

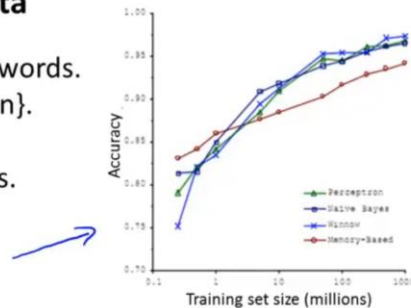
# Large scale machine learning

## 1 Learning with large datasets

### Machine learning and data

Classify between confusable words.  
E.g., {to, two, too}, {then, than}.

For breakfast I ate two eggs.



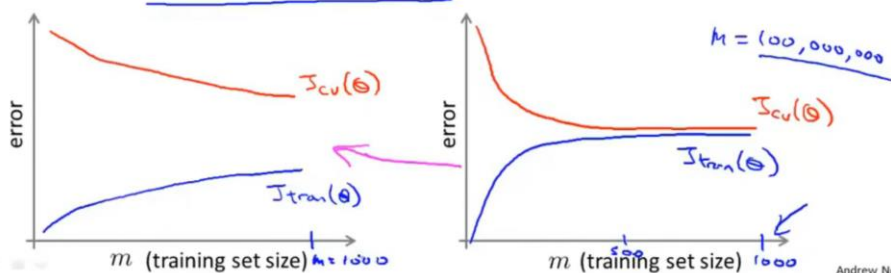
→ "It's not who has the best algorithm that wins.  
It's who has the most data."

### Learning with large datasets

$m = 100,000,000$  ←

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

$m = 1,000$  ←



Andrew Ng

## 2 Stochastic gradient descent

### Linear regression with gradient descent

$$h_{\theta}(x) = \sum_{j=0}^n \theta_j x_j$$

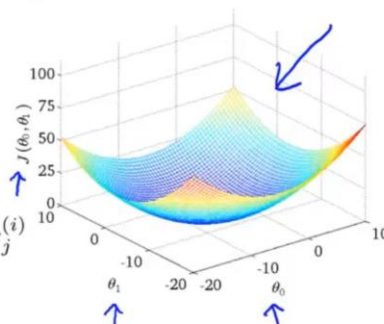
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}



## Linear regression with gradient descent

$$h_{\theta}(x) = \sum_{j=0}^n \theta_j x_j$$

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

Repeat {

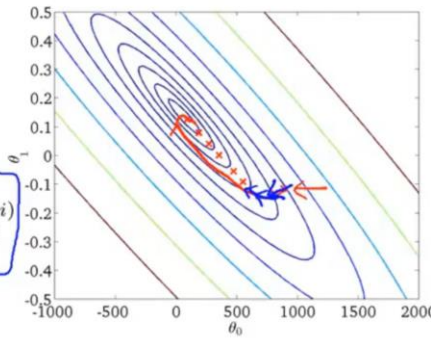
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

}

$m = 300,000,000$

Batch gradient descent



### Batch gradient descent

$$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

$\frac{\partial}{\partial \theta_j} J_{train}(\theta)$

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

}

$m = 300,000,000$

### Stochastic gradient descent

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

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

1. Randomly shuffle dataset. ←

2. Repeat {

for  $i = 1, \dots, m$  {

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

(for  $j = 0, \dots, n$ )

}

$$\rightarrow (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots$$

Andrew Ng

### Stochastic gradient descent

→ 1. Randomly shuffle (reorder) training examples

→ 2. Repeat { 1-10x

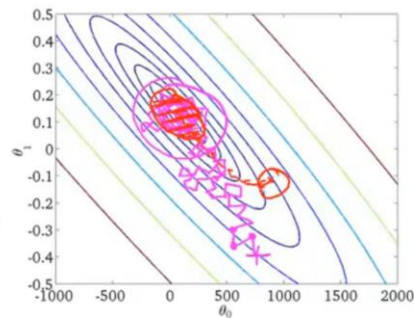
for  $i := 1, \dots, m$  {

$$\rightarrow \theta_j := \theta_j - \alpha (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

}

→  $m = 300,000,000$



## 3 Mini-batch gradient descent

### 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

$b = \text{mini-batch size} \quad b = 10$

Get  $b = 10$  examples  $(x^{(i)}, y^{(i)}), \dots, (x^{(i+9)}, y^{(i+9)})$

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

$j := i + 10$

### Mini-batch gradient descent

Say  $b = 10, m = 1000$ .

