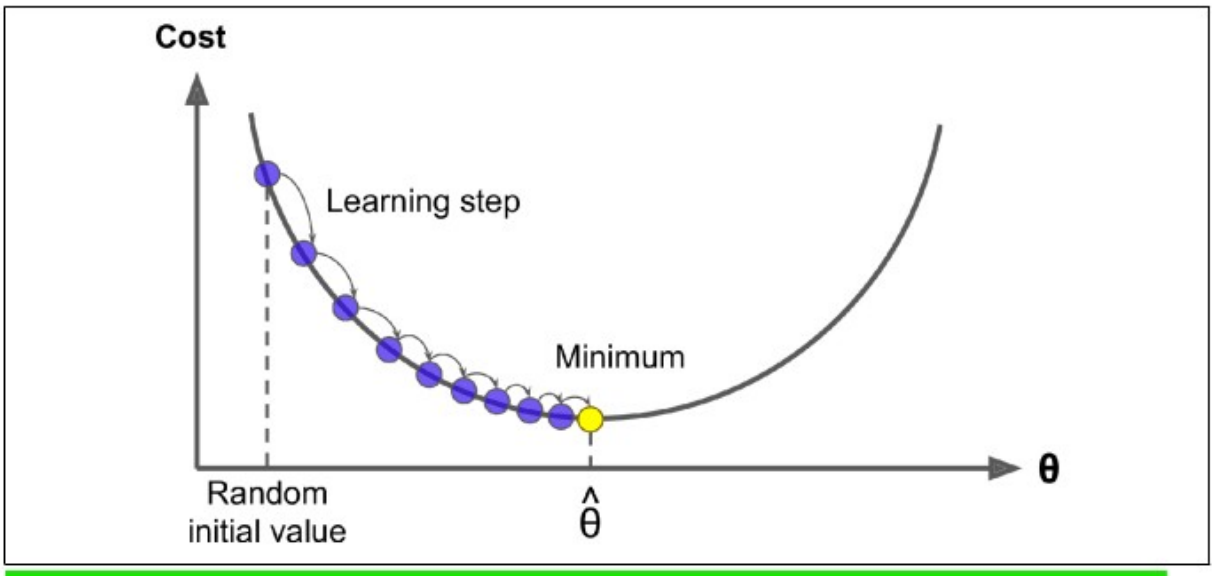
Gradient Descent

Gradient Descent is a generic optimization algorithm capable of finding optimal solution to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

Suppose we are lost in the mountains in a dense fog and we can only feel the slope of the ground below our feet. A good strategy to get to the bottom of the valley quickly is to go downhill in the direction of the steepest slope. This is exactly what Gradient Descent does , it measures the local gradient of the error function with regard to the parameter vector theta and it goes on the direction of descending gradient. Once the gradient is zero , we have reached to the minimum.

Concretely we start by filling theta with random values (this is called random initialization) Then we improve it gradually , taking one baby step at a time , each step attempting to decrease the cost function (Example the MSE) until the algorithm converges to a minimum.



In the depiction of Gradient Descent , the model parameters are initialized randomly and get tweaked repeatedly to minimize the cost function , the learning step size is proportional to the slope of the cost function , so the steps gradually get smaller as the parameters approach to the minimum.

An important parameter in Gradient Descent is the size of the steps , determined by the learning rate hyper parameter. If the learning rate is too

small , then the algorithm will have to go through many iterations to converge ,which will take a long time.

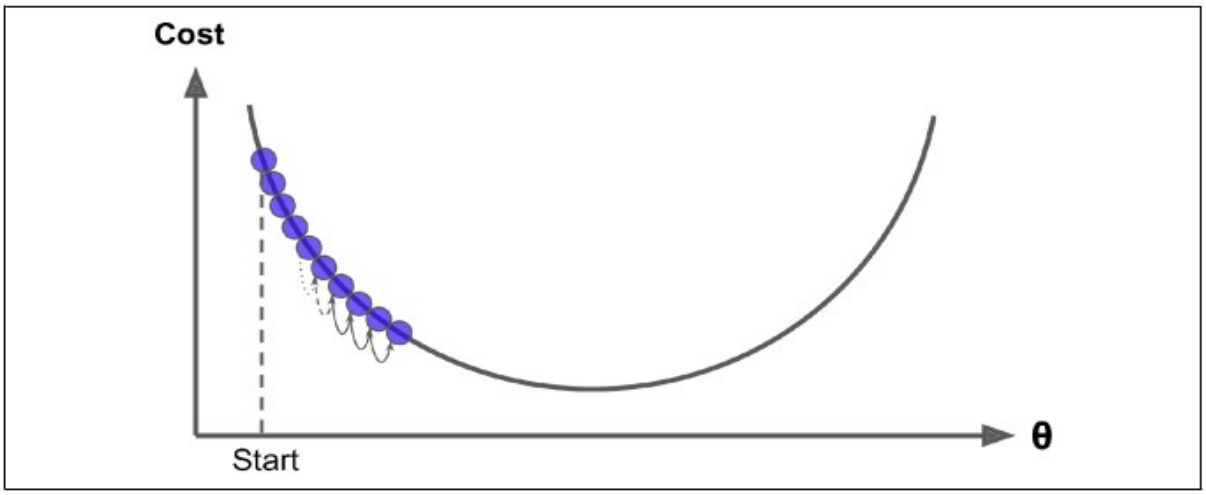


Figure 4-4. The learning rate is too small

On the other hand , if the learning rate is too high , we might jump across the valley and end up on the other side , possibly even higher up that we were before. This might make the algorithm diverge , with larger and larger values , failing to find a good solution.

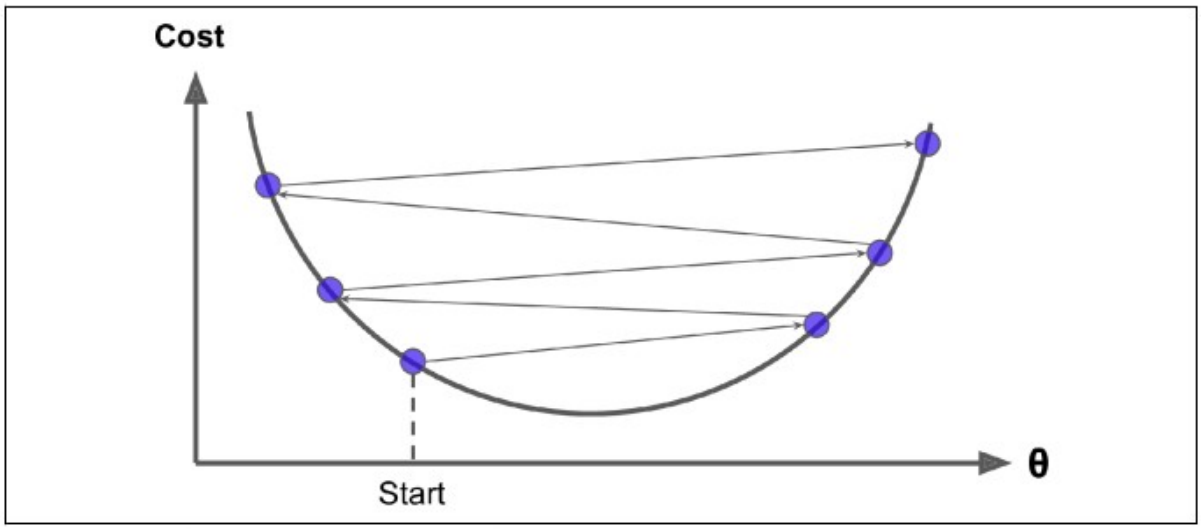


Figure 4-5. The learning rate is too large

Finally , not all cost functions look like nice regular bowls. There may be holes, ridges , plateaus and all sorts of irregular terrains , making convergence to the minimum difficult. The next figure shows the two main challenges with Gradient Descent. If the random initialization starts the algorithm on the left ,

then it will converge to a local minimum which is not as good as the global minimum. If it starts on the right , then it will take a very long time to cross the plateau and if we stop too early we will never reach to the global minimum.

