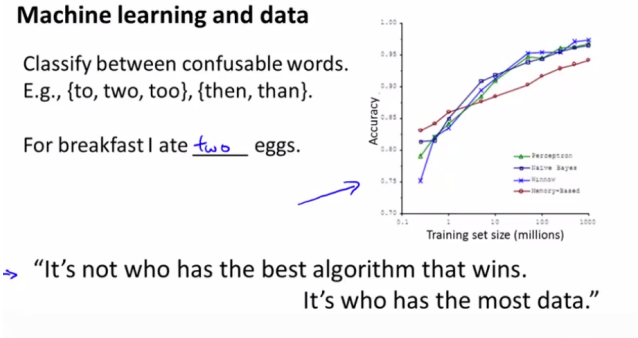
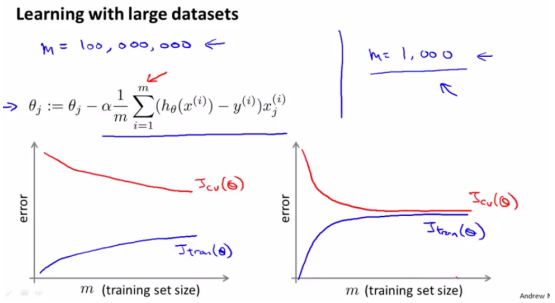
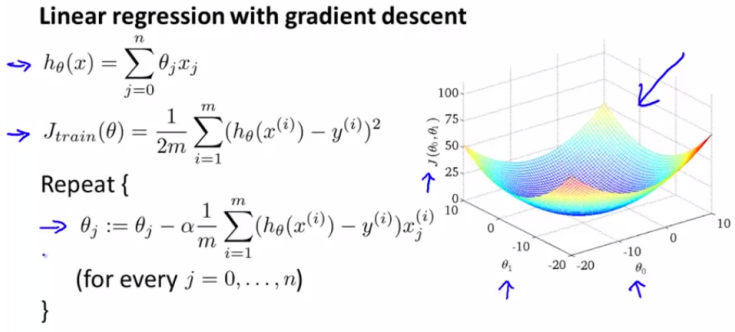
- **Gradient Descent with Large Datasets**
- **Learning With Large Datasets**
- Machine Learning and data



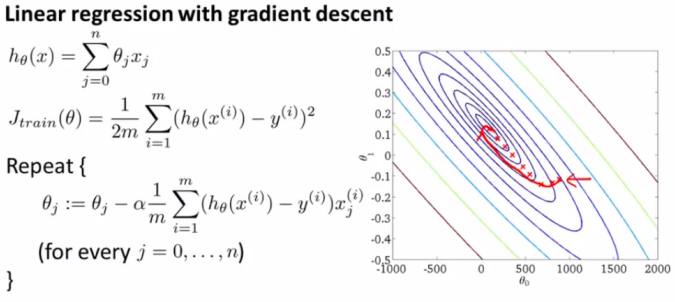
- Learning with large datasets



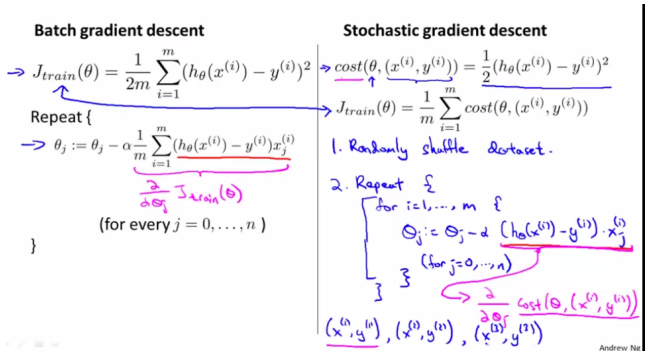
- **Stochastic gradient descent**
- Linear regression with gradient descent



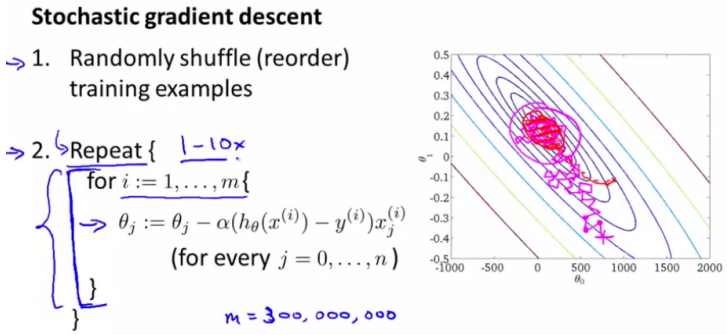
- If m is large computing the derivative term can be very expensive



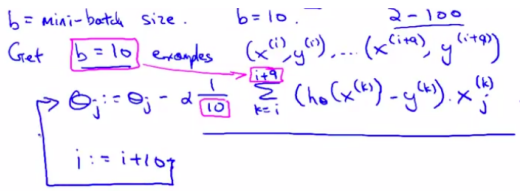
- Batch gradient this particular version
  - means that we are looking at all the training examples every time
- Batch gradient descent and Stochastic gradient descent



- stochastic gradient descent
  - doesn't actually find the global minimum stays in a region around the global minimum



