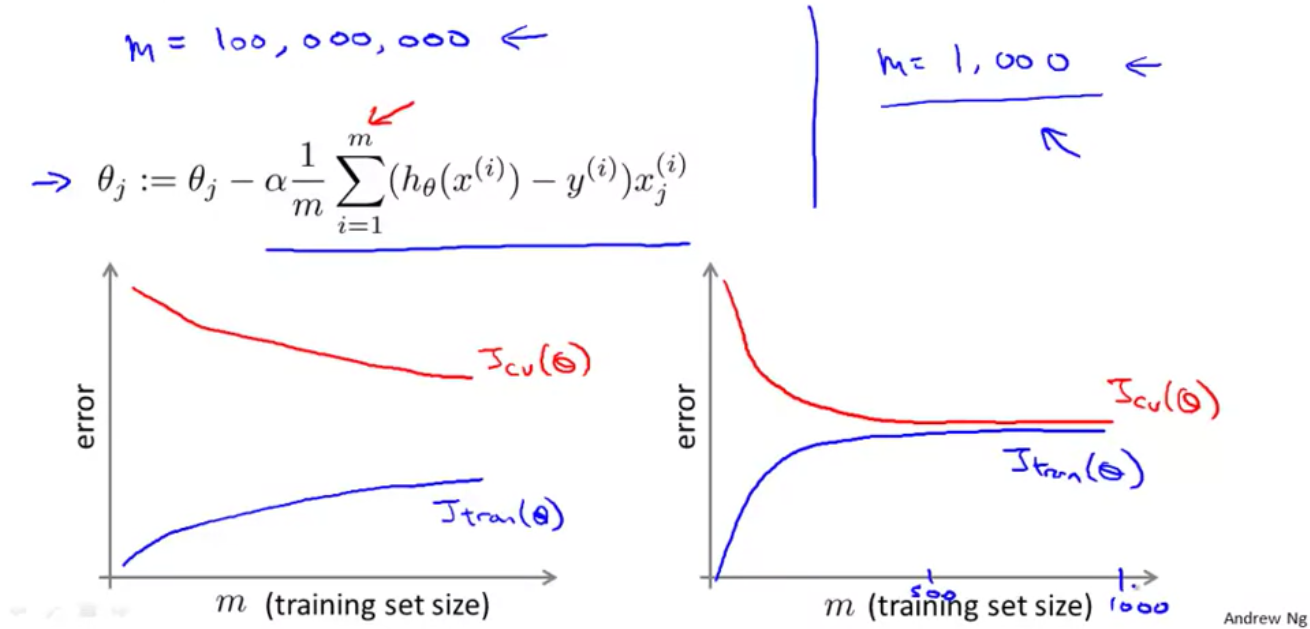
*Deciding when to Use all the Rows of a very large Dataset:*

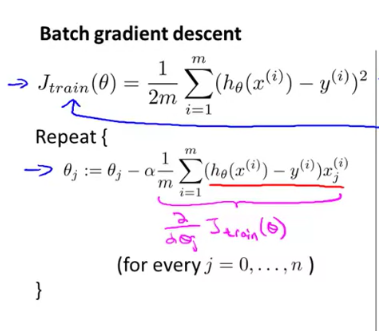


In the above picture we see that after plotting the learning curve, it becomes clear that whether using all the m datapoints will be helpful or not:

1. If we have a case of high variance or overfitting (as shown by the curve on the left), then using more training examples may help converge the gap between training error and cross-validation error.

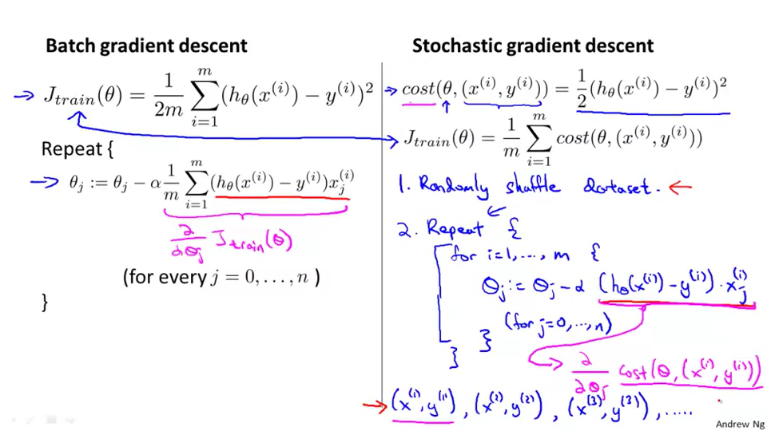
2. If we have a case of high bias or underfitting (as shown by the curve on the right), then using more training examples may not be helpful and we may work with say 1000 datapoints instead of all the m.

*Dealing With Large Datasets:*If we have a very large dataset say 300 million rows of datapoints, then for computing gradient descent several times and cost function a each time becomes computationally very expensive.





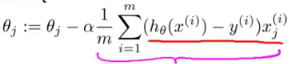
Computing the sum of for 1 to 100 million data points k times, can prove to be computationally very expensive. Hence we use stochastic gradient descent:



In stochastic gradient descent, we break down the summation into loops, i.e. when running gradient descent, instead of subtracting say 10 at once from theta, we first subtract 2 then 3 then 3 then 2 (2+3+3+2=10).

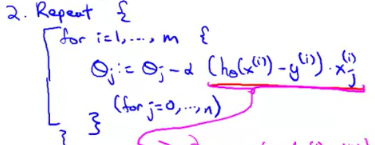
*So how does this relax computational expense?*

Consider that we do not use stochastic gradient descent. Now while performing gradient descent:



We have to perform the sum of 1 to 100 million entries and then subtract the value obtained from theta and repeat the process say k times. Now say we have added 10000 entries and we still need to add a lot. Say the sum of these 10000 entries is x. Now, adding so many more terms to x will give a very large number. Say for the 200 millionth term to 201 millionth we have to add 1003440 to 1524300, it does become heavy for a computer to does such a process another 100 million times till 300 million.