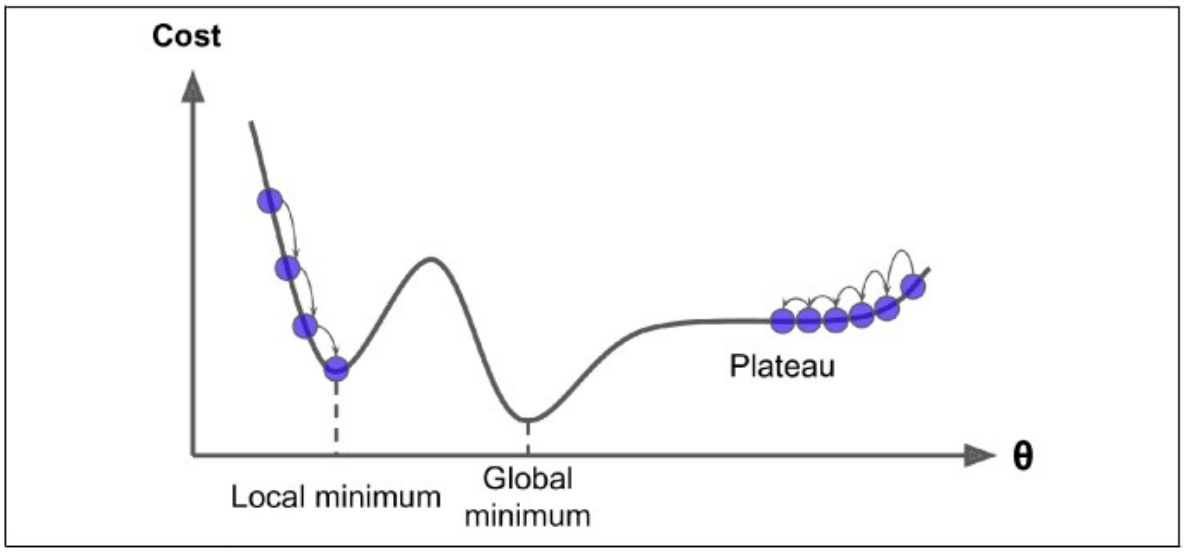


Figure 4-6. Gradient Descent pitfalls

Fortunately , the MSE cost function for a linear regression model happens to be a convex function , which means that if we pick any two points on the curve , the line segment joining them never crosses the curve. This implies that there are no local minima , just one global minimum.

In fact , the cost function has the shape of a bowl , but it can be an elongated bowl if the features have very different scales. In the figure shows the Gradient Descent on a training set where feature 1 and feature 2 have the same scale(the image on the left) and on a training set where feature 1 has much smaller values than feature 2(on the right)

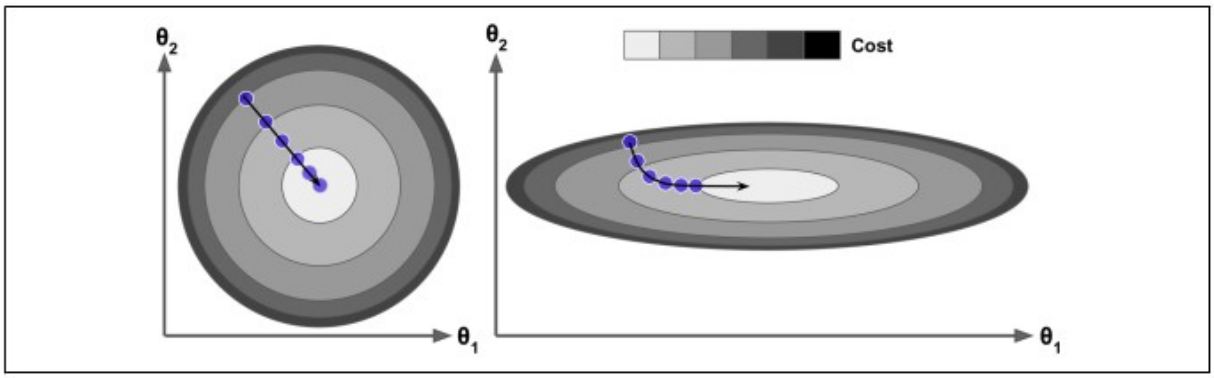


Figure 4-7. Gradient Descent with (left) and without (right) feature scaling

If the features are not scaled , the shape of the cost function will be skewed , it takes longer time to converge but if the features are scaled , it shortly goes to the minimum.

The best way to scale our features is to use a mean normalization

x1 = (1 - 2000 ft∧2) - size of the house

x2 = 1 - 5 - number of the bed rooms

use x-average instead of x

Mean-Normalization = #size of the house - average / # total

Mean-Normalization = #number of bedrooms - average / # total

- Most features are scaled between -1 <= features(x) <=1

- But it doesn't have to be only this , which means

- -3 <= x < = 3 * this is scaled

- -0.5 <= x <= 0.5 * this is also scaled

* - 0.0000001 <= x <= 0.0000001 - this is not scaled

* -100 <= x <= 100 * this is also not scaled

As we can see on the image above , on the left the Gradient Descent algorithm goes straight forward toward the minimum , there by reaching it quickly , where as on the right it first goes in the direction almost orthogonal

to the direction of the global minimum , and it ends with a long march down an almost flat valley , it will eventually reach the minimum but it will take a long time.

This diagram also illustrates the fact that training a model means searching for a combination of model parameters that minimizes a cost function (over the training set). It is a search in the model's parameters space , the more parameters a model has the more dimension this space has and the harder the search is , searching for a needle in a 300 dimensional haystack is much trickier than in 3 dimensions , Fortunately since the cost function is convex in the case of Linear regression , the needle is simply at the bottom of the bowl.

Gradient Descent is a more General algorithm and is used not only in linear regression , is actually used all over the place , it helps to minimize other cost functions as well.

Have some function $J(\theta_0, \theta_1)$ $J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$ $\min_{\theta_0 \ldots \theta_n}$

Outline:

