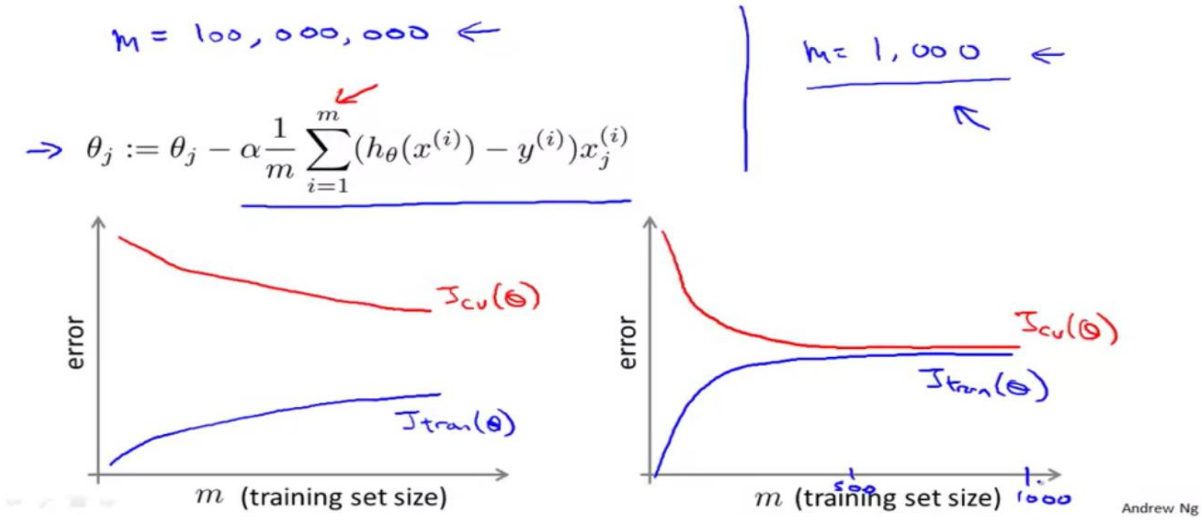
Deciding when to Use all the Rows of a very large Dataset:



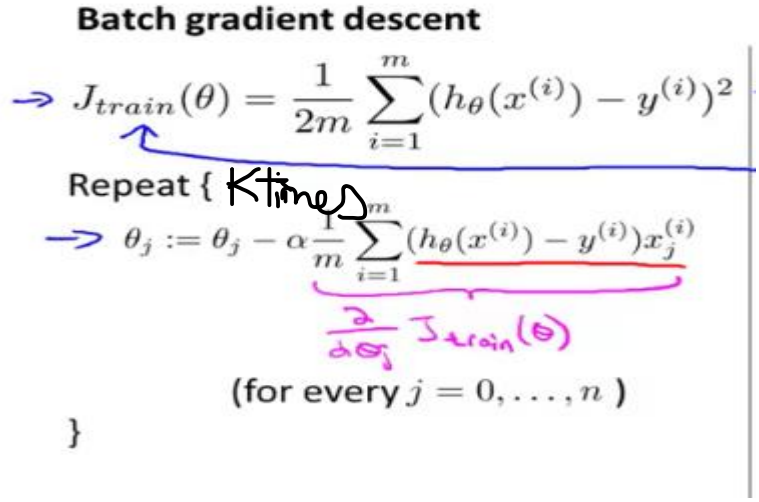
In the above picture we see that after plotting the learning curve, it becomes clear that whether using all the m datapoints will be helpful or not:

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.