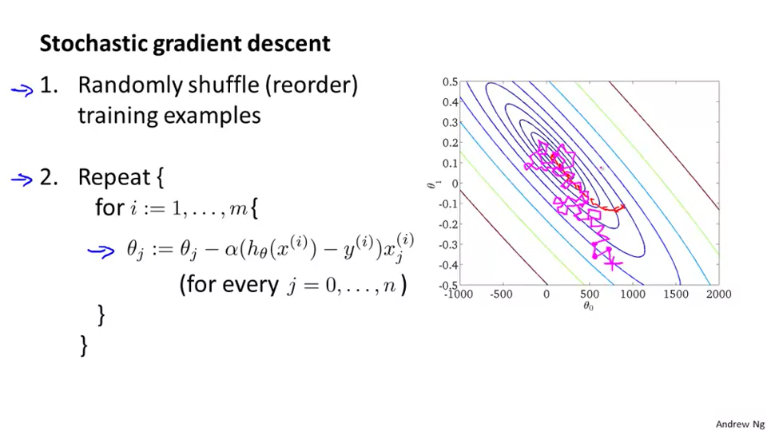
Now consider that we use stochastic gradient descent. Now while performing gradient descent:



As we subtract the derivate term each time from theta, there is not heavy load on the computer. The computer just has to do small calculations repeatedly, while in batch gradient descent, the computer has to do heavy calculations repeatedly.

Though the number of iterations increase in stochastic gradient descent as compared to batch gradient descent, performing light calculations again and again is more computationally feasible than performing heavy calculations a small number of times.

*Comparing Batch Gradient Descent and Stochastic Gradient Descent Graphically:*

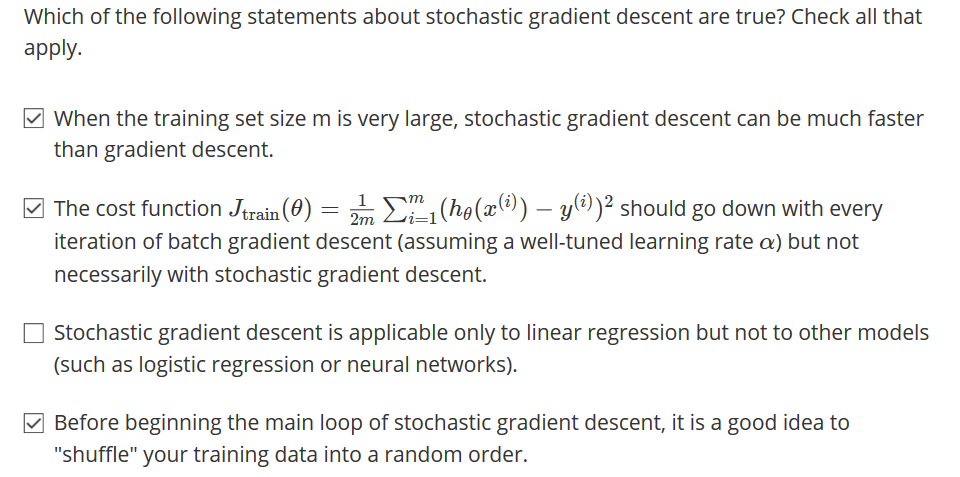


The descent of Stochastic is shown in pink while of batch is shown in red. We see that Stochastic gradient descent can occasionally go in the wrong direction, which can increase the value of the cost function. Yet eventually it ends up in the approximately right region of minima. Once it reaches the region where the global/local minimum lies, instead of converging to the local/global minima, it oscillates around the point. Hence though the gradient descent may not end up at the exact minima, it does end up close to the actual it.

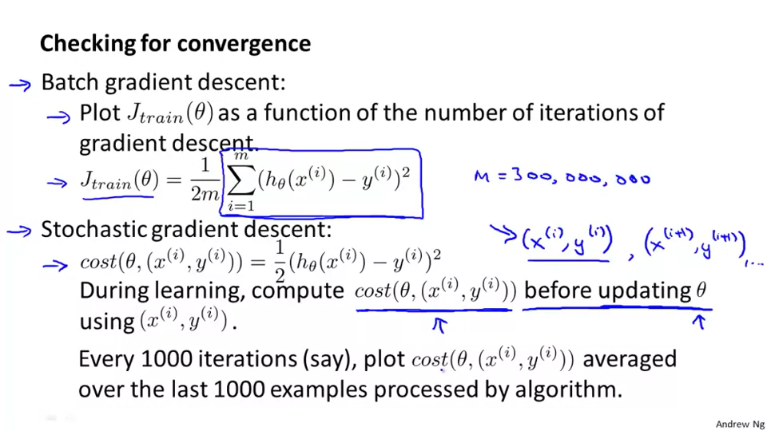
*Note:* Usually running the algorithm once for all the m examples is enough , but for better results the whole process can be repeated a small number of times say between 1 to 10 times.

After the computing the sum from 1 to 100 million and then subtracting this value from theta, batch gradient descent moves 1 step ahead, hence it is slower than stochastic gradient descent.