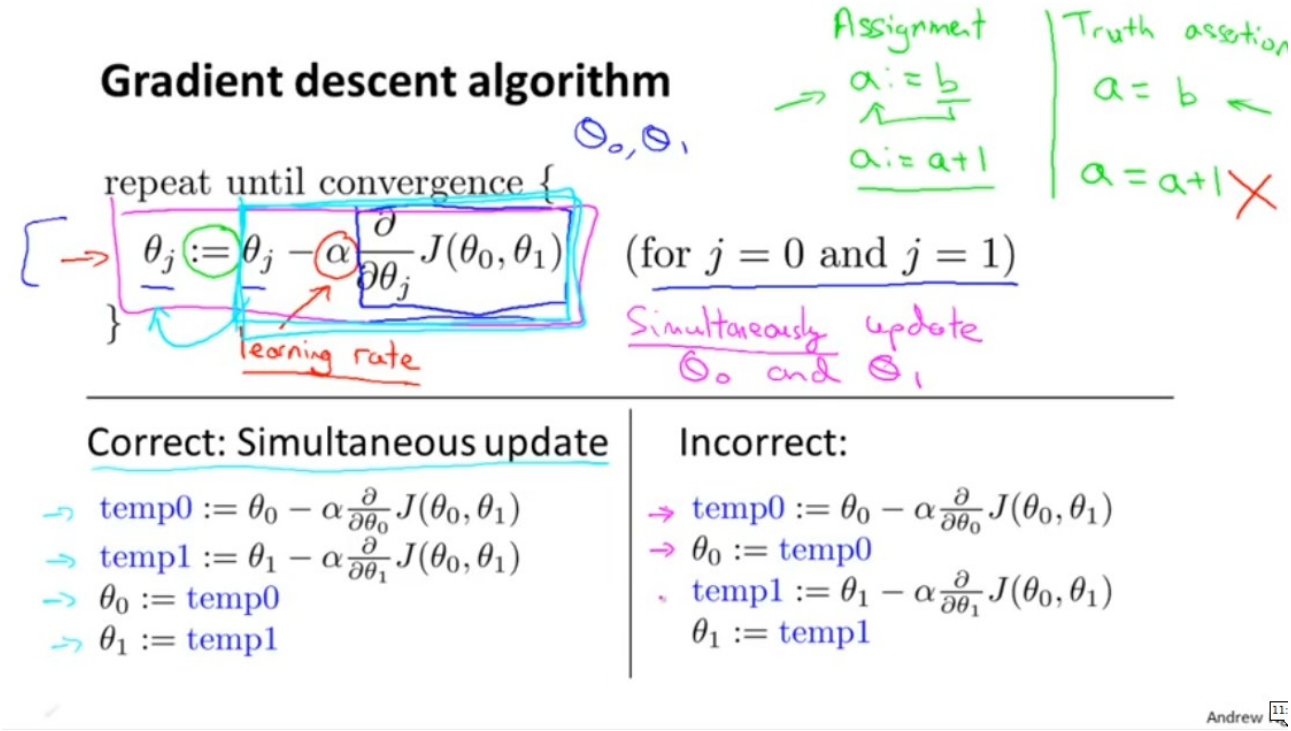
- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum

Here our cost function might be for a linear regression or may be for some other function.

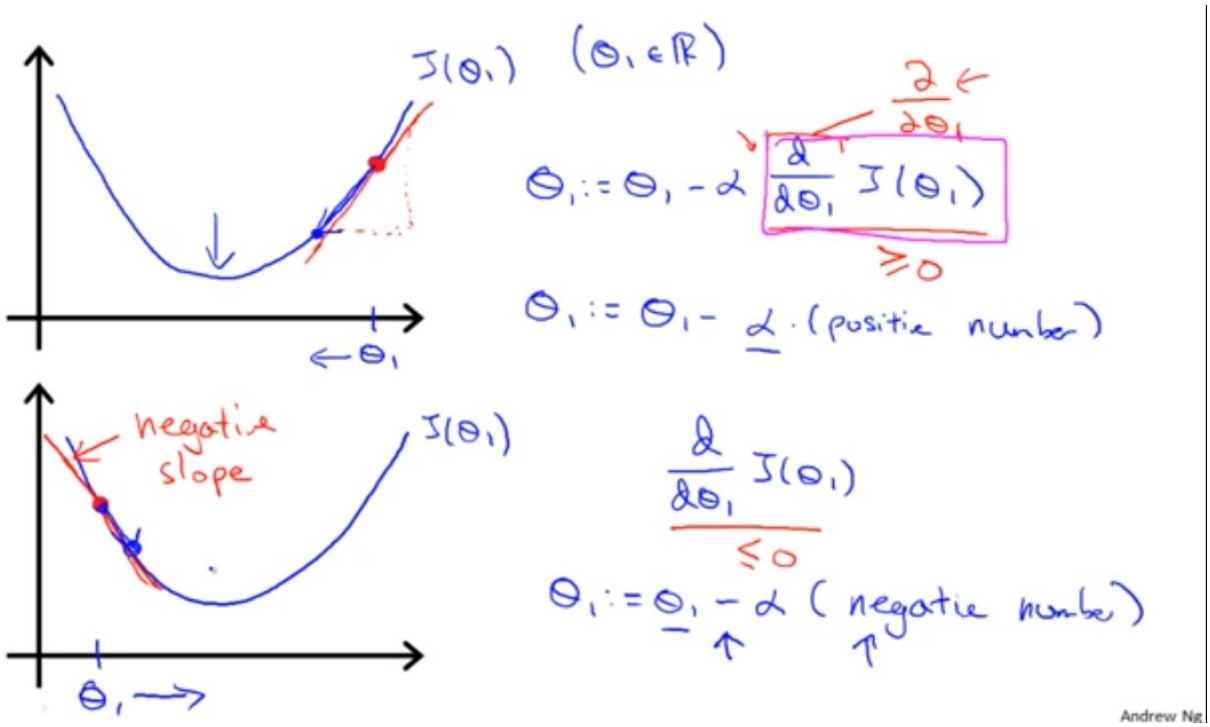
In a more General form gradient descent works for all parameters $J(\Theta_0 , \Theta 1, \Theta 2 , \ldots , \Theta n)$ but for simplicity in the above we will go with only two parameters $J(\Theta_0 , \Theta 1)$.

And at first we usually initialize the parameters will be random or full of zeros.

The Definition of the Gradient Descent Algorithm



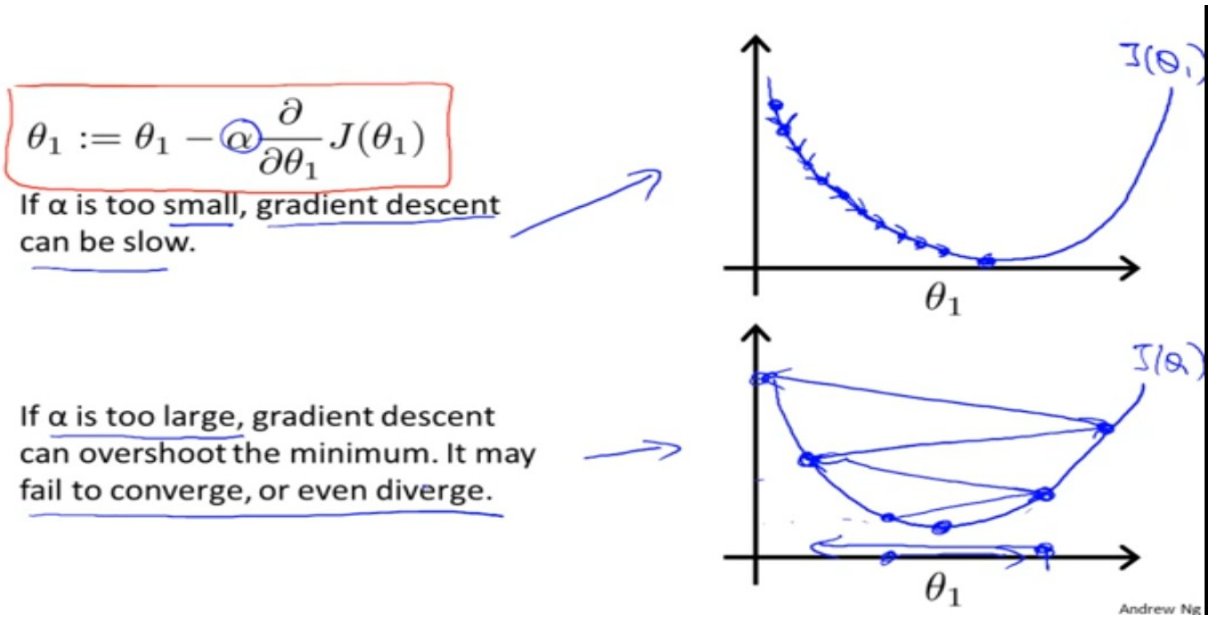
We first compute Θ_0 and then update it but when we compute Θ_1 the gradient is calculated with the updated Θ_0 which works but not the right way of doing it.