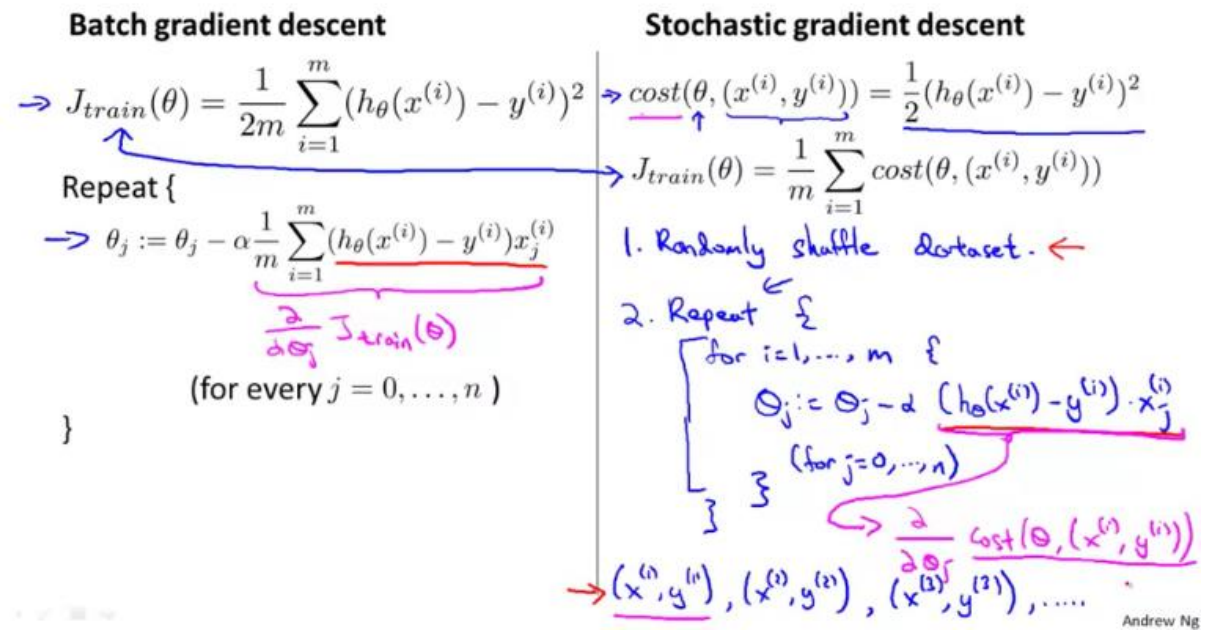
2. If we have a case of high bias or underfitting (as shown by the curve on the right), then using more training examples may not be helpful and we may work with say 1000 datapoints instead of all the m.

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).

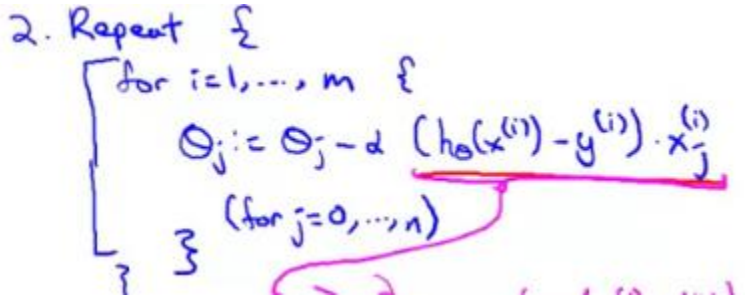
So how does this relax computational expense?

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

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

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.

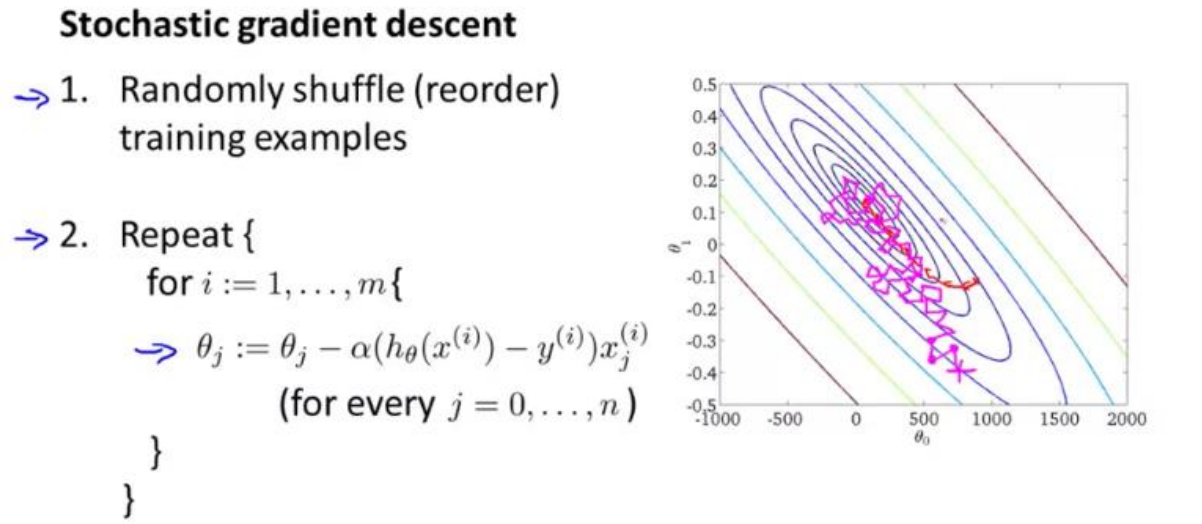
Now consider that we use stochastic gradient descent. Now while performing gradient descent:



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.