*Question:*



*Evaluating Gradient Descent for Stochastic Gradient Descent:*

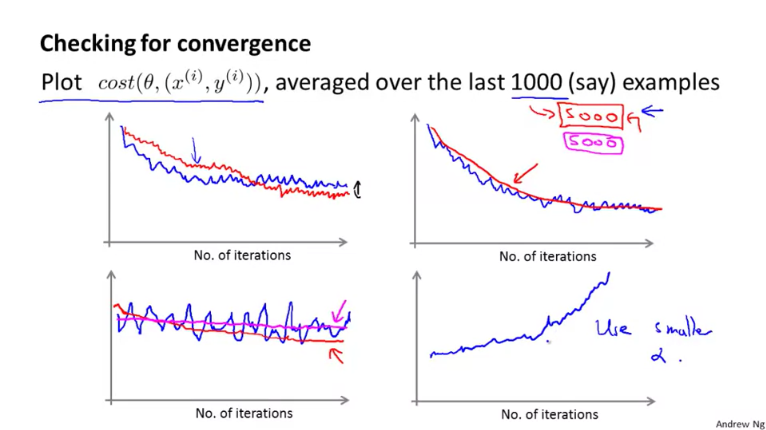


We run our algorithm m\*r times where m is the number of data points from ranging from 1 to m and r is the number of times we are repeating the whole process. Now for every 1000 examples out of m, we can average the cost function and plot it on the graph cost vs number of iterations. This means for the 1st 1000 examples we compute the average cost function that is:

(cost for x1, y1 +cost for x2,y2 + ….. + cost for x1000,y1000 ) /1000

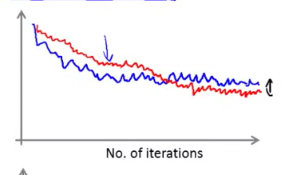
Then for the next 1000 iterations that is x1001, y1001…..x2000, y2000 we do the same. We then plot average cost function value vs number of iterations.

*Checking for convergence:*



*Now we will examine each of the above graphs one by one.*

*1.*

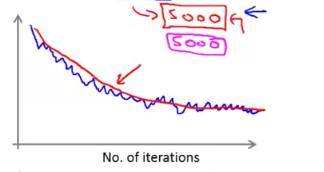


Here we have a plot of cost vs number of iterations where the cost function has been averaged for every 1000 iterations and iterations increase in sums of 1000 i.e. iteration 1 = 1000, iteration 2=2000 and so on.

As cost can also increase in stochastic gradient descent with increase in number of iterations, we see that the curve is wavy that is, at one point it is increasing and at another it is decreasing. Yet the overall effect is that the cost function is decreasing with increase in number of iterations.

The red line is the same descent but with a smaller value of learning rate alpha. Due to a small value of learning rate , the algorithm oscillates very slowly around the region of global/local minima and hence lands on a value closer to the minima than an algorithm with large learning rate.

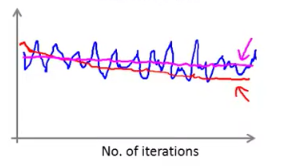
*2.*



Here we have a plot of cost vs number of iterations where the blue line is the same as the blue line there in the previous graph. The red line shows a cost function that has been averaged for every 5000 iterations and iterations increase in sums of 5000 i.e. iteration 1 = 5000, iteration 2=10000 and so on.

We see that the red line is much smoother than the blue one. This is because when we take 5000 data points, the individual variations of the 5 data points of size 1000 each are suppressed.

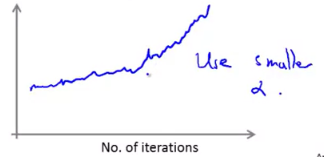
*3.*



If we have a line that looks like the blue line shown in the above graph, we cannot infer that whether the cost function is decreasing or not. To solve this problem, we can increase the number of sets from 1000 to 5000. The line in the red shows the data averaged over 5000 data points.

If even after increasing the number of data points, we get a line same as the pink line, we can infer that the cost function is not decreasing with increase in the number of iterations.

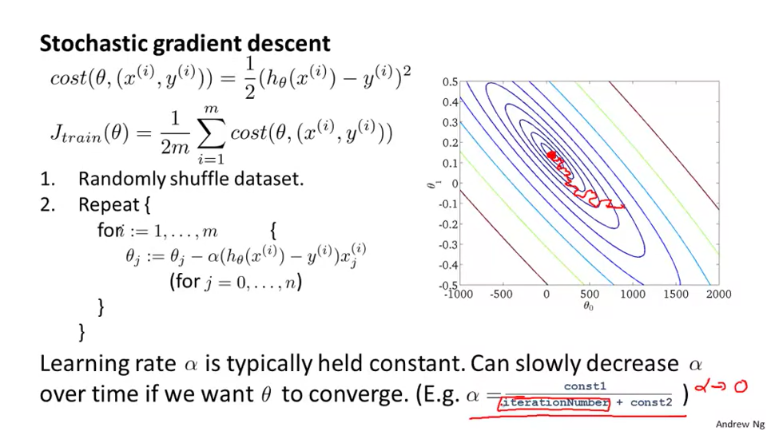
*4.*



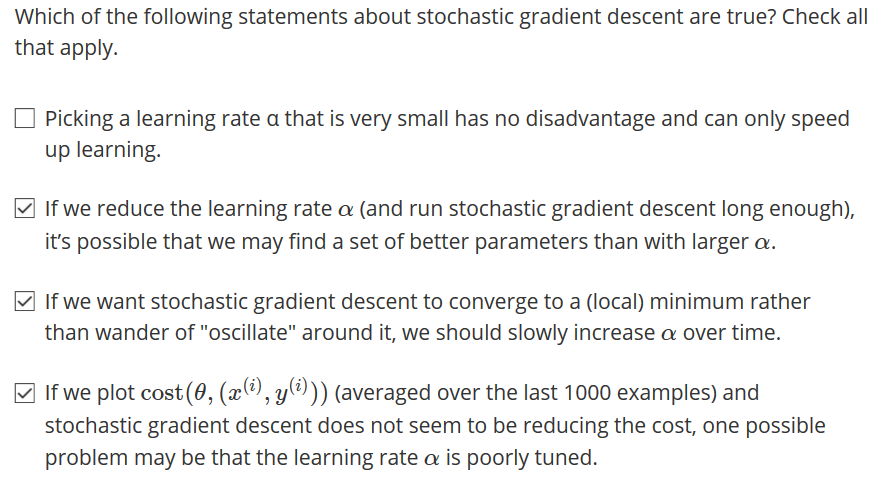
If we have a line on the graph same as the blue one, then we can attribute it to a large value of alpha and can solve this problem by decreasing the value of alpha.

