Stochastic Gradient Descent

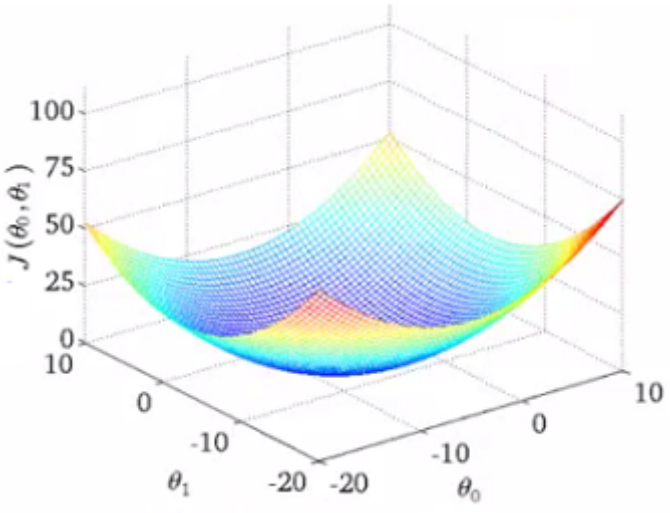
For many learning algorithms, among them linear regression, logistic regression and neural networks, the way we derive the algorithm was by coming up with a cost function or coming up with an optimization objective and then using an algorithm like gradient descent to minimize that cost function. When we have a very large training set, gradient descent becomes a **computationally very expensive procedure**.

- So here we'll define a different way to optimize for large data sets which will allow us to scale the algorithms

Suppose we're training a linear regression model with gradient descent

- **Hypothesis:**
  - $h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$
- **Cost Function:**
  - $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

If we plot our two parameters θ_0 and θ_1 vs. the cost function $J(\theta_0, \theta_1)$ we get something like this (looks like a bowl shape surface plot):



Gradient descent looks like the following expression, where in the inner loop of gradient descent we repeatedly update the parameters theta using that expression:

Repeat {

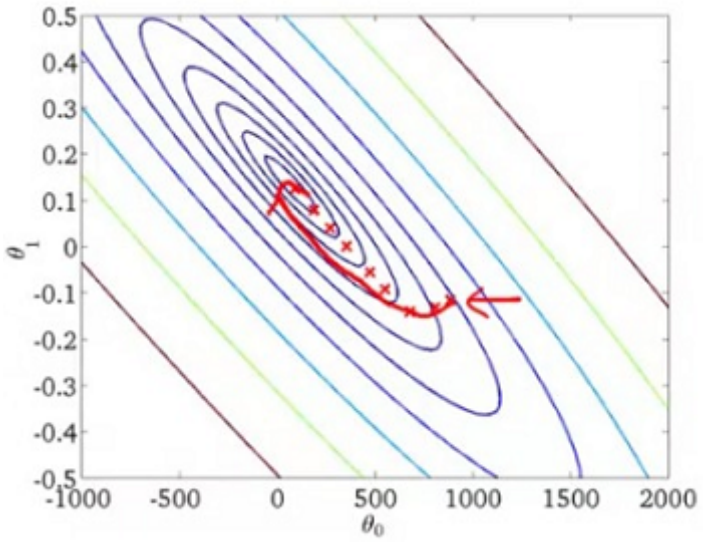
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

}

We will use linear regression for our algorithmic example here when talking about stochastic gradient descent, but the ideas of Stochastic gradient descent is fully general and also applies to other learning algorithms like **logistic regression, neural networks and other algorithms** that are based on **training gradient descent** on a specific training set.

Below we have a contour plot for gradient descent showing iteration to a global minimum, as we run gradient descent different iterations of gradient descent will take the parameters to the global minimum:



- As mentioned, if m is large gradient descent can be very expensive
  - Computing the derivative term can be very expensive, because we're summing over all m examples.
  - Although so far we just referred to it as gradient descent, this kind of gradient descent is called **batch gradient descent**
    - The term Batch refers to the fact that **we're looking at all of the training examples at a time**. We call it sort of a batch of all of the training examples.
- **Batch gradient descent is not great for huge datasets:**
  - If we have 300,000,000 records we need to **read in all the records** into memory from disk because we can't store them all in memory
  - By reading all the records, we can move one step (iteration) through the algorithm
  - **Then repeat for every step**
  - This means it take a long time to converge

In contrast to Batch gradient descent, what we are going to do is come up with a different algorithm that **doesn't need to look at all the training examples** in every single iteration, but that needs to look at only a **single training example** in one iteration.

Stochastic Gradient Descent

Stochastic gradient descent is an alternative to classic (or batch) gradient descent and is more efficient and scalable to large data sets.

