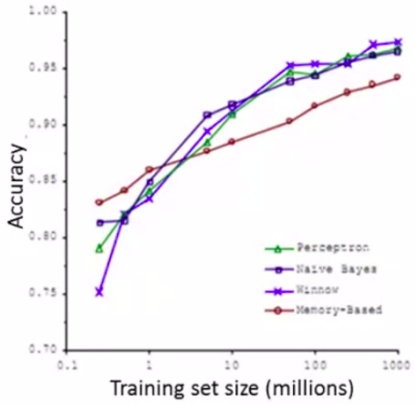
# Week 10 - Lecture 1

## 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."

[Figure from Banko and Brill, 2001]

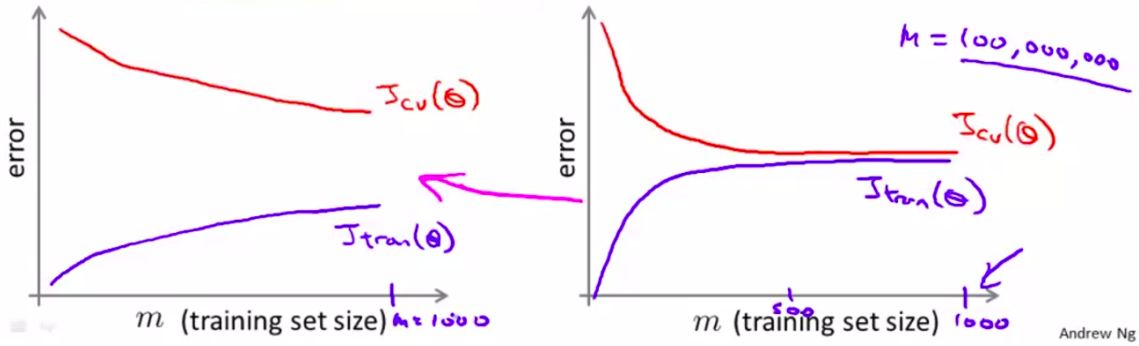
Andrew Ng

### 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

## Stochastic Gradient Descent

Remember the **Gradient Descent Algorithm**:

## 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 {

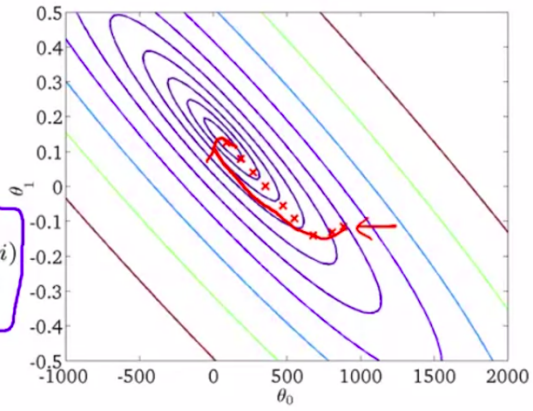
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

}

$m = 300{,}000{,}000$

Batch gradient descent




Andrew Ng

## Batch gradient descent

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

Repeat {

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

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

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

}

## Stochastic gradient descent

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

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

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

$\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$

Andrew Ng

**Stochastic gradient descent**

1. Randomly shuffle (reorder) training examples

2. Repeat { 1-10×

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

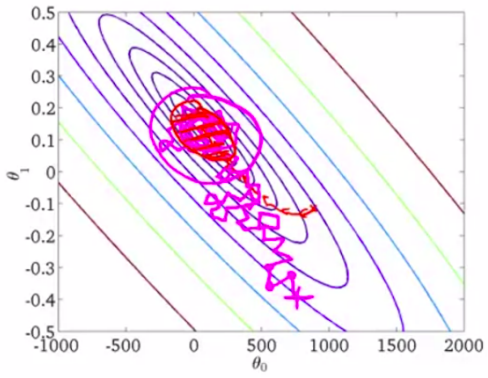
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