*Decreasing Alpha with each iteration for Convergence at the Local/Global optima:*

If we slowly decrease the value alpha with increasing number of iterations, the algorithm can converge to a global minimum. This is because due to a small value of learning rate , the algorithm oscillates very slowly around the region of global/local minima and hence lands on a value closer to the minima than an algorithm with large learning rate.



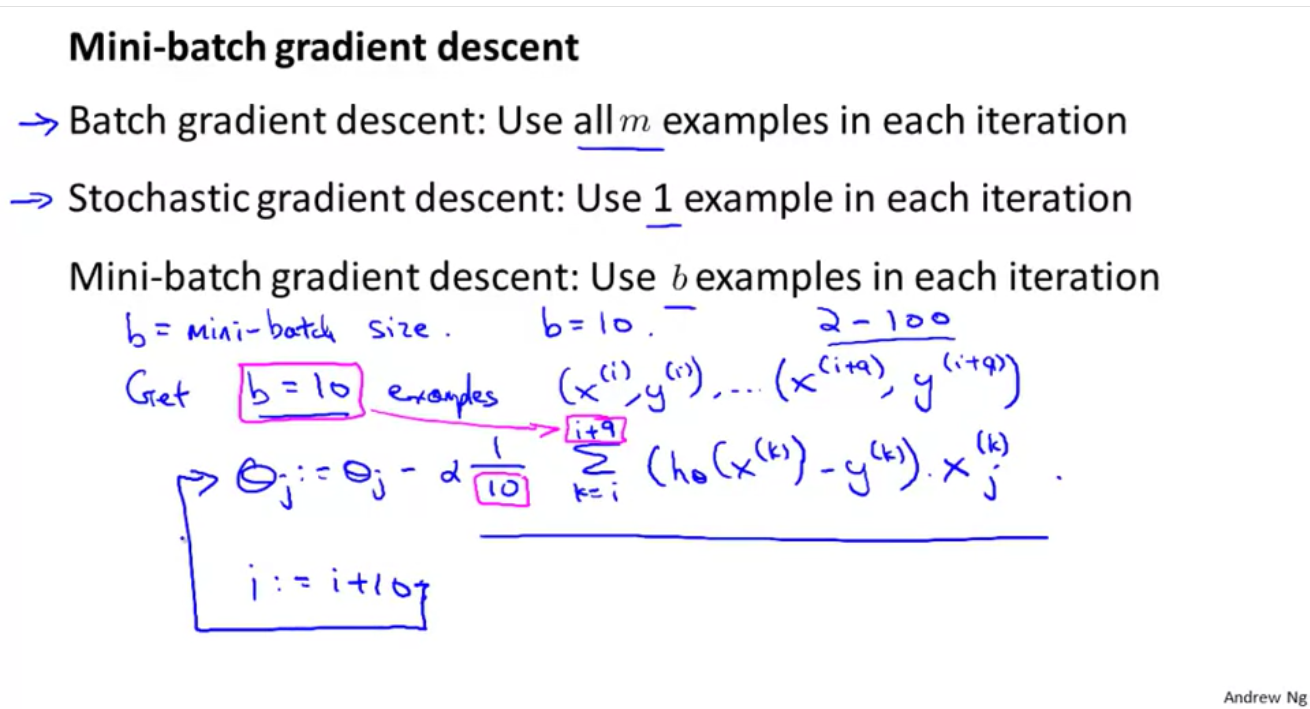
*Question:*



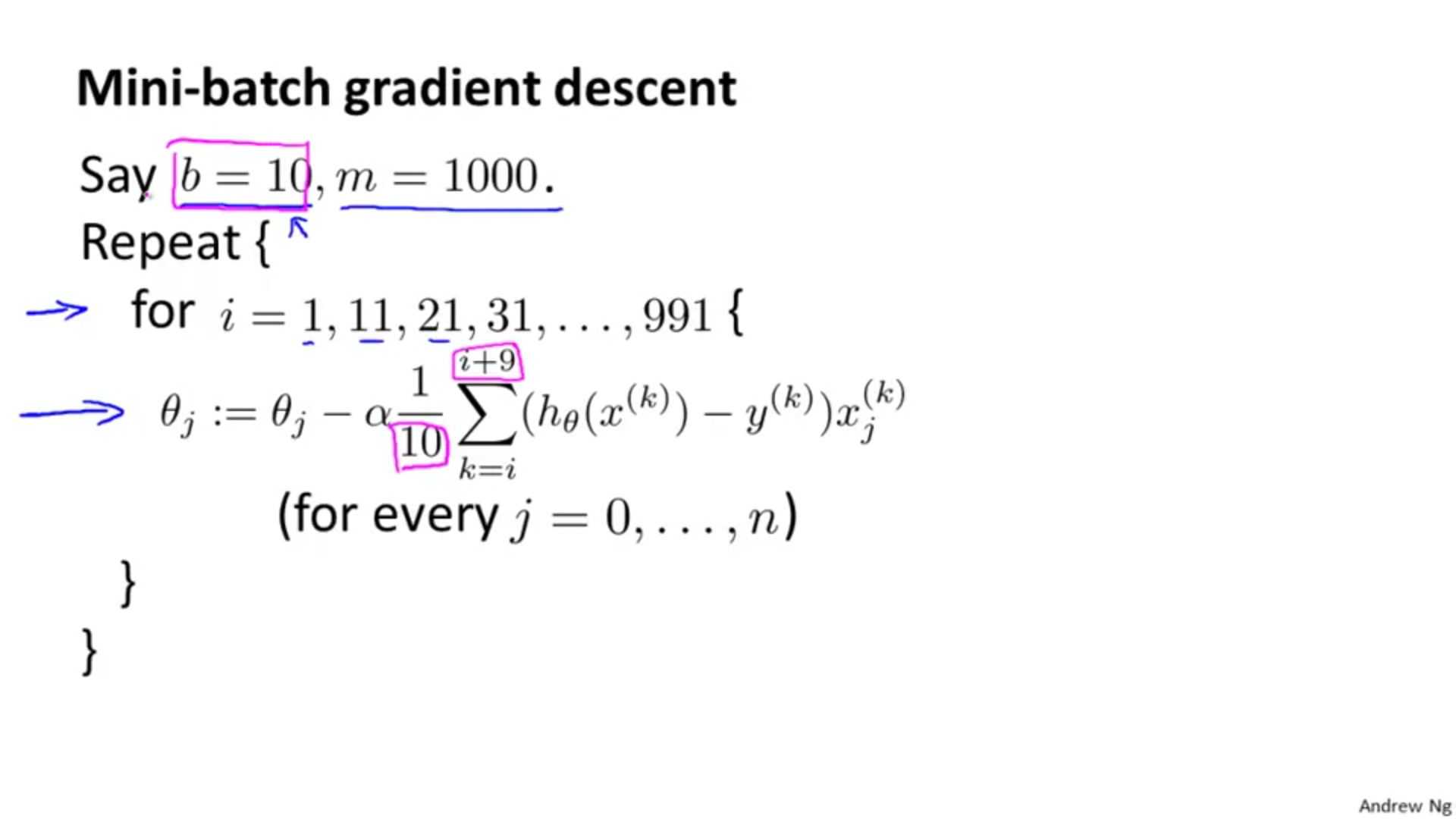


*Mini Batch Gradient Descent:*

Mini batch gradient descent is in the middle of stochastic and batch gradient descent:



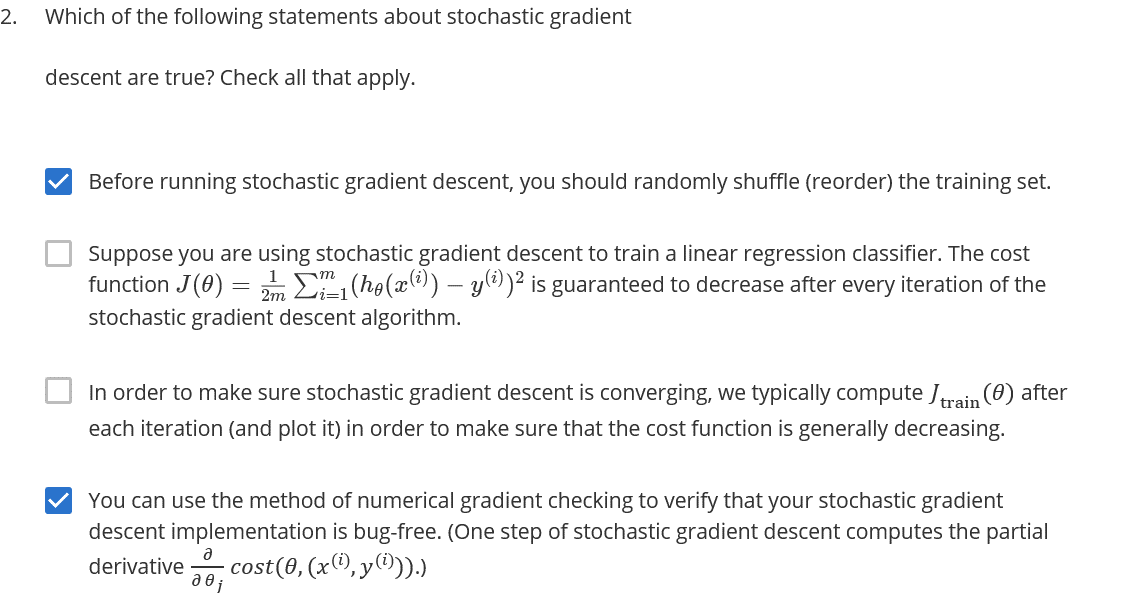
*Example:*

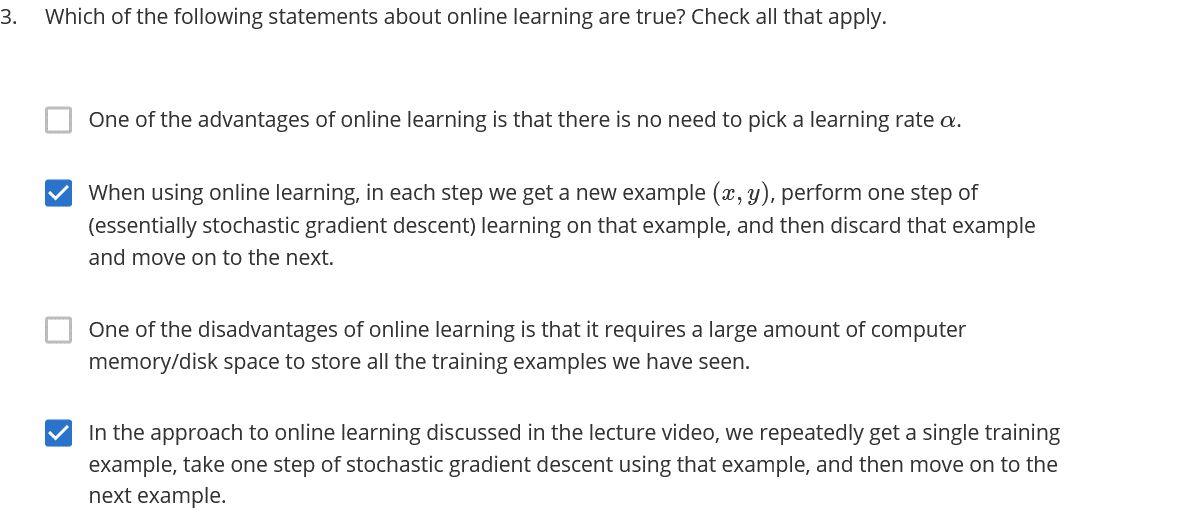


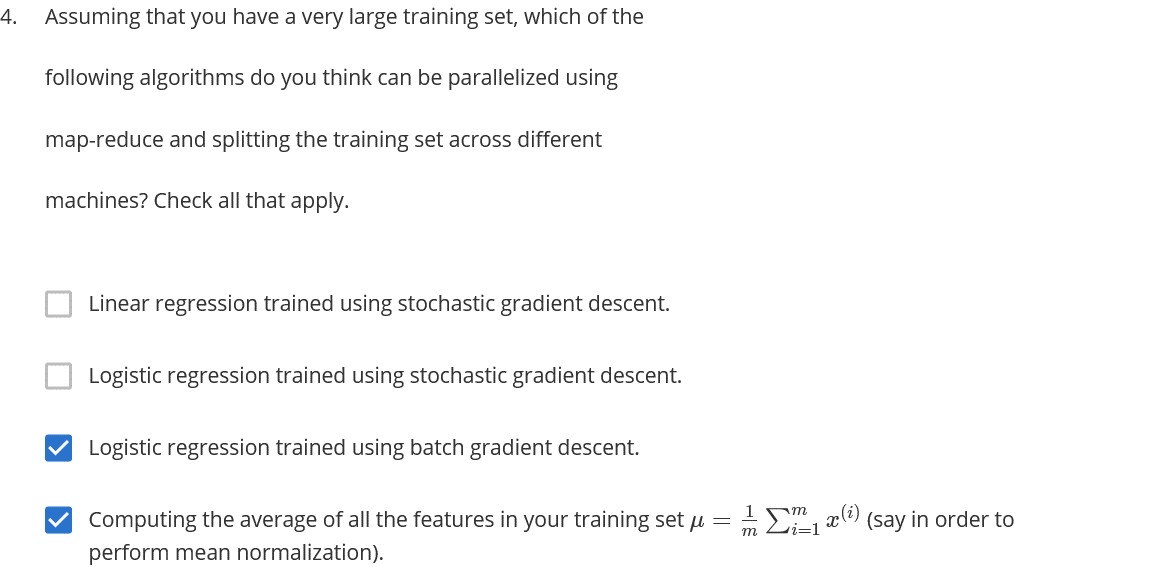
Mini batch gradient descent is better than batch gradient descent because in the former we take small batches and perform calculations. Hence the computational load doesn’t increase much.