Repeat {

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

→  $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) x_j^{(k)}$   
(for every  $j = 0, \dots, n$ )

}

}

$M = 300,000,000$   
↑

→  $b$  examples

→ 1 example

Vectorization

$b = 10$   
↑

## 4 Stochastic gradient descent convergence

### Checking for convergence

→ Batch gradient descent:

→ Plot  $J_{\text{train}}(\theta)$  as a function of the number of iterations of gradient descent.

$$J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$M = 300,000,000$

→ Stochastic gradient descent:

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

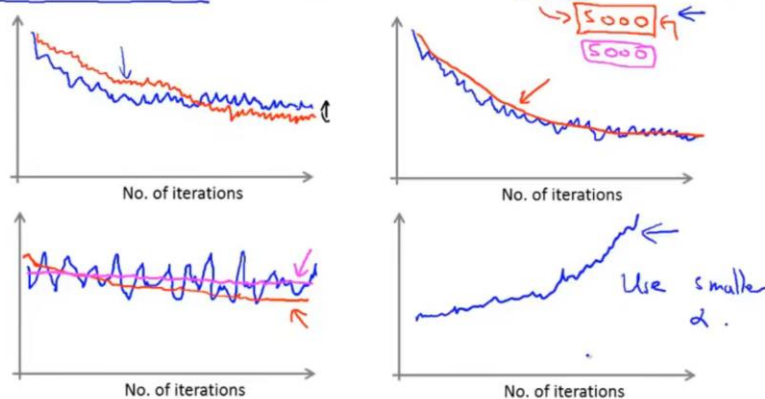
→ During learning, compute  $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$  before updating  $\theta$  using  $(x^{(i)}, y^{(i)})$ .

→  $(x^{(i)}, y^{(i)}), (x^{(i+1)}, y^{(i+1)})$

→ Every 1000 iterations (say), plot  $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$  averaged over the last 1000 examples processed by algorithm.

## Checking for convergence

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



Andrew Ng

## Stochastic gradient descent

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

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

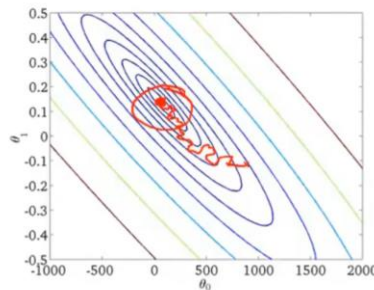
1. Randomly shuffle dataset.
2. Repeat {
 

for  $i := 1, \dots, m$  {
 

$\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j$   
 (for  $j = 0, \dots, n$ )

 }

 }



Learning rate  $\alpha$  is typically held constant. Can slowly decrease  $\alpha$  over time if we want  $\theta$  to converge. (E.g.  $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$ )  $\alpha \rightarrow 0$

Andrew Ng

## 5 Online learning

### 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.

Repeat forever {  
 Get  $(x, y)$  corresponding to user.  
 Update  $\theta$  using  $(x, y)$ :  
 $\rightarrow \theta_j := \theta_j - \alpha(h_{\theta}(x) - y) \cdot x_j$  ( $j = 0, \dots, n$ )  
 }  
 $\rightarrow$  Can adapt to changing user preference.

Andrew Ng



## Other online learning example:

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.

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

→ Learn  $p(y = 1|x; \theta)$ . ←

predicted CTR

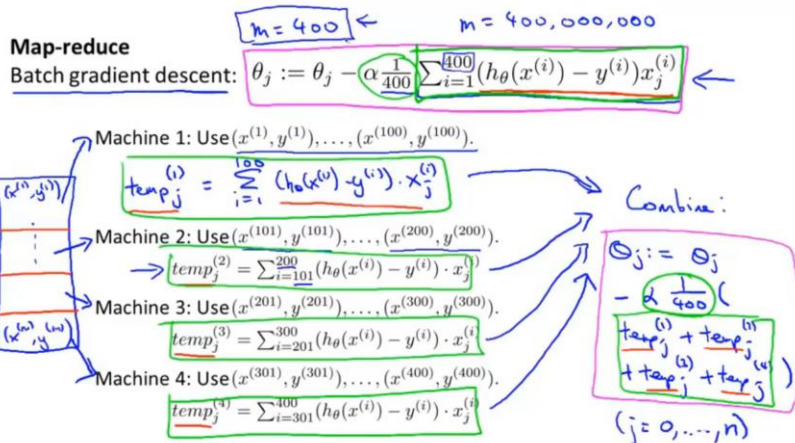
$(x, y)$  ←  
↑ ↑

→ 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; ...