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

   }

}

$m = 300,000,000$




Andrew Ng

## 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

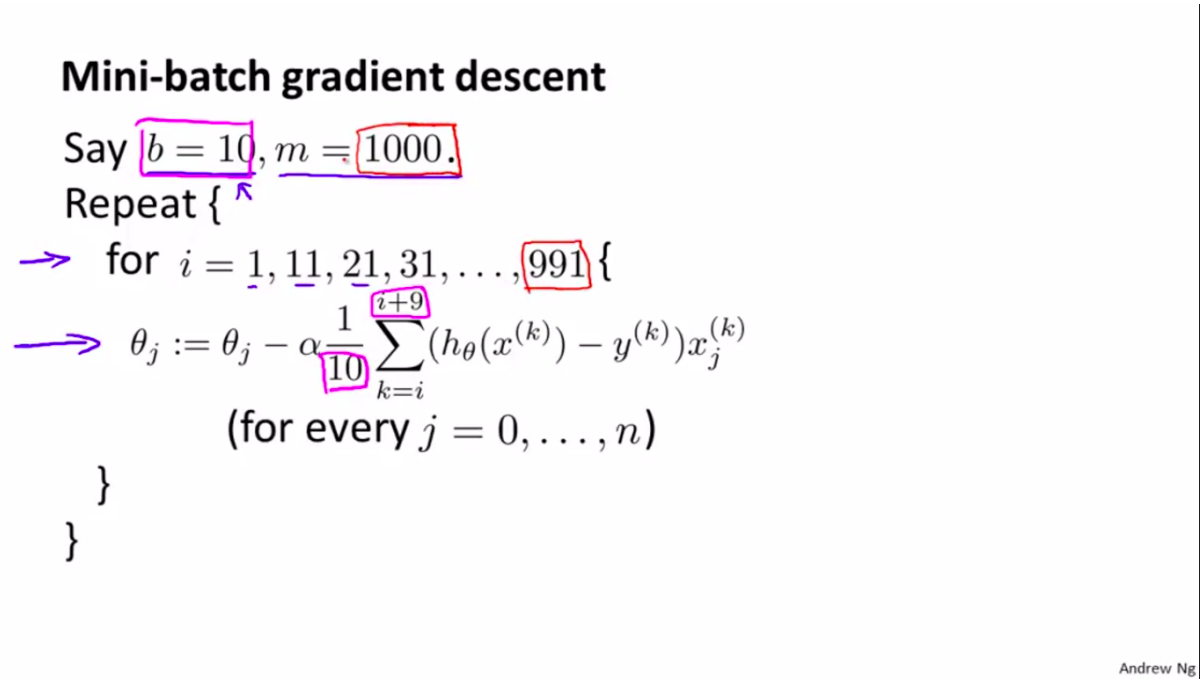
Mini-batch gradient descent: Use $b$ examples in each iteration

$b$ = mini-batch size. $b = 10$. 2-100

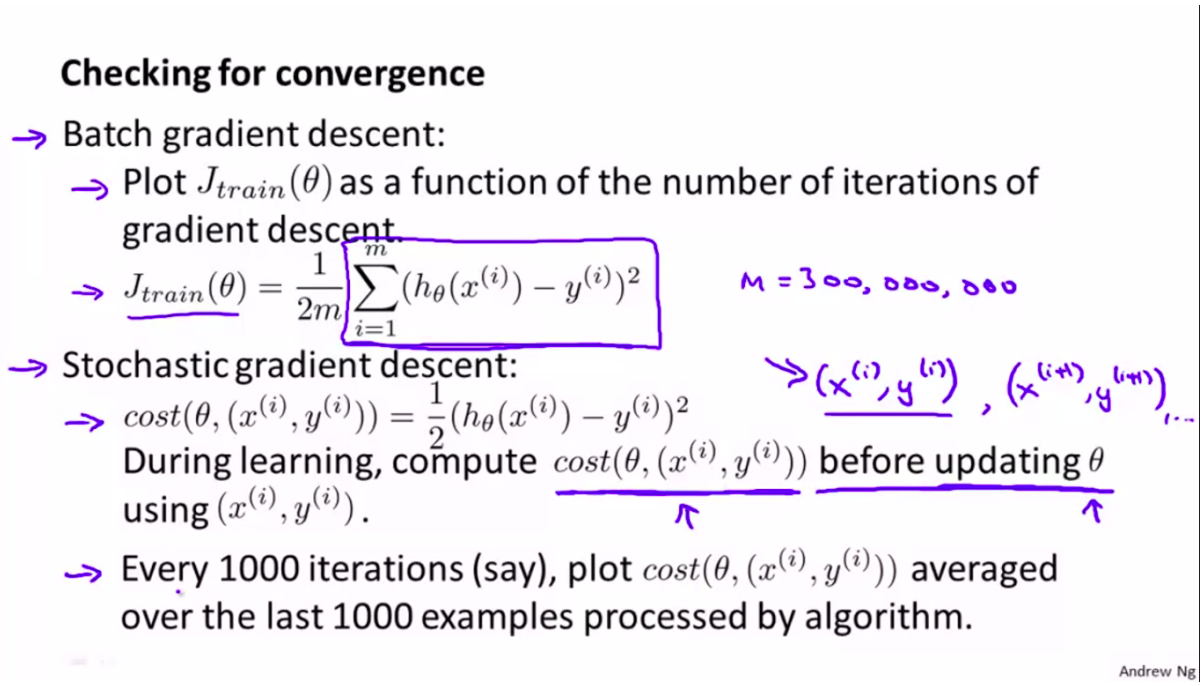
Get $b = 10$ examples $(x^{(i)}, y^{(i)}), \ldots (x^{(i+9)}, y^{(i+9)})$