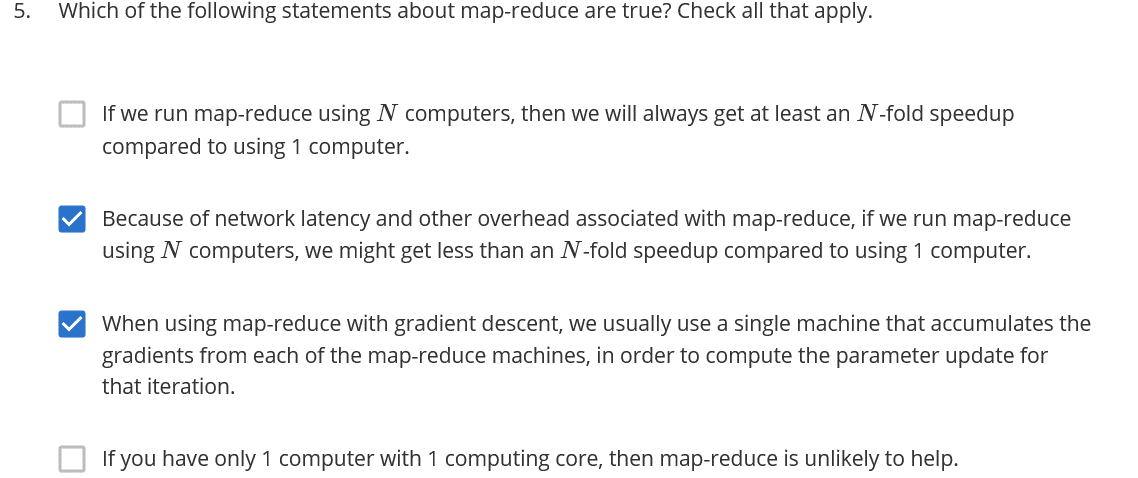
If we compare it to stochastic gradient descent, then it can be faster if we perform vectorized implementation using matrices. Doing small matrix multiplications for each batch again and again is fast as well as computationally efficient.