- Mini-batch gradient descent
  - batch gradient descent: Use all m examples in each iteration
  - stochastic gradient descent: Use 1 example in each iteration
  - mini-batch gradient descent: Use b examples in each iteration
    - b = mini-batch size. b = 10 range b = 2 to 100 range
  - Get b = 10 examples from training set



- Mini-batch gradient descent

Mini-batch gradient descent

Say $b = 10, m = 1000.$

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

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

}

}

  - Vectorization
    - will output perform stochastic gradient descent if you have a good vectorized implementation
- **Stochastic Gradient Descent Convergence**
- Checking for convergence
  - Batch gradient descent
  - decreasing on every iteration

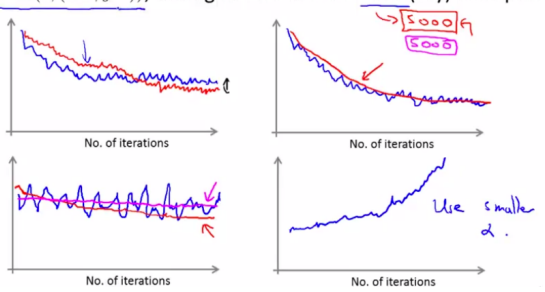
Checking for convergence

- Batch gradient descent:
  - Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.
  - $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
- Stochastic gradient descent:
  - $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$
  - During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.
  - Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.

- Checking for convergence

Checking for convergence

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



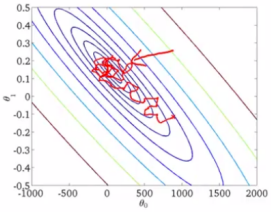
- Stochastic gradient descent

Stochastic gradient descent

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

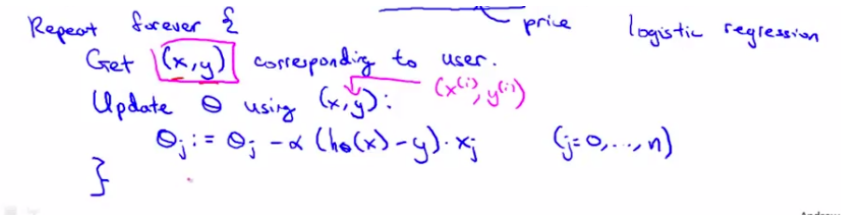
$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$

1. Randomly shuffle dataset.
2. Repeat {
   for $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }

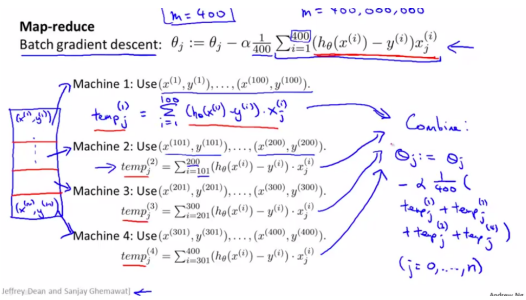


Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{const1}{iterationNumber + const2}$)

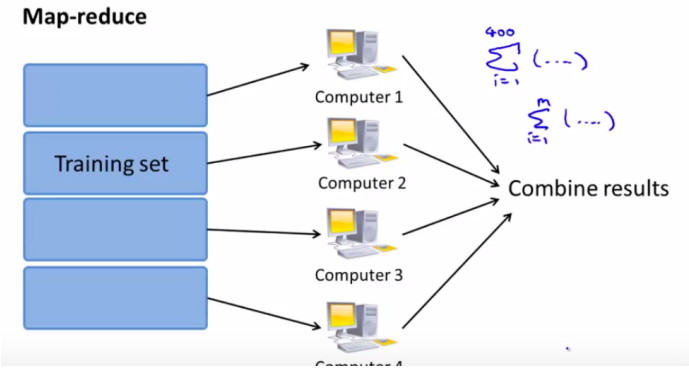
- Online Learning
  - Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service sometimes not. Y = 1 or 0
  - Features x captures properties of user, or origin/ destination and asking price. We want to learn p(y=1|x;theta) to optimize price.



  - can adapt to changing user preferences
- Other online learning example
  - Product search (learning to search)
    - User search 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.
  - y = 1 if user clicks on link. Y = 0 otherwise.
  - Learn p(y = 1| x; theta) <- predicts CTR
  - User to show user the 10 phones they’re most likely to click on
  - **get example learn from it then discard and move on**
- **Map Reduce and Data Parallelism**
- Map-reduce



- Map-reduce



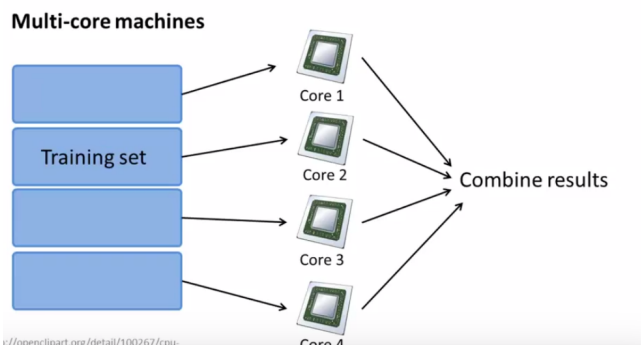
- Map-reduce and summation over the training set
  - many learning algorithms can be expressed as computing sums of functions over the training set

E.g. for advanced optimization, with logistic regression, need:

$$J_{train}(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) - (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))$$

$$\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

- Multi-core Machines



  - don't have to worry about network latency
-