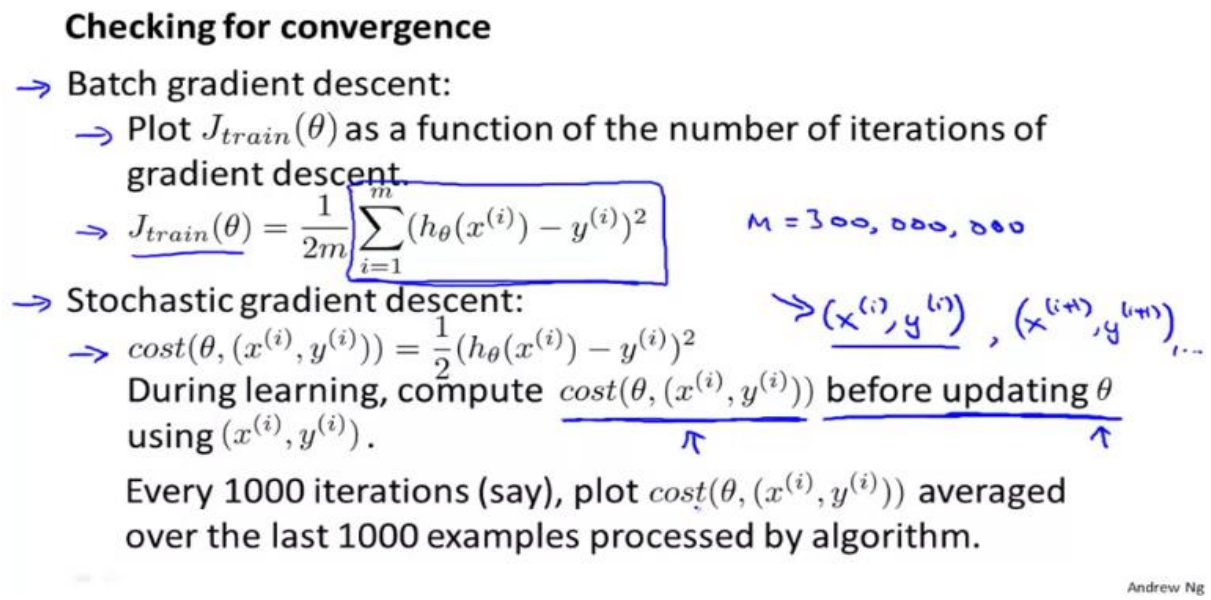
After the computing the sum from 1 to 100 million and then subtracting this value from theta, batch gradient descent moves 1 step ahead, hence it is slower than stochastic gradient descent.

Question:

Which of the following statements about stochastic gradient descent are true? Check all that apply.

- [x] When the training set size m is very large, stochastic gradient descent can be much faster than gradient descent.
- [x] The cost function $J_{\text{train}}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ should go down with every iteration of batch gradient descent (assuming a well-tuned learning rate α) but not necessarily with stochastic gradient descent.
- [ ] Stochastic gradient descent is applicable only to linear regression but not to other models (such as logistic regression or neural networks).
- [x] Before beginning the main loop of stochastic gradient descent, it is a good idea to "shuffle" your training data into a random order.

Evaluating Gradient Descent for Stochastic Gradient Descent:

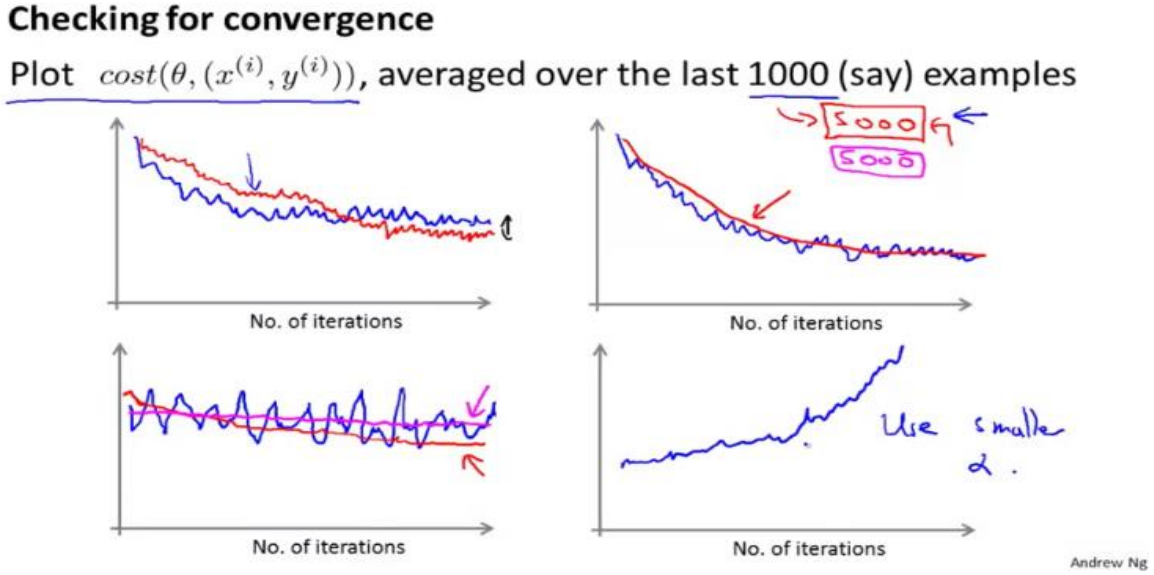


We run our algorithm m*r times where m is the number of data points from ranging from 1 to m and r is the number of times we are repeating the whole process. Now for every 1000 examples out of m, we can average the cost function and plot it on the graph cost vs number of iterations. This means for the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

(cost for x^1, y^1 +cost for x^2, y^2 + ….. + cost for x^{1000}, y^{1000}) /1000

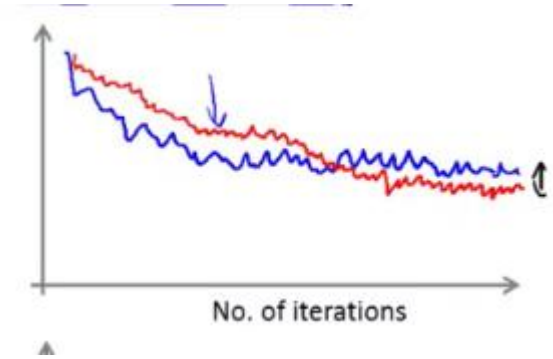
Then for the next 1000 iterations that is x^{1001}, y^{1001}…..x^{2000}, y^{2000} we do the same. We then plot average cost function value vs number of iterations.

Checking for convergence:



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

1.

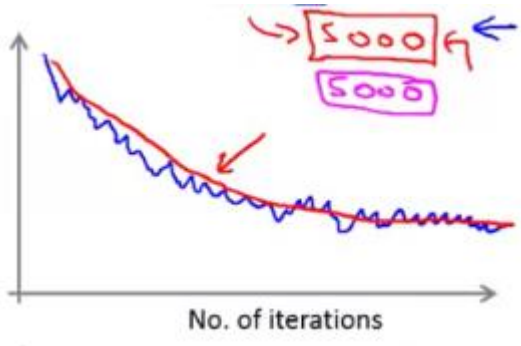


Here we have a plot of cost vs number of iterations where the cost function has been averaged for every 1000 iterations and iterations increase in sums of 1000 i.e. iteration 1 = 1000, iteration 2=2000 and so on.

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.

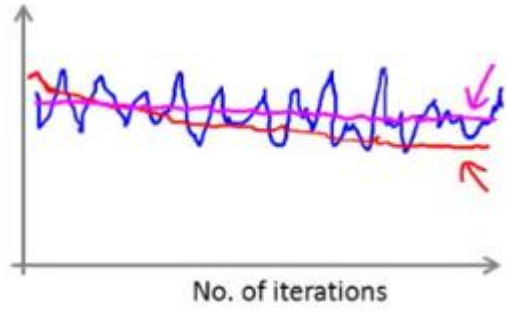
2.



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.