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

Andrew Ng



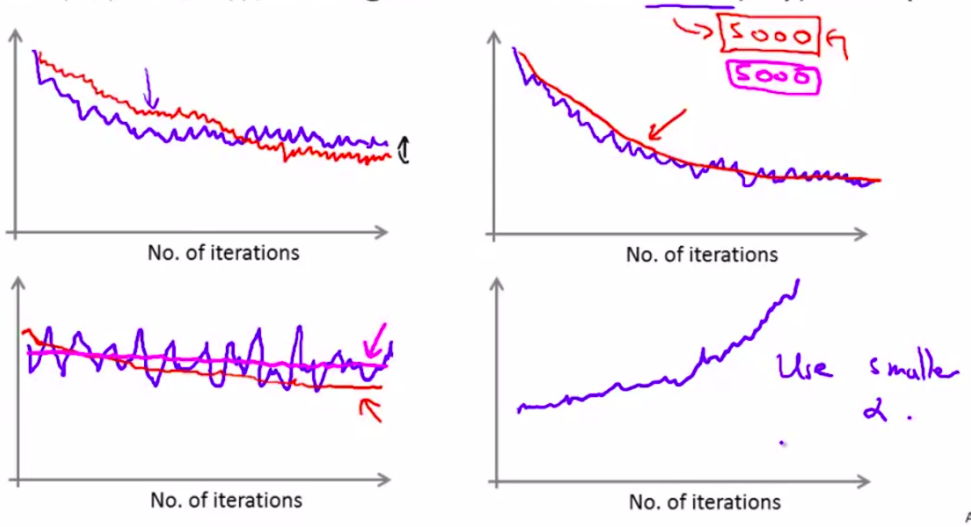

## Stochastic Gradient Descent Convergence




- **upper left:** Stochastic Gradient Descent convergence with different learning rates (red line has a smaller learning rate).
- **upper right:** Convergence with different number of examples to calculate the mean of the cost function (the red line represents the curve with a higher number of examples).
- **bottom left:** looks like the cost is not decreasing, i.e., the algorithm is not learning (the blue line is too noisy). The red line represents the same curve, just by looking at a higher number of examples. This migh indicate that the algorithm is learning, just at a small convergence rate. If the curve remains flat, it's a stronger indicator that the algorithm isn't learning.
- **bottom right:** the algorithm is diverging. So, to try to remedy this, we should try using a smaller value of $\alpha$.

## Checking for convergence

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




Andrew Ng

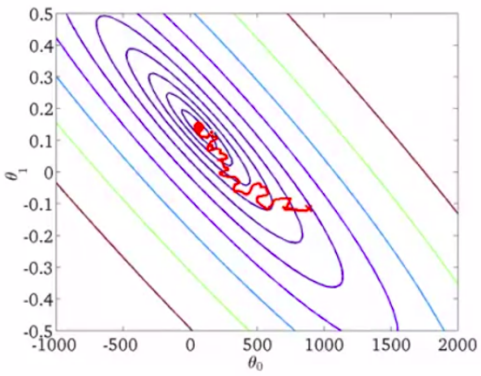
**Note:** If we want stochastic gradient descent to really converge to the global minimum, we can try decreasing he value of $\alpha$ over time.