These is how the gradient (slope) works , if the slope of the cost function with respect to that parameter is increasing(positive slope) then performing the

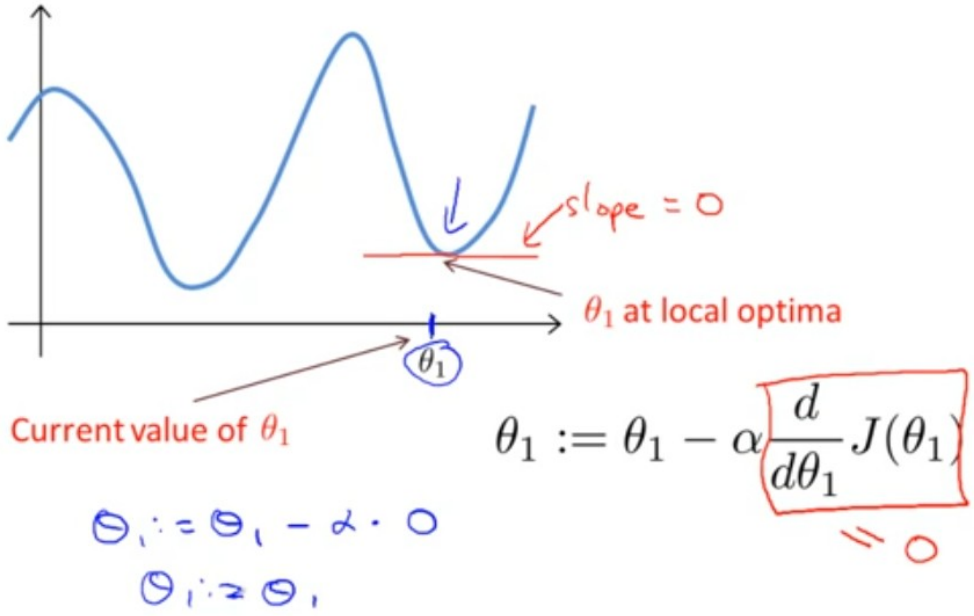
gradient equation which is multiplying that negative with the positive slope will make the entire formula on the right negative and then subtracting that from the previous parameter will make the updated parameter smaller than the previous one.

Whereas if the slope of the cost function with respect to that parameter is decreasing (Negative Slope) then performing the gradient equation which is multiplying that negative sign with the negative slope will make the entire formula on the right positive and then adding that from the previous parameter will make the updated parameter larger than the previous one.

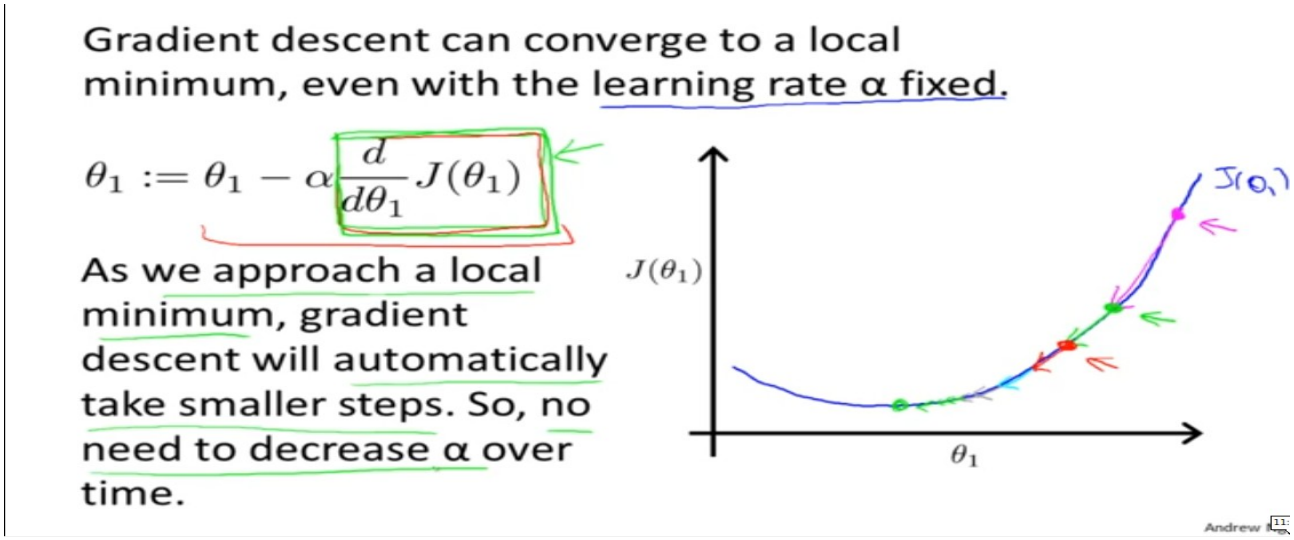


If the learning rate is too small , then the algorithm will have to go through many iterations to converge ,which will take a long time.

On the other hand , if the learning rate is too high , we might jump across the valley and end up on the other side , possibly even higher up that we were before. This might make the algorithm diverge , with larger and larger values , failing to find a good solution.



Let's suppose say we initialize Θ at the local minimum , it turns out the derivative is zero , the slope of a horizontal tangent is zero , so the updated parameter parameter won't have any change with the previous parameter.



The learning step size is proportional to the slope of the cost function , so the steps gradually get smaller as the parameters approach to the minimum.

Batch , Mini-Batch and Stochastic Gradient Descent

Batch Gradient Descent :-

Questions :-

1) What is Batch Gradient Descent

2) What kind of advantage can this give over the normal gradient descent

3) Why is it called Batch Gradient Descent

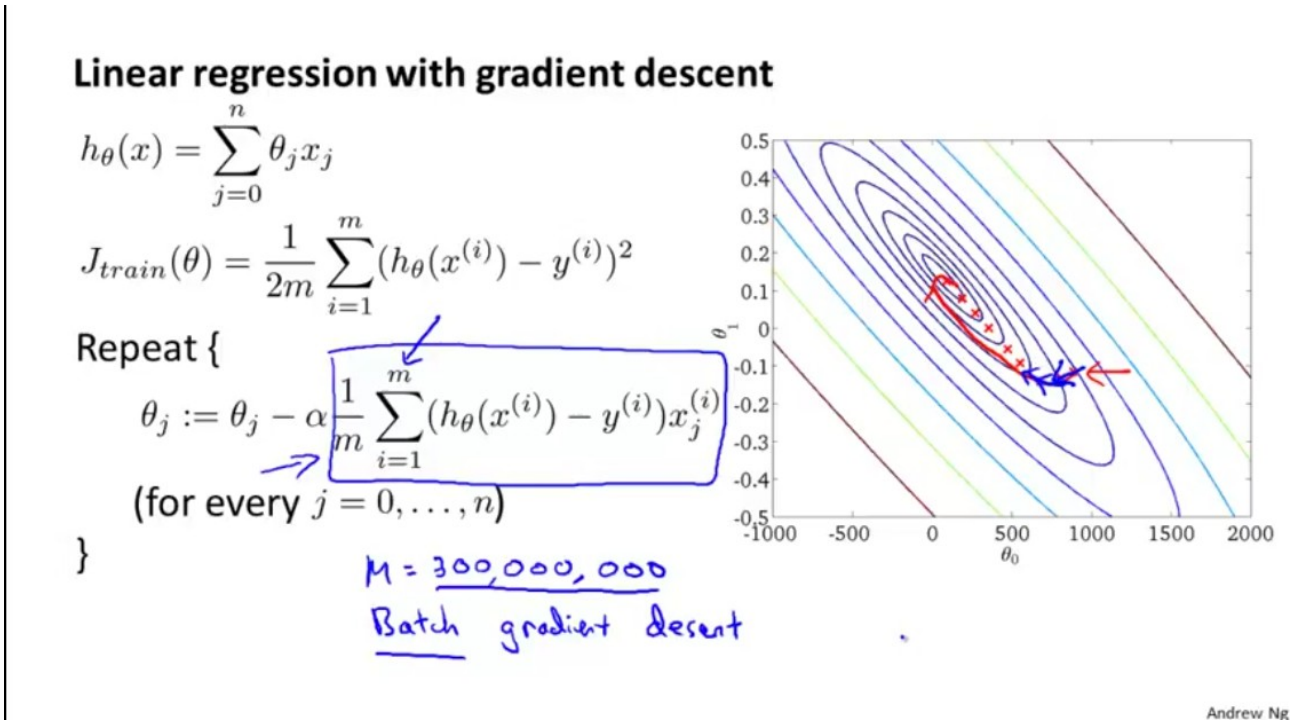
In Batch Gradient Descent, all the training data is taken into consideration to take a single step. We take the average of the gradients of all training examples and then use the mean gradient to update our parameters. So that is just one step gradient step in one epochs.