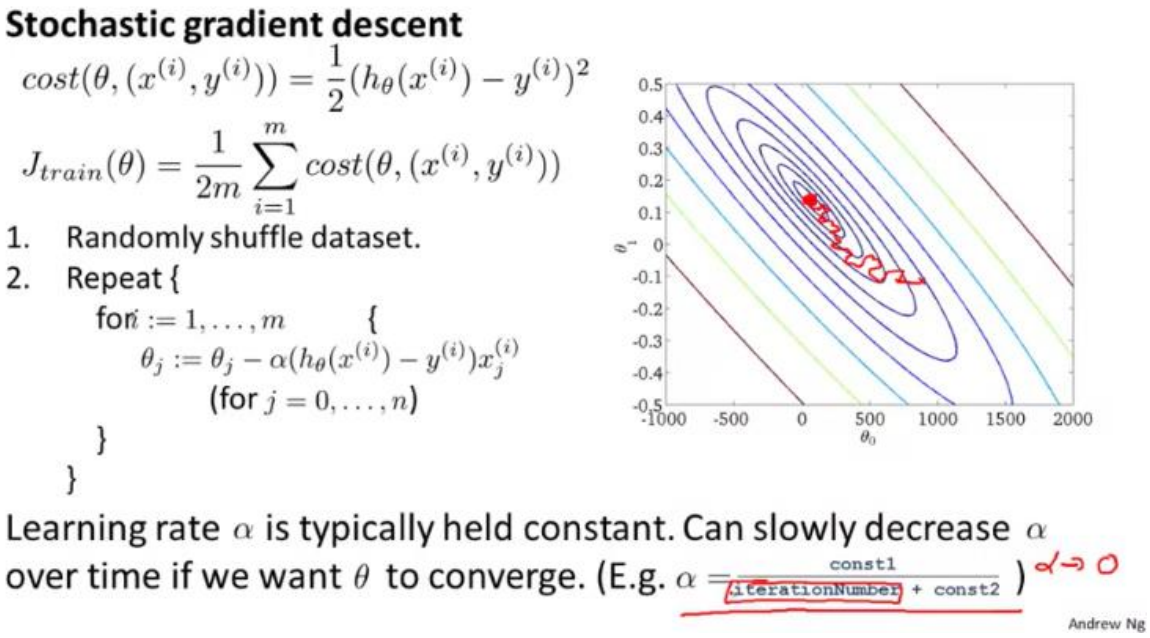
If even after increasing the number of data points, we get a line same as the pink line, we can infer that the cost function is not decreasing with increase in the number of iterations.

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.



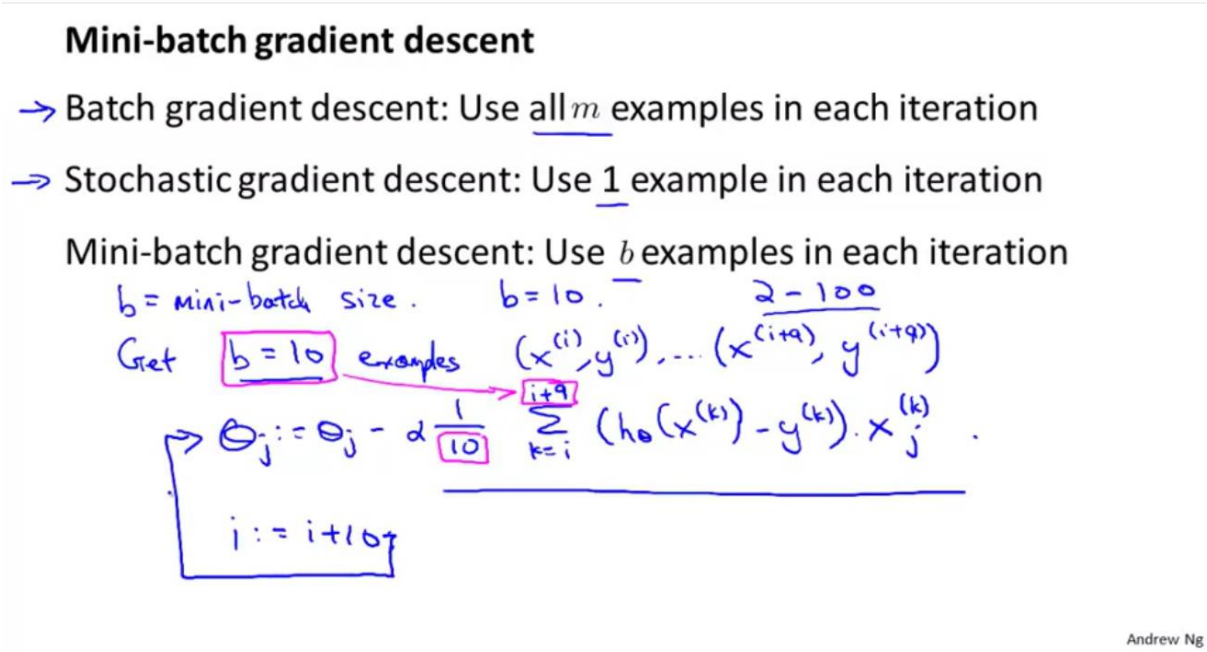
Question:

Which of the following statements about stochastic gradient descent are true? Check all that apply.