- Stochastic gradient descent is written out in a different but similar way:

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

- So the function represents the cost of θ with respect to a specific example $(x^{(i)}, y^{(i)})$
- Measures how well the hypothesis works on a single example
- The only difference in the above cost function is the elimination of the m constant within $\frac{1}{2}$

The overall cost function can now be re-written in the following form:

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

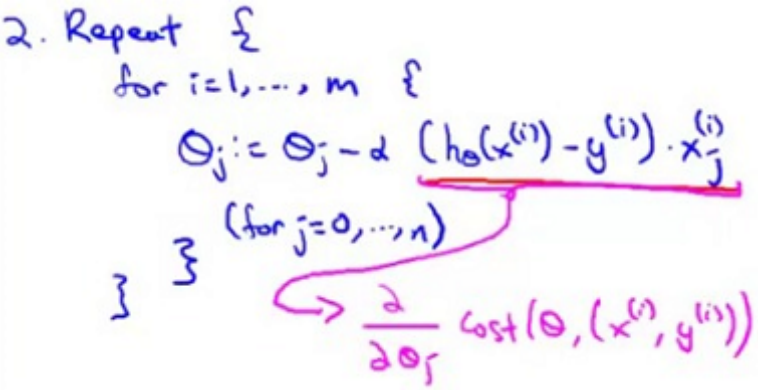
- J_{train} is now just the average of the cost applied to all of our training examples.
- **This is equivalent to the batch gradient descent cost function**

With this slightly modified (but equivalent) view of linear regression we can write out how stochastic gradient descent works:

1. Randomly 'shuffle' the dataset

- Randomly shuffle, or randomly reorder our m training examples.

2. Implement the gradient descent algorithm



What stochastic gradient descent algorithm is doing is scanning through each example:

- The inner for loop does something like this:
  - **Looking at example 1, take a step with respect to the cost of just the 1^{st} training example**
    - Having done this, we go on to the second training example
  - **Now take a second step in parameter space to try and fit the second training example better**
    - Now move onto the third training example, and so on...
  - **Until it gets to the end of the data**
    - We may now repeat this who procedure and take multiple passes over the data

The **randomly shuffling** at the start means we ensure the data is in a random order so we don't bias the movement

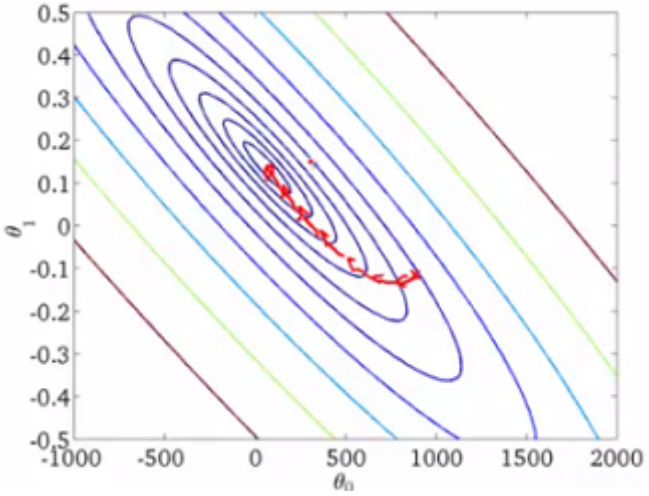
- Randomization should **speed up** convergence a little bit

Although stochastic gradient descent is a lot **like batch gradient descent**, rather than waiting to sum up the gradient terms over all m examples, we take just **one example** and **make progress** in improving the parameters already.

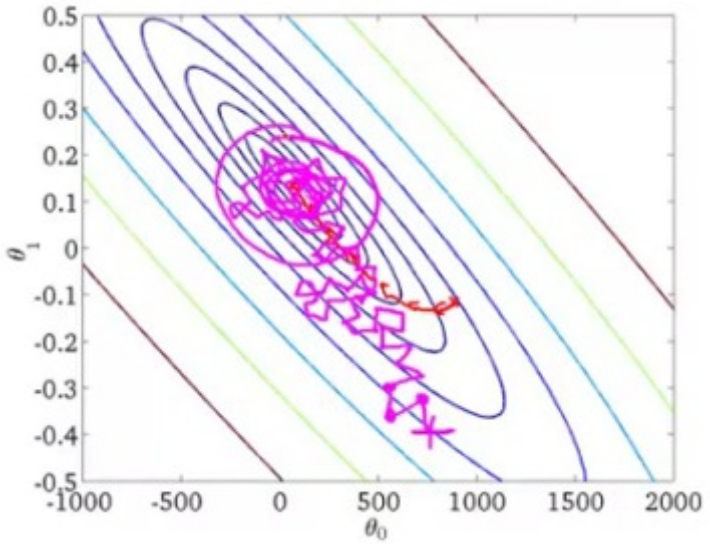
- Means we update the parameters on every step through data, instead of at the end of each loop through all the data

Stochastic Gradient Descent Contour plot

Previously, we saw that when we are using Batch gradient descent, that is the algorithm that looks at all the training examples in time, batch gradient descent will tend to take a reasonably straight line trajectory to get to the global minimum.



With stochastic gradient descent every iteration is much faster, but every iteration is flitting a single example



- What we find is that we "generally" **move in the direction of the global minimum**, but not always
- **Never actually converges** like batch gradient descent does, but ends up wandering around some region close to the global minimum.
- In practice, **this isn't a problem** - as parameters end up pretty close to the global minimum, that will be a pretty good hypothesis.
  - So usually running Stochastic gradient descent we get a parameter near the global minimum and that's good enough for the most practical purposes.

One final implementation note

- Stochastic Gradient Descent may need to loop over the entire dataset 1-10 times
- If you have a truly massive dataset it's possible that by the time we've taken a single pass through the dataset we may already have a perfectly good hypothesis
  - In which case the inner loop might only need to happen 1 if m is very very large
- **If we contrast this to batch gradient descent**
- We have to make k passes **through the entire dataset**!, where k is the number of steps needed to move through the data

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

> When the training set size m is very large, stochastic gradient descent can be much faster than gradient descent.

- 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).

> Before beginning the main loop of stochastic gradient descent, it is a good idea to "shuffle" your training data into a random order.