Notice that this formula involves calculations over the full training set X, at each Gradient Descent step! This is why the algorithm is called Batch Gradient Descent: it uses the whole batch of training data at every step (actually, Full Gradient Descent would probably be a better name). As a result it is terribly slow on very large training sets (but we will see much faster Gradient Descent algorithms shortly).

Batch Gradient descent (Mini-Batch size = m) processing a huge training set on every iteration , so the main advantage is too much time per iteration assuming that you have very large training instance , If you have a small training set the batch gradient is fine to use.

Batch Gradient descent mostly helps for those that have a convex cost function. Batch gradient descent are mostly in problem when we have large training datasets , gradient descent becomes a computationally very expensive procedure.

Suppose we have a training Linear Regression with gradient descent



For now we are going to keep using Linear Regression as the running example but the idea of gradient descent is fully general and also applies to other learning algorithms like Logistic Regression , neural networks and other algorithms.

The problem of gradient descent is when m is large , computing the derivative term can be very expensive because this requires summing over all m examples. To give the algorithm a name this particular version of Gradient descent also called batch gradient , the term batch refers we are looking all of the training examples at a time , the batch of all the training examples. So Batch algorithm takes a long time to converge to the minimum.

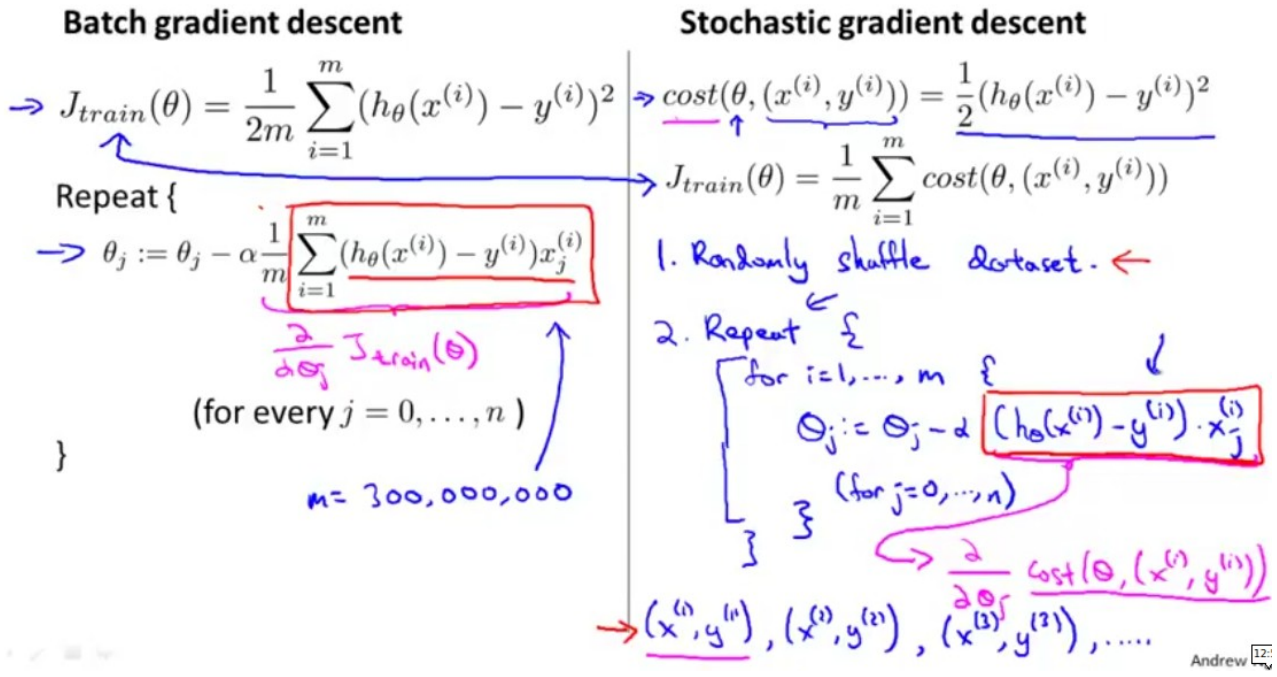
Stochastic Gradient Descent

In Batch Gradient Descent we were considering all the examples for every step of Gradient Descent. But what if our data-set is very huge. Deep learning models crave for data. The more the data the more chances of a model to be good. Suppose our data-set has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples. This does not seem an efficient way. To tackle this problem we have

Stochastic Gradient Descent. In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a single step.

We do the following steps in one epoch for SGD:

1. Take an example
2. Feed it to Neural Network
3. Calculate it's gradient
4. Use the gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the examples in training data-set

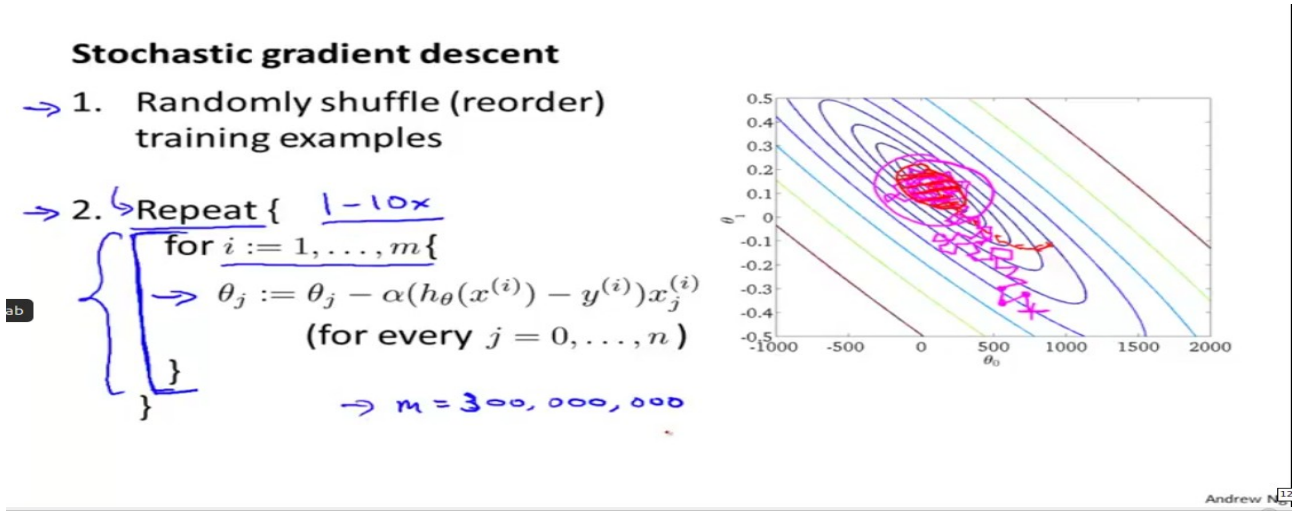


So this cost function term really measures how well is our hypothesis is doing on a single example $x^{(i)}$, y(i). The first step of stochastic gradient descent is to randomly shuffle the data-set.

