Stochastic Gradient Descent Convergence

How do we choose the **learning rate** α for stochastic gradient descent? Also, how do we **debug stochastic gradient descent** to make sure it is getting as close as possible to the global optimum?

Checking for convergence

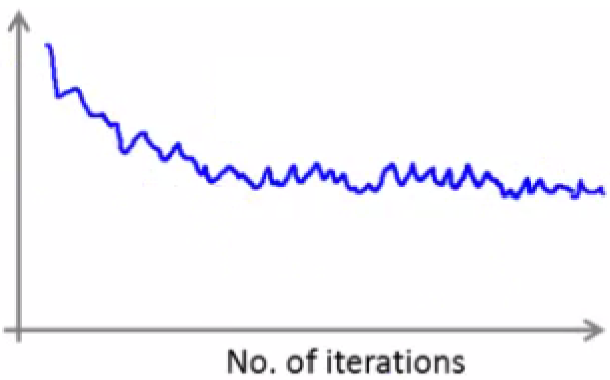
Back when we were using batch gradient descent, our standard way for making sure that gradient descent was converging was we would plot the **optimization cost function** as a function of the **number of iterations**.

- **Batch Gradient Descent:**
  - Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.
  - $J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$
    - $J_{train}(\theta)$ Should decrease on every iteration
  - This works when the training set size was small because we could sum over all examples m.
    - Doesn't work when we have a massive dataset (e.g. m = 300,000,000 records)

- **With stochastic gradient descent:**
  - We don't want to have to pause the algorithm periodically to do a summation over all data
  - Moreover, the whole point of stochastic gradient descent is to avoid those whole-data summations
  - **Take cost function definition:**
    - $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$
      - One half the squared error on a single example
    - **During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.**
      - i.e. we compute how well the hypothesis is working on the training example
      - Need to do this before we update θ because if we did it after θ was updated the algorithm would be performing a bit better (because we'd have just used $(x^{(i)}, y^{(i)})$ to improve θ).
- **Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.**
  - Gives a running estimate of how well we've done on the last 1000 estimates
  - By looking at the plots we should be able to check convergence is happening

Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples.

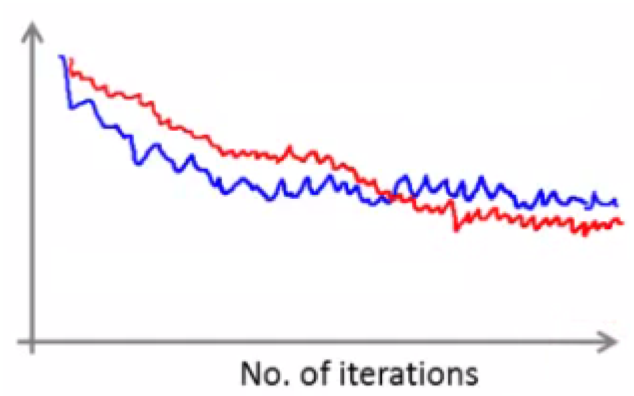
- Might be a bit noisy (1000 examples isn't that much)



If we get the previous figure, that would be a pretty decent run with the algorithm where it looks like the **cost has gone down** and then the plateau that looks kind of flattened out, that is what our cost function looks like then maybe our learning algorithm **has converged**.

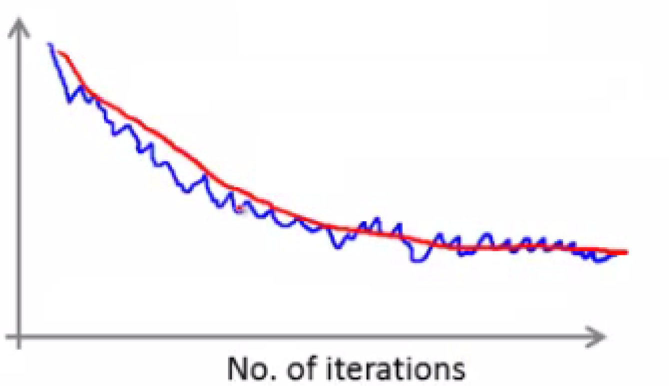
- **So concretely, the previous figure that's a pretty decent run and the algorithm may have convergence.**

