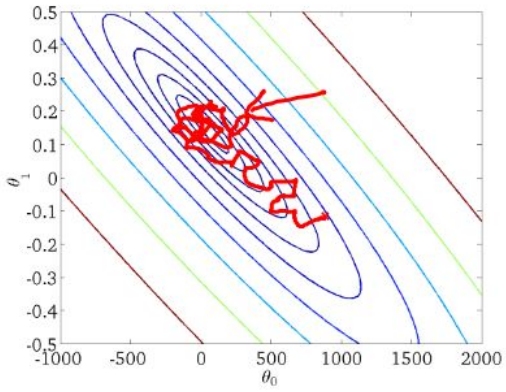
Setting such a small learning rate may make our algorithm slower. But if we set a large learning rate then it also misleads our algorithm. To solve both problems we can set a large learning rate at first of our algorithm and decrease it by every iteration. To do so, we have been given a formula in the picture below.

Stochastic gradient descent

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

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

1. Randomly shuffle dataset.
2. Repeat {
   for $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }



Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$)

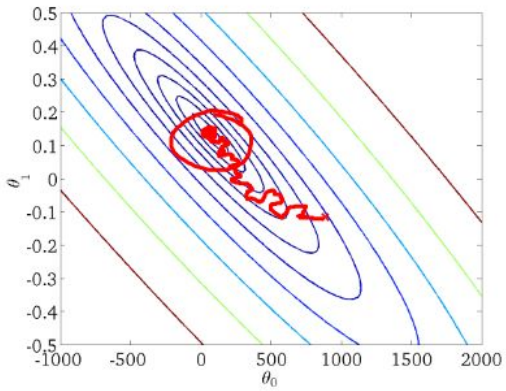
Andrew Ng

Stochastic gradient descent

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

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

1. Randomly shuffle dataset.
2. Repeat {
   for $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }

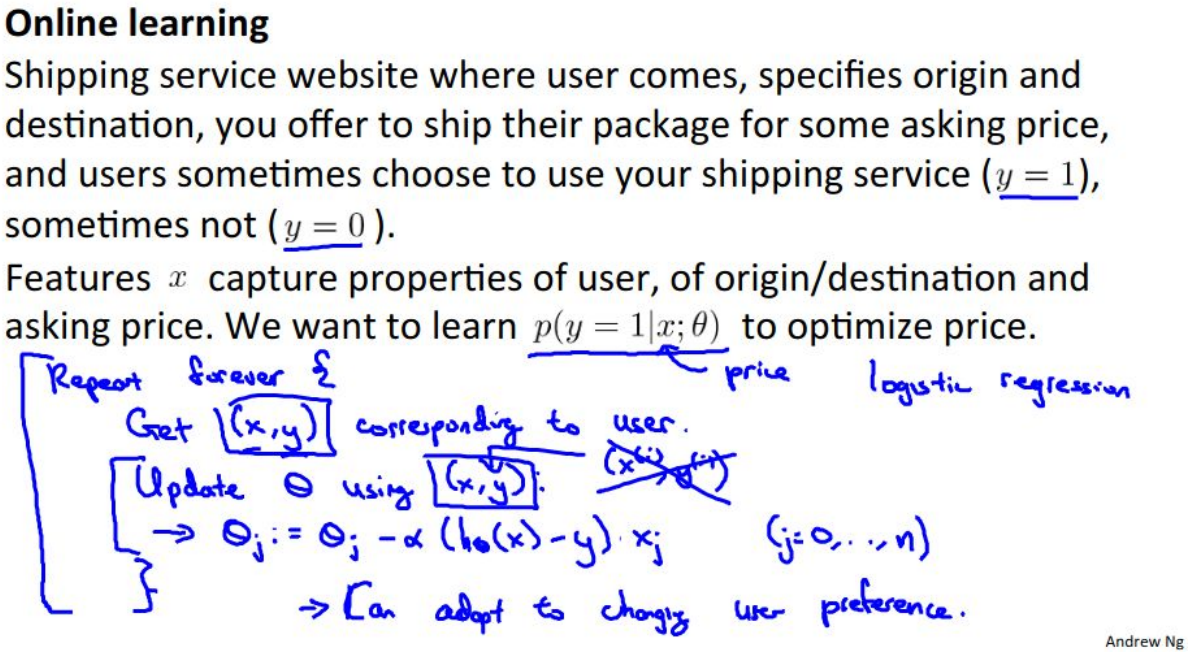


Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$)

Andrew Ng

Online learning :

Sometimes websites have a large number of users. As a result, the would have a huge number of training set. If they want to train their algorithm in a conventional way they may need a large number of storage as well as a huge time for learning. On the other hand, if they have an algorithm which can learn online and per-user click they can avoid both storage and time problems. In that technique, the online algorithm takes a training set from a user and feeds it to the algorithm and throws it away.