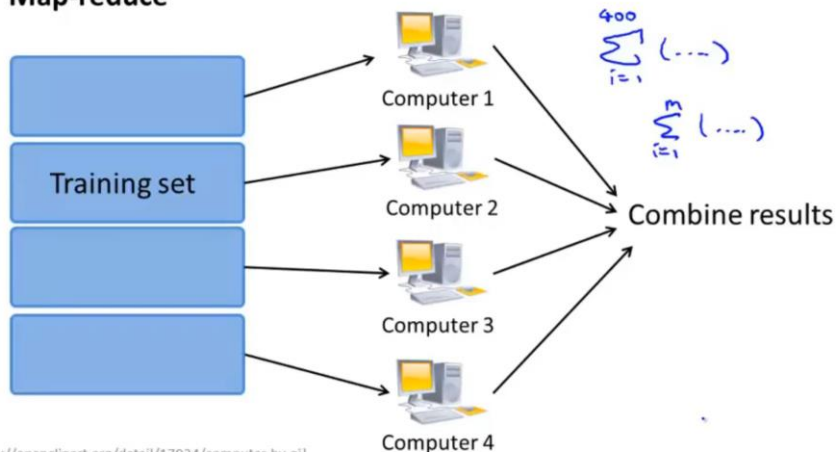
Andrew Ng

## 6 Map-reduce and data parallelism



Andrew Ng

### Map-reduce



Andrew Ng

## Map-reduce and summation over the training set

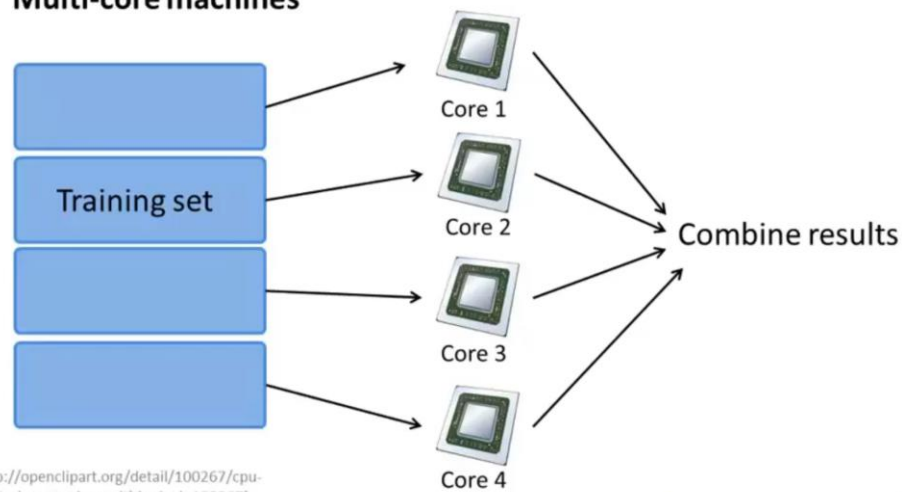
Many learning algorithms can be expressed as computing sums of functions over the training set.

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

$$\begin{aligned} \rightarrow J_{train}(\theta) &= -\frac{1}{m} \sum_{i=1}^m \underbrace{y^{(i)} \log h_{\theta}(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))}_{\text{temp}^{(i)}} \\ \rightarrow \frac{\partial}{\partial \theta_j} J_{train}(\theta) &= \frac{1}{m} \sum_{i=1}^m \underbrace{(h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}}_{\text{temp}_j^{(i)}} \end{aligned}$$

Andrew Ng

## Multi-core machines



[<http://openclipart.org/detail/100267/cpu-central-processing-unit-by-ivak-100267>]

Andrew Ng