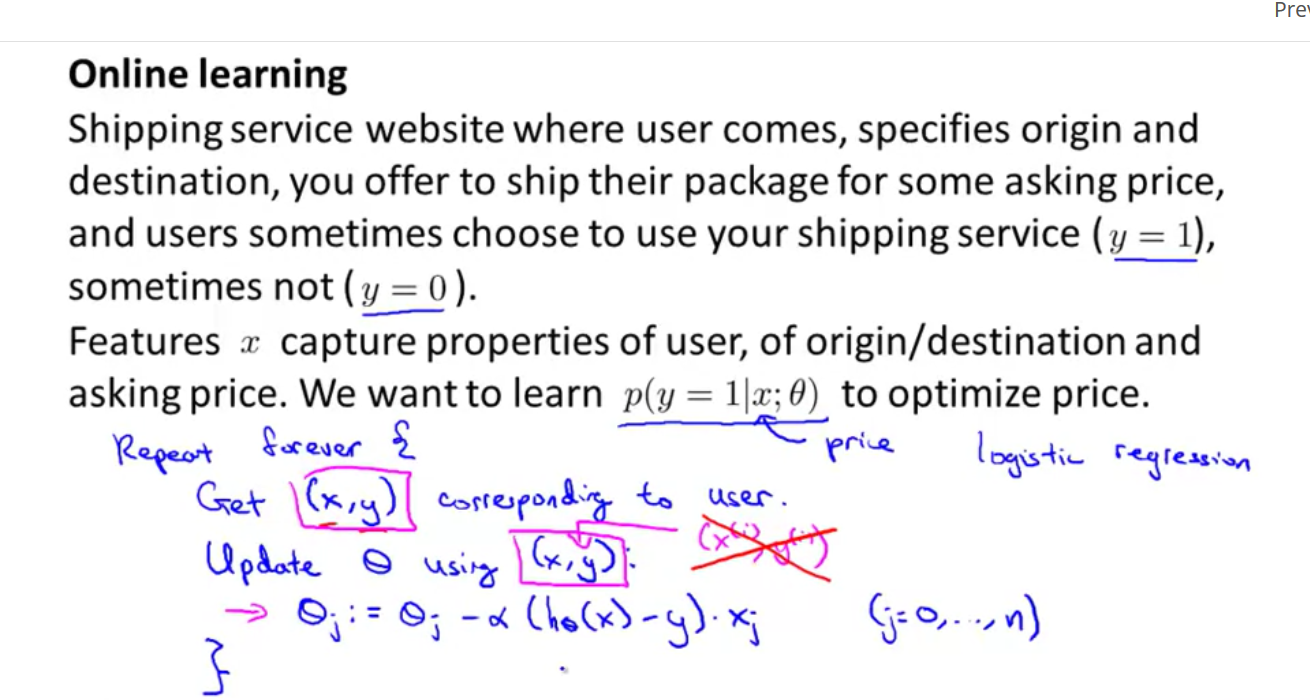


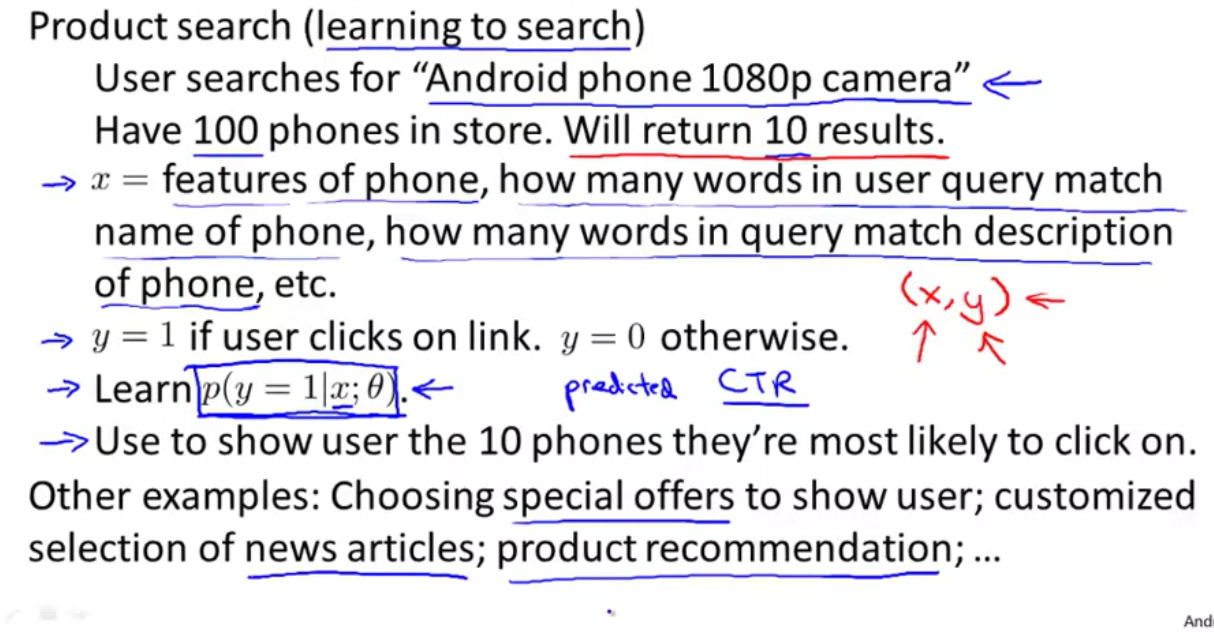


Though answer 5 was marked as wrong.

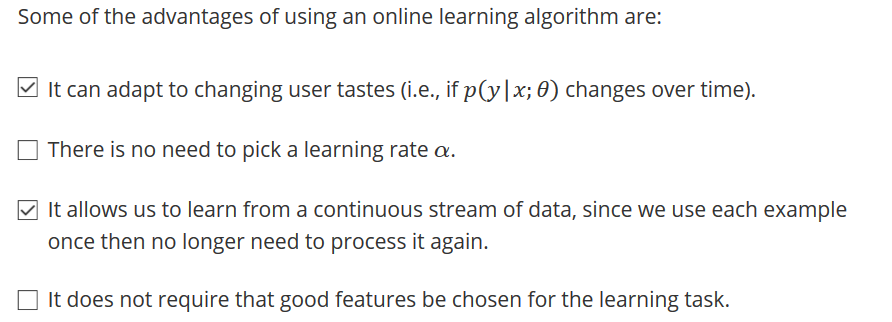
*Online Learning:*

In online learning, we do not have a fixed dataset but our data keeps on getting updated now and then. For example, if we have a website which is visited by many people every now and then, then instead of storing the data, we can instead update theta with each new data point and the discard that data point. Say a new user visits our website, then we can update theta for x values corresponding to this new user. After updating theta, we can then discard this data point.