So what stochastic gradient descent is doing scanning through the training examples and first it's gonna look out first training example ($x^{<1>}$, $y^{<1>}$) and then looking up only this first training example , we are gonna take a little gradient step having done this we are going to take on the second training

example , ($x^{<2>}$, $y^{<2>}$) so we will take a little gradient step on the gradient space just for the second example , then go to ($x^{<3>}$, $y^{<3>}$) and so on until we end up to the entire training set. So we are going to repeat this for some number of epochs and the randomly shuffle can speed up the gradient step.

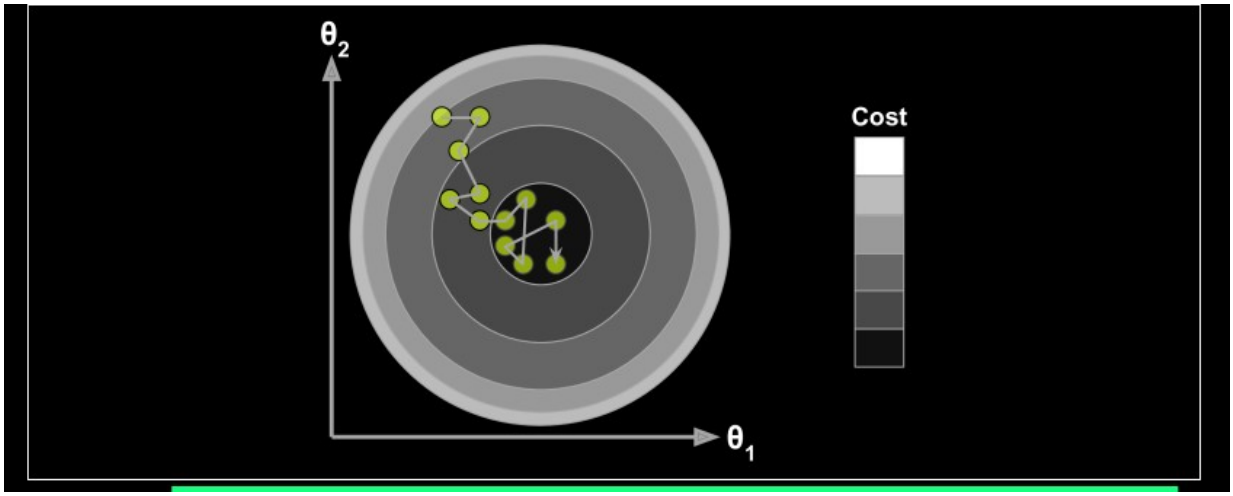
Another view of stochastic gradient descent is that a lot more like Batch gradient descent but rather than waiting to sum up the gradient terms for all of the m examples what we doing is we are taking this gradient term using just one single training example and we start making progress in improving the parameters , rather than taking a pass through all m training examples we just need to look out a single training examples and we already start making progress towards moving the parameters to the global minimum.



Previously we saw that when we using batch gradient descent what is the algorithm that looks all the training examples at a time , batch gradient descent will tend to make a reasonably straight line trajectory to get to the global minimum like that. In contrast what a stochastic gradient descent tell us is every iteration is going to be much faster because we don’t need to sum up over all the training examples but every iteration is just trying to fit a single training example better.

Since we are considering just one example at a time the cost will fluctuate over the training examples and it will not necessarily decrease. But in the long run, you will see the cost decreasing with fluctuations.

In Stochastic , this algorithm is much less regular than Batch Gradient Descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average. Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down , So once the algorithm stops, the final parameter values are good, but not optimal.



When the cost function is very irregular, this can actually help the algorithm jump out of local minima, so Stochastic Gradient Descent has a better chance of finding the global minimum than Batch Gradient Descent does.

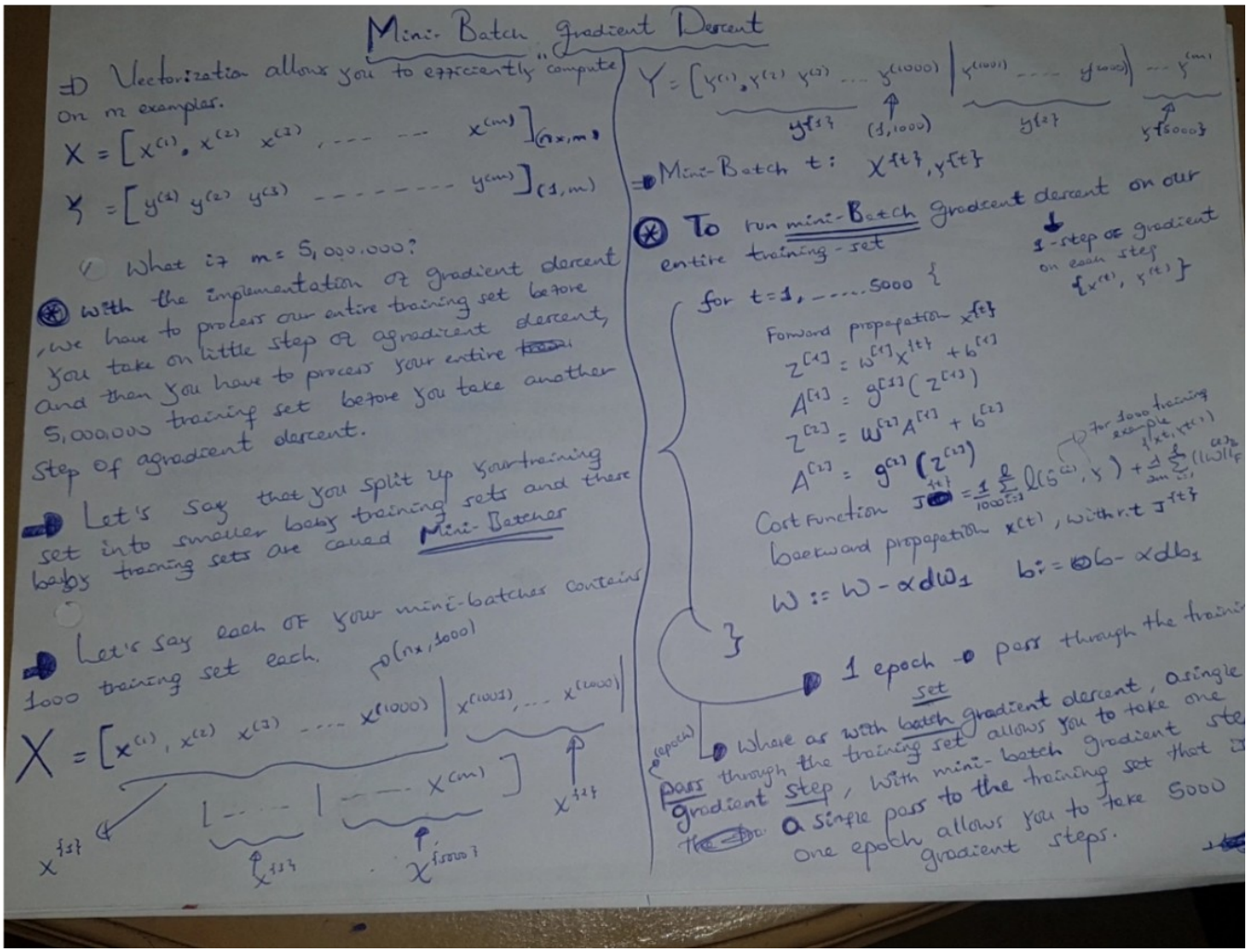
Therefore, randomness is good to escape from local optima, but bad because it means that the algorithm can never settle at the minimum. One solution to this dilemma is to gradually reduce the learning rate. The steps start out large (which helps make quick progress and escape local minima), then get smaller and smaller, allowing the algorithm to settle at the global minimum.