Suppose a user goes to a mobile phone website and searches mobile phones by writing “android phone 1080p camera”. And if the website wants to show 10 most click-likely mobile phones then it needs to pick 10 best mobile phones based on these keywords. And to do so, they can apply logistic regression and learn online.

Other online learning example:

Product search (learning to search)

User searches for "Android phone 1080p camera"

Have 100 phones in store. Will return 10 results.

→ $x =$ features of phone, how many words in user query match name of phone, how many words in query match description of phone, etc.

→ $y = 1$ if user clicks on link. $y = 0$ otherwise.

→ Learn $p(y = 1|x; \theta)$.

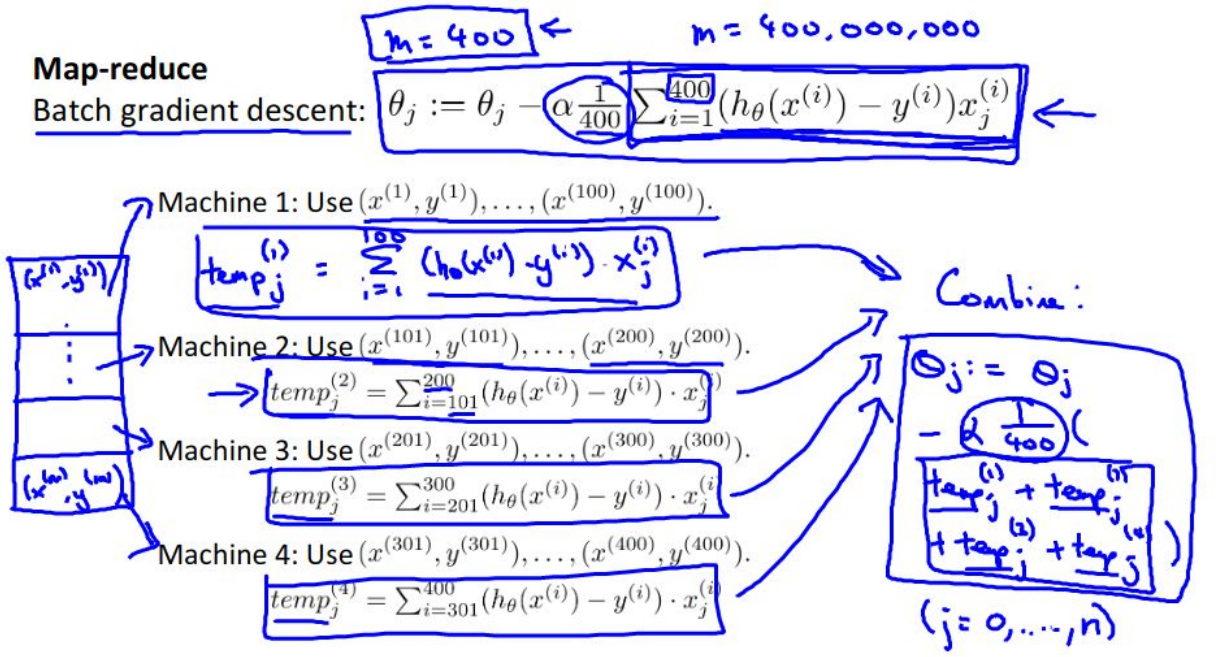
→ Use to show user the 10 phones they're most likely to click on.

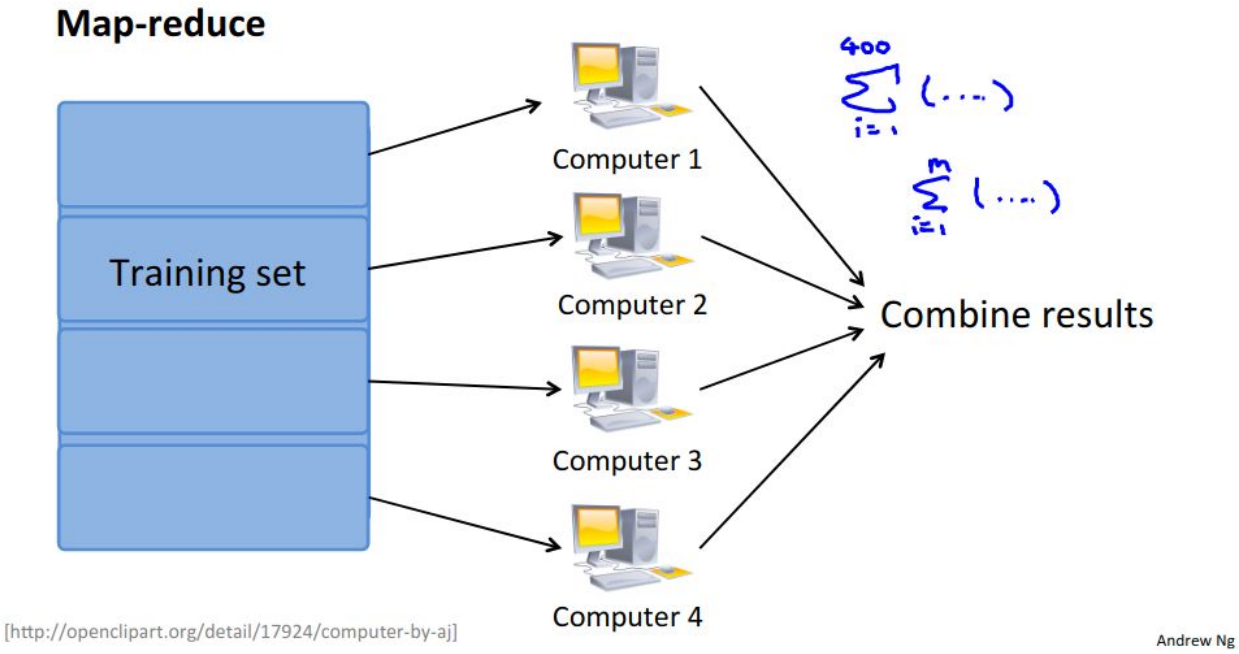
Other examples: Choosing special offers to show user; customized selection of news articles; product recommendation; …