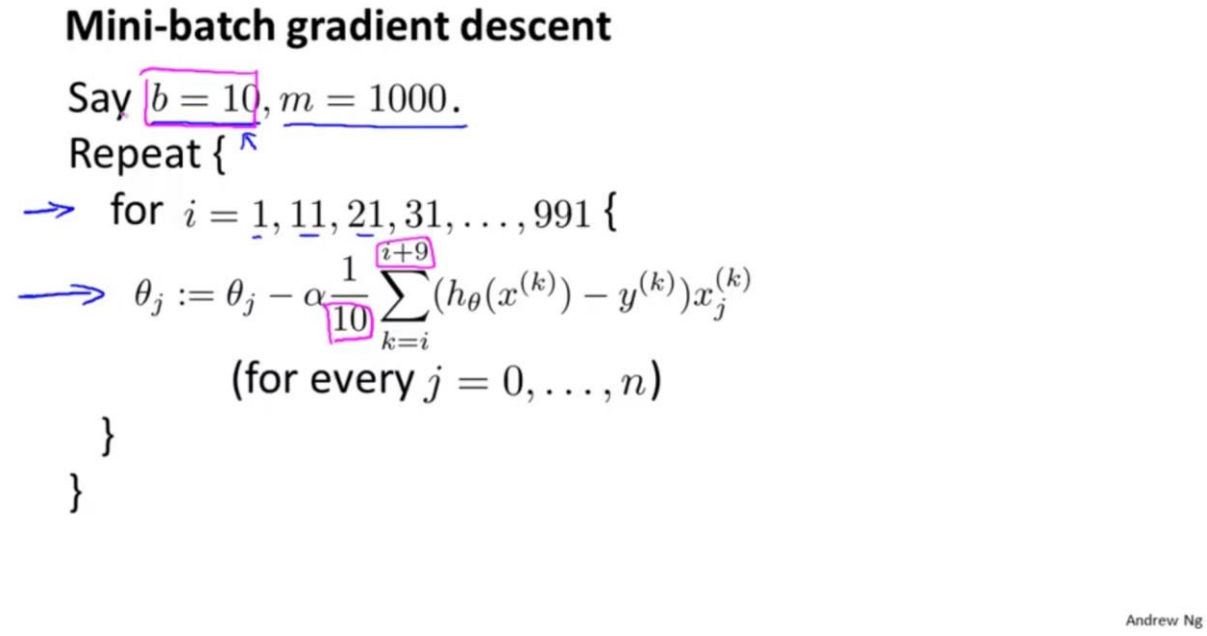
- [ ] Picking a learning rate α that is very small has no disadvantage and can only speed up learning.
- [x] If we reduce the learning rate α (and run stochastic gradient descent long enough), it's possible that we may find a set of better parameters than with larger α.
- [ ] If we want stochastic gradient descent to converge to a (local) minimum rather than wander of "oscillate" around it, we should slowly increase α over time.
- [x] If we plot $cost(\theta, (x^{(i)}, y^{(i)}))$ (averaged over the last 1000 examples) and stochastic gradient descent does not seem to be reducing the cost, one possible problem may be that the learning rate α is poorly tuned.

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.

1. Suppose you are training a logistic regression classifier using stochastic gradient descent. You find that the cost (say, $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 500 examples), plotted as a function of the number of iterations, is slowly increasing over time. Which of the following changes are likely to help?

- (●) Try halving (decreasing) the learning rate α, and see if that causes the cost to now consistently go down; and if not, keep halving it until it does.
- ( ) This is not possible with stochastic gradient descent, as it is guaranteed to converge to the optimal parameters θ.
- ( ) Try averaging the cost over a smaller number of examples (say 250 examples instead of 500) in the plot.
- ( ) Use fewer examples from your training set.

2. Which of the following statements about stochastic gradient

descent are true? Check all that apply.

- [x] Before running stochastic gradient descent, you should randomly shuffle (reorder) the training set.
- [ ] Suppose you are using stochastic gradient descent to train a linear regression classifier. The cost function $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ is guaranteed to decrease after every iteration of the stochastic gradient descent algorithm.
- [ ] In order to make sure stochastic gradient descent is converging, we typically compute $J_{\text{train}}(\theta)$ after each iteration (and plot it) in order to make sure that the cost function is generally decreasing.
- [x] You can use the method of numerical gradient checking to verify that your stochastic gradient descent implementation is bug-free. (One step of stochastic gradient descent computes the partial derivative $\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$.)

3. Which of the following statements about online learning are true? Check all that apply.

- [ ] One of the advantages of online learning is that there is no need to pick a learning rate α.
- [x] When using online learning, in each step we get a new example (x, y), perform one step of (essentially stochastic gradient descent) learning on that example, and then discard that example and move on to the next.
- [ ] One of the disadvantages of online learning is that it requires a large amount of computer memory/disk space to store all the training examples we have seen.
- [x] In the approach to online learning discussed in the lecture video, we repeatedly get a single training example, take one step of stochastic gradient descent using that example, and then move on to the next example.