The function that determines the learning rate at each iteration is called the learning schedule. If the learning rate is reduced too quickly, you may get stuck in a local minimum, or even end up frozen halfway to the minimum. If the learning rate is reduced too slowly, you may jump around the minimum for a long time and end up with a suboptimal solution if you halt training too early.

In Stochastic gradient descent the randomness(Noise in the movement of the data) is not that much a problem , because we can decrease the learning rate but the big problem is you loose almost your speed up from vectorization , Because you are processing only a single example at a time , so the way you process each example is inefficient.

Mini-batch Gradient Descent

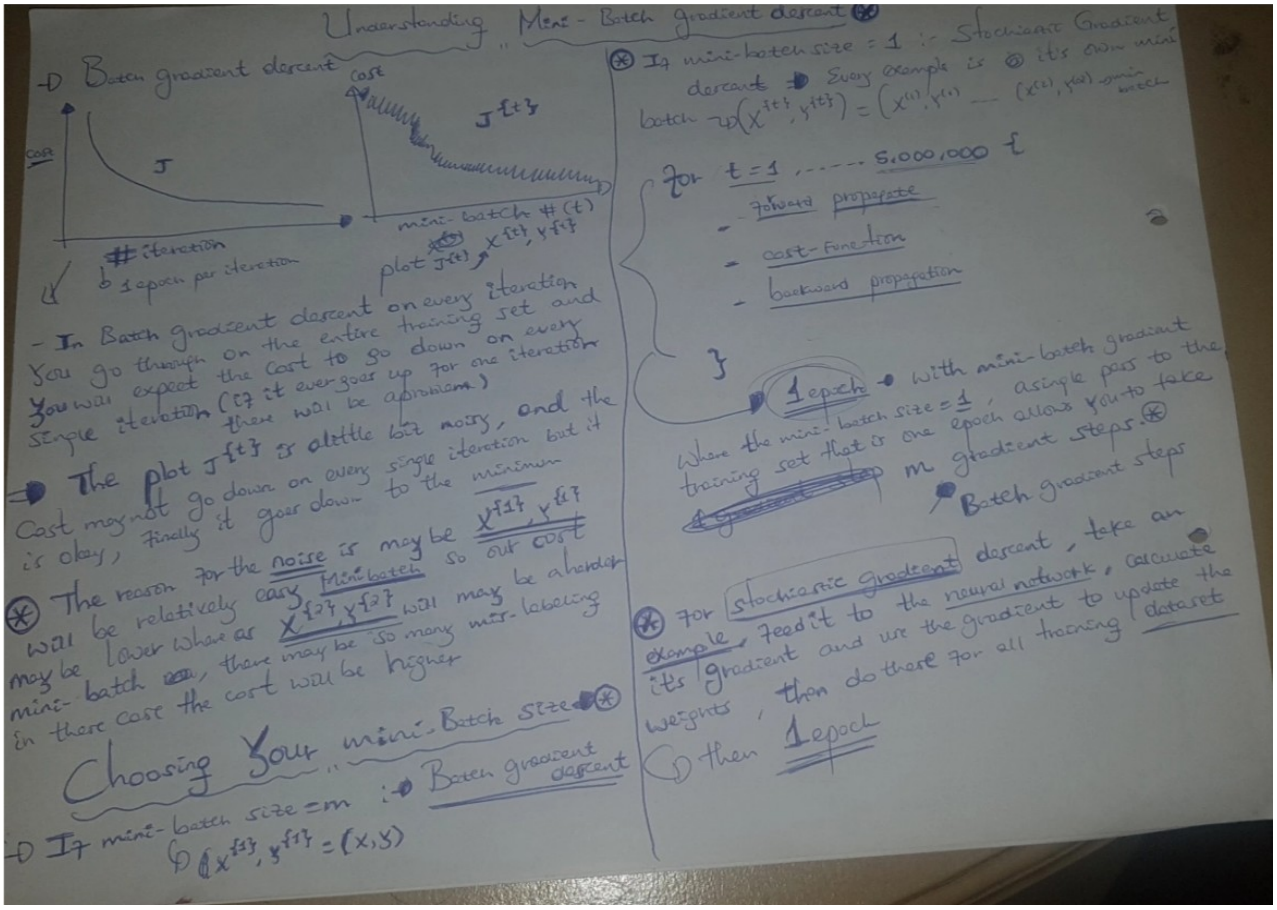
The last Gradient Descent algorithm we will look at is called Mini-batch Gradient Descent. It is simple to understand once you know Batch and Stochastic Gradient Descent: at each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini-batch GD computes the gradients on small random sets of instances called mini-batches.



Let's say our entire training data-set is **5,000,000** , With the implementation of the gradient descent we have to process our entire training set before you take on little step of a gradient descent , and then you have to process your entire set before you take another step of a gradient step.

Let's say that you split up your training set in to smaller baby sets and these baby training sets are called Mini – Batches.

In a Batch Gradient step , a single pass (epochs) through the training set allows you to take one gradient step , with mini batches gradient step a single pass to the training set that is one epoch allows you to take 5000 gradient steps.



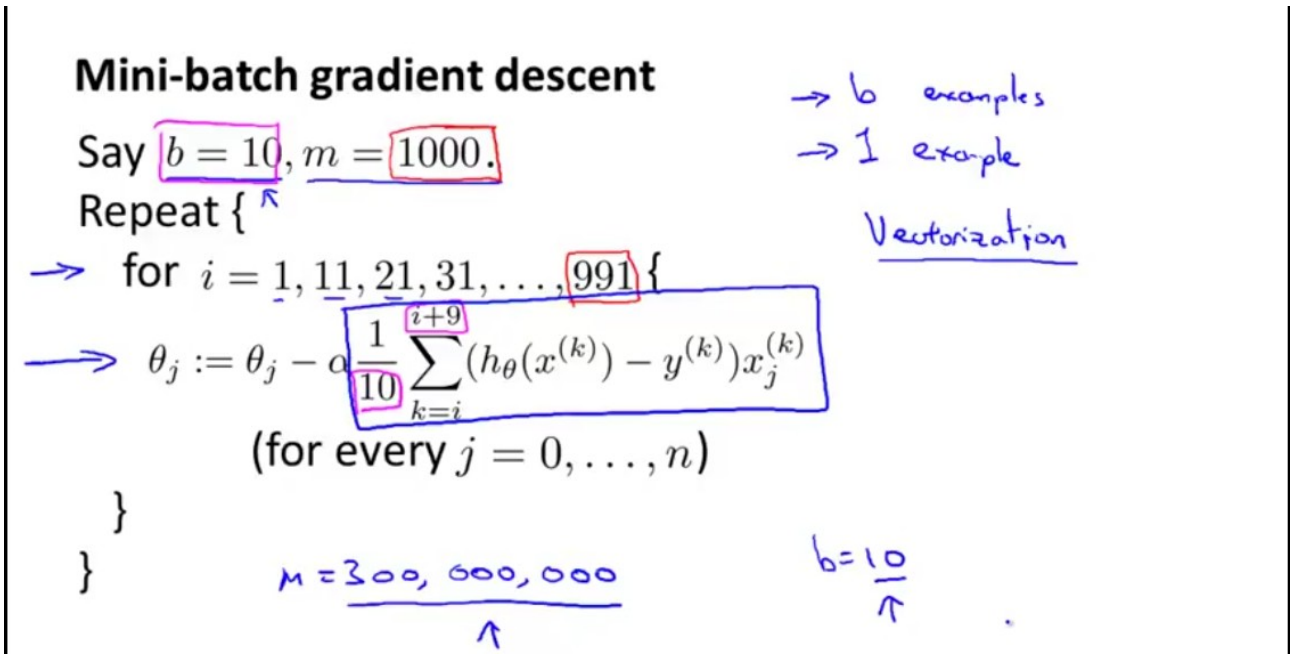
In Batch gradient descent on every iteration you go through on the entire training set and you will expect the cost to go down on every single iteration (If it ever goes up for one iteration there will be a problem).