If we use a smaller learning rate we may get an even better final solution



With a smaller learning rate, it is possible that we may get a slightly better solution with stochastic gradient descent. That is because stochastic gradient descent **will oscillate and jump around the global minimum**, and it will make smaller random jumps with a **smaller learning rate** (a smaller learning rate means smaller oscillations).

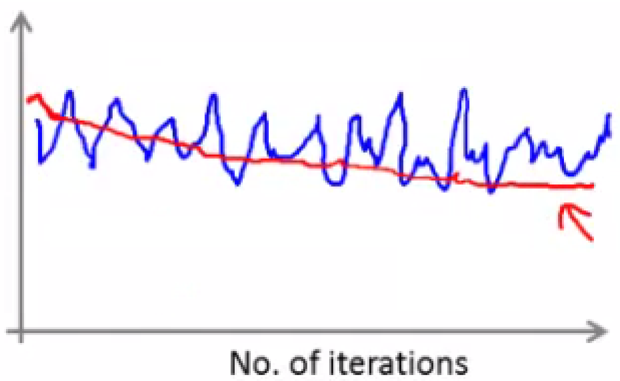
If we average over 1000 examples and 5000 examples we may get a smoother curve



If we increase the number of examples we average over to plot the performance of our algorithm, the plot's line will become smoother.

- This disadvantage of a larger average means we get less frequent feedback

Sometimes we may get a plot that looks like this:



With a very small number of examples for the average, the line will be too noisy and it will be difficult to find the trend.

- So the previous plot looks like cost is not decreasing at all
- But if we then increase to averaging over a larger number of examples we do see this general trend
  - Means the blue line was too noisy, and that noise is ironed out by taking a greater number of entires per average
- So the cost, it may not decrease, even with a large number

If we see a curve the looks like its increasing then the algorithm may be displaying divergence

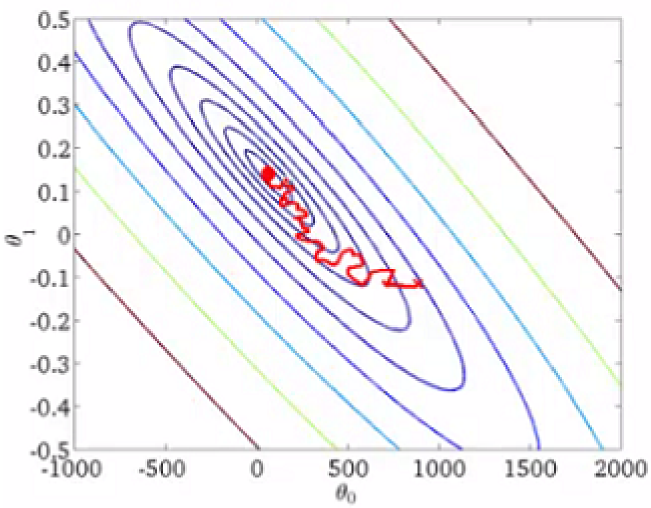
- We should use a smaller learning rate

Stochastic Gradient Descent: Learning rate α

In most typical implementations of stochastic gradient descent, **the learning rate α is typically held constant. One strategy for trying to actually converge at the global minimum is to slowly decrease α over time. For example α =** $\frac{const1}{iterationNumber + const2}$. Which means we're guaranteed to converge somewhere.

- We also need to determine const1 and const2

If we tune the parameters well, we can get something like this:



However, this is not often done because people don't want to have to fiddle with even more parameters.

Video 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.

> 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.

> If we plot $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ (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.