4. Assuming that you have a very large training set, which of the following algorithms do you think can be parallelized using map-reduce and splitting the training set across different machines? Check all that apply.

- [ ] Linear regression trained using stochastic gradient descent.
- [ ] Logistic regression trained using stochastic gradient descent.
- [x] Logistic regression trained using batch gradient descent.
- [x] Computing the average of all the features in your training set $\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$ (say in order to perform mean normalization).

5. Which of the following statements about map-reduce are true? Check all that apply.

- [ ] If we run map-reduce using N computers, then we will always get at least an N-fold speedup compared to using 1 computer.
- [x] Because of network latency and other overhead associated with map-reduce, if we run map-reduce using N computers, we might get less than an N-fold speedup compared to using 1 computer.
- [x] When using map-reduce with gradient descent, we usually use a single machine that accumulates the gradients from each of the map-reduce machines, in order to compute the parameter update for that iteration.
- [ ] If you have only 1 computer with 1 computing core, then map-reduce is unlikely to help.

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.

Online learning

Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ($y = 1$), sometimes not ($y = 0$).

Features x capture properties of user, of origin/destination and asking price. We want to learn $p(y = 1|x; \theta)$ to optimize price.

price logistic regression

Repeat forever {
  Get (x, y) corresponding to user.
  Update θ using (x, y).
  → $\theta_j := \theta_j - \alpha (h_\theta(x) - y) \cdot x_j$ $(j = 0, \ldots, n)$
}

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.

(x, y) ←

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

→ Learn $p(y = 1|x; \theta)$. ← predicted CTR

→ 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; ...

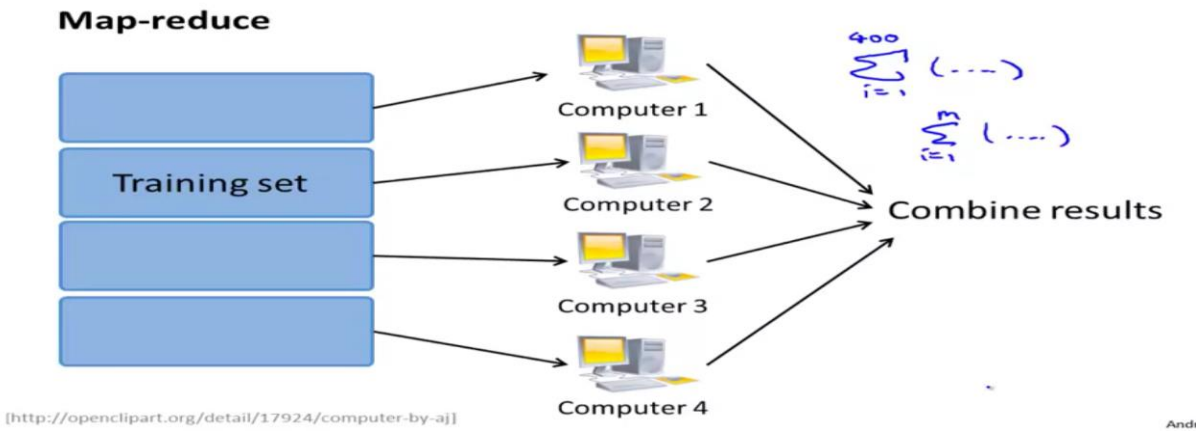
Product search is like deep learning in which we try to find the best set of data suitable on the basis of the parameters entered.