The plot $J^{\{t\}}$ is a little bit noisy and the cost may not go down on every single iteration but it is okay , finally it goes down to the minimum.

The reason for the noise is may be $x^{\{1\}}$, $y^{\{1\}}$ will be relatively easy mini batches our cost may be lower where as $x^{\{2\}}$, $y^{\{2\}}$ will may be a harder mini batch there may be so many misleading in these case the cost will be higher.

- ✔ In a Batch gradient descent :- use all m examples in each iteration
- ✔ Stochastic Gradient Step :- use 1 example in each iteration
- ✔ Mini Batch Gradient Descent :- use b examples in each iteration

What B is a parameter called Mini - Batches , a typical choice for is 10 , the range might be any where from 2-100. The idea is rather than using 1 example at a time or m example at a time we are going to use b examples at a time.



why do we need to look out b examples at a time rather than looking out 1 example at a time as a stochastic gradient step , the answer for this is vectorization , the answer for this is Mini - Batch gradient descent is likely to outperform stochastic gradient descent. Since we have a good vectorization implementation in that case the gradient sum can be performed over with 10 examples in a more vectorized way which will allow us to partially parallelize our computation over 10 examples.

One disadvantage of a mini-batch gradient descent is there is this extra parameter B , which we need to tweak on , this take time and some search.

Choosing Your Mini Batch Size

If Mini batch size = m :- Batch gradient descent

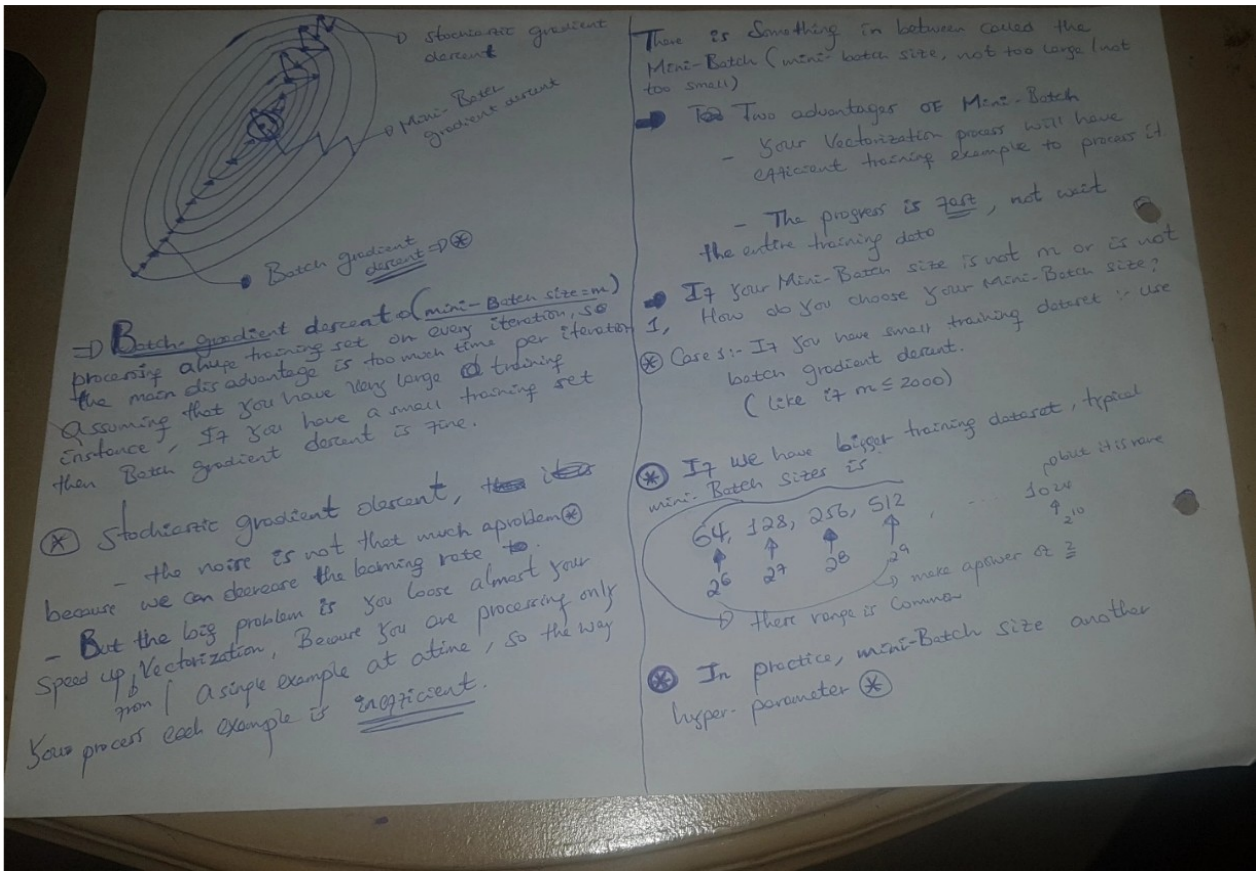
$x^{\{1\}}$, $y^{\{1\}}$ = (x , y)

If Mini batch size = 1 :- Stochastic Gradient Descent -> Every example is it's own mini batch -> ($X^{\{1\}}$, $y^{\{1\}}$) = ($x^{\{1\}}$, $y^{\{1\}}$) ...($x^{\{2\}}$, $y^{\{2\}}$).

Batch Gradient descent (Mini-Batch size = m) processing a huge training set on every iteration , so the main advantage is too much time per iteration assuming that you have very large training instance , If you have a small training set the batch gradient is fine to use.

In Stochastic gradient descent the randomness(Noise in the movement of the data) is not that much a problem , because we can decrease the learning rate but the big problem is you loose almost your speed up from vectorization , Because you are processing only a single example at a time , so the way you process each example is inefficient.

There is something in between called the mini batch (mini batch size , not too large , not too small)



Two advantages Of Mini -Batch:-

- Your Vectorization process will have efficient training example to process it
- The progress is fast, not wait the entire training data.

If your Mini-Batch size is not m or is not 1 , How do you choose your mini batch size ?

Case 1:- If you have small training data-set , just use batch gradient descent. (like if m<=2000)

Case 2 :- If we have a bigger training data-set , typical mini batch size is

64 , 128 , 256 , 512 , -----, 1024

- In Practice , Mini Batch size is another hyper parameter that we must tune.