Andrew Ng

Map reduce and data parallelism :

If we have a large number of data that no one machine we have can process all together, then we can use map reduce algorithm to use more than one machine to process data. In that case if we have p number of machines, we divide our data into p blocks and send each block to one machine. After computing these data all machines send back the result to the main machine, and the main machine will sum all of the work.





We can use map reduce algorithm on all algorithms which have summation over the training set.

Map-reduce and summation over the training set

Many learning algorithms can be expressed as computing sums of functions over the training set.

E.g. for advanced optimization, with logistic regression, need:

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

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

$temp^{(i)} = h_\theta(x^{(i)}) - y^{(i)}$

$temp_j^{(i)} = temp^{(i)} \cdot x_j^{(i)}$

Map-reduce and summation over the training set

Many learning algorithms can be expressed as computing sums of functions over the training set.

E.g. for advanced optimization, with logistic regression, need:

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

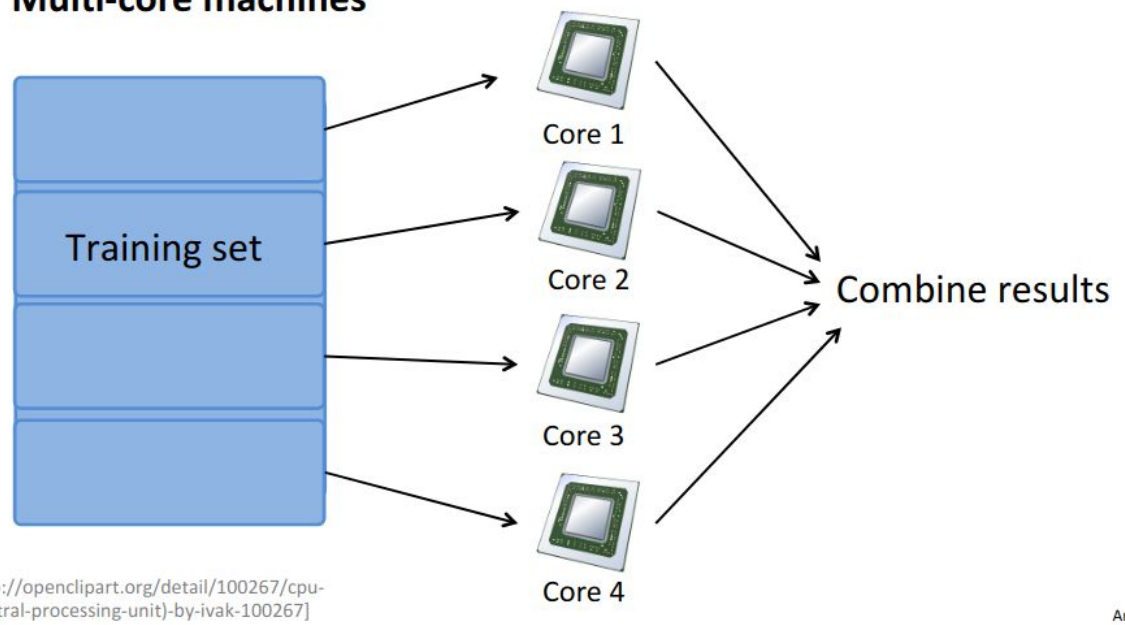
$$\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

$temp^{(i)} = h_\theta(x^{(i)}) - y^{(i)}$

$temp_j^{(i)} = temp^{(i)} \cdot x_j^{(i)}$

Andrew Ng

Multi-core machines



[http://openclipart.org/detail/100267/cpu-(central-processing-unit)-by-ivak-100267]

Andrew Ng