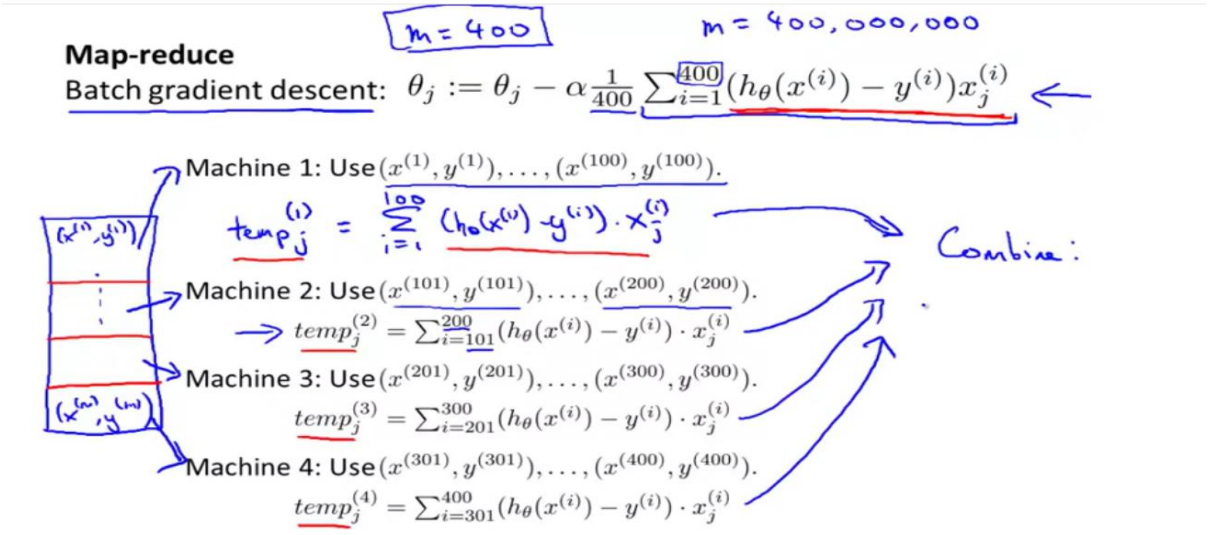
Some of the advantages of using an online learning algorithm are:

- [x] It can adapt to changing user tastes (i.e., if $p(y|x;\theta)$ changes over time).
- [ ] There is no need to pick a learning rate α.
- [x] It allows us to learn from a continuous stream of data, since we use each example once then no longer need to process it again.
- [ ] It does not require that good features be chosen for the learning task.

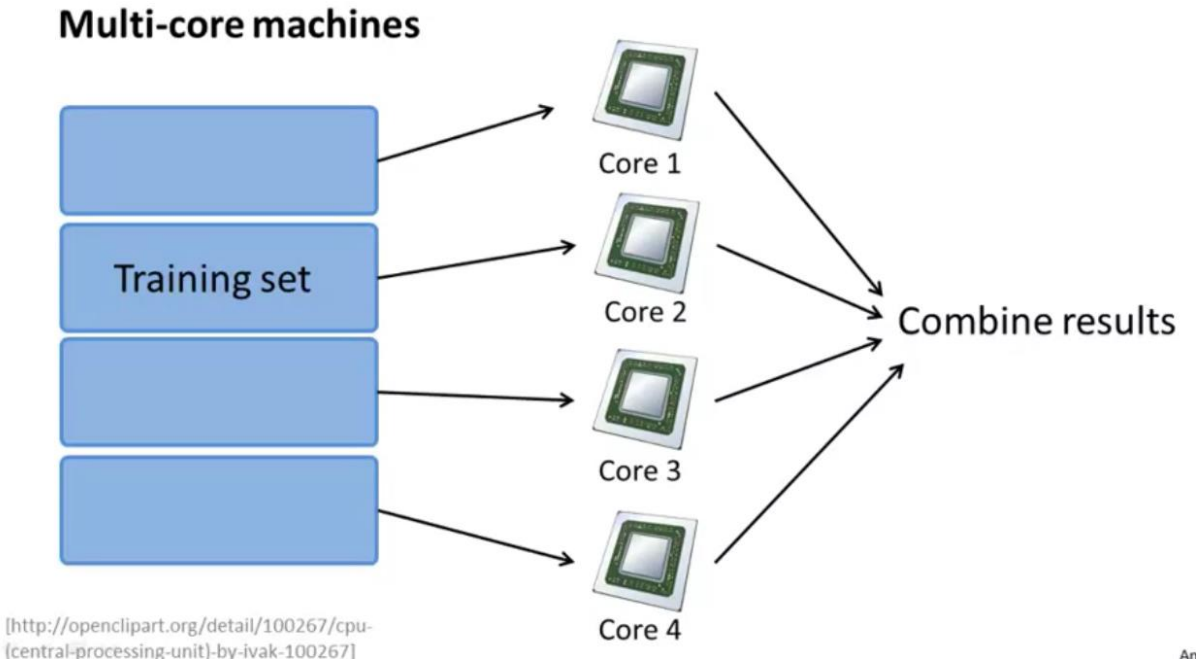
Map Reduction:

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:



Suppose you apply the map-reduce method to train a neural network on ten machines. In each iteration, what will each of the machines do?

- ○ Compute either forward propagation or back propagation on 1/5 of the data.
- ◉ Compute forward propagation and back propagation on 1/10 of the data to compute the derivative with respect to that 1/10 of the data.
- ○ Compute only forward propagation on 1/10 of the data. (The centralized machine then performs back propagation on all the data).
- ○ Compute back propagation on 1/10 of the data (after the centralized machine has computed forward propagation on all of the data).