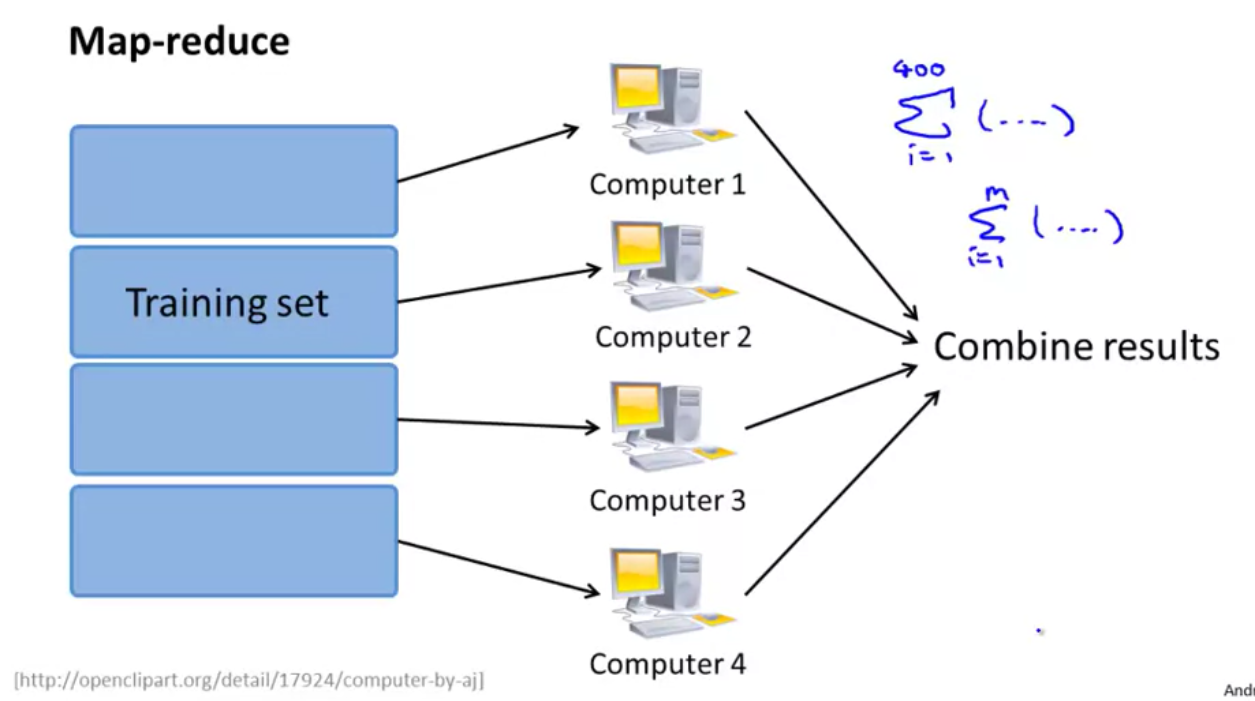


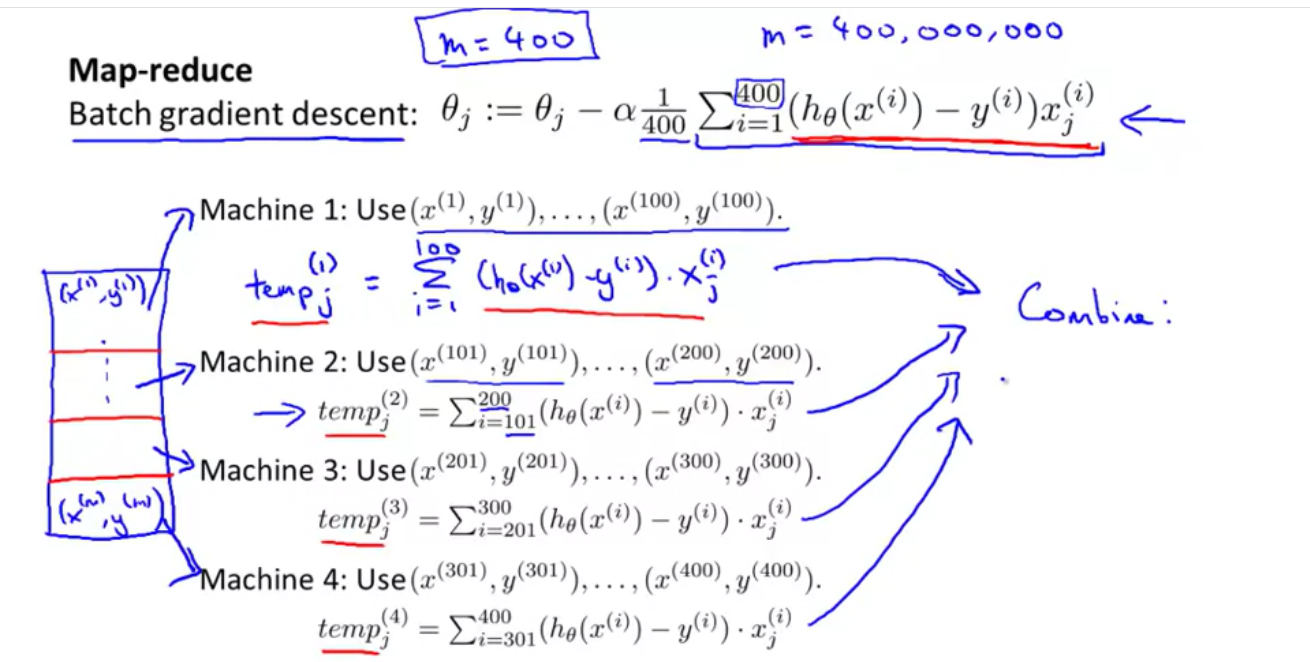
Product search is like deep learning in which we try to find the best set of data suitable on the basis of the parameters entered.



*Map Reduction:*

We can lessen the load of our computer by dividing the computation. For example, if we have to compute sum from 1 to m and m is very large. Then we can compute this sum on 4 computers instead of 1 and each machine will compute the sum of m/4 terms. We can then combine these sums on a single machine.





*Instead of using different set of machines, we can also split calculations on different cores of our processor:*