## 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 $j = 0, \ldots, n$)
   }
   }

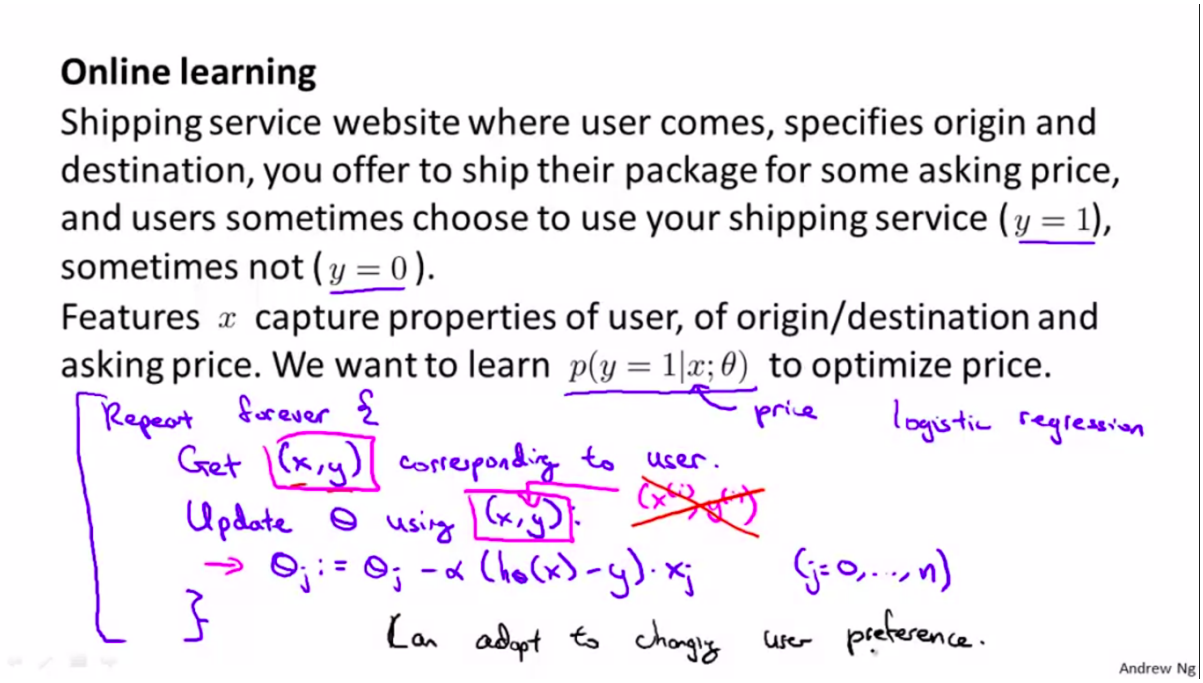



Learning rate $\alpha$ is typically held constant. Can slowly decrease $\alpha$ over time if we want $\theta$ to converge. (E.g. $\alpha = \frac{\texttt{const1}}{\texttt{iterationNumber + const2}}$ )

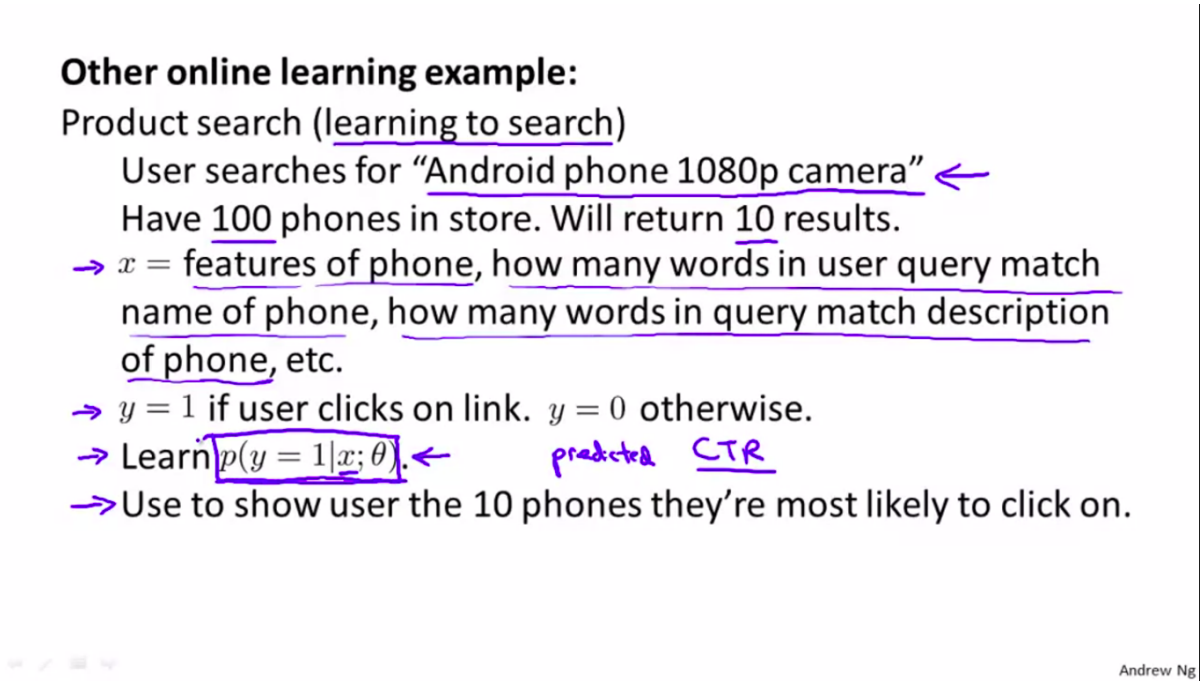
Andrew Ng

# Advanced Topics

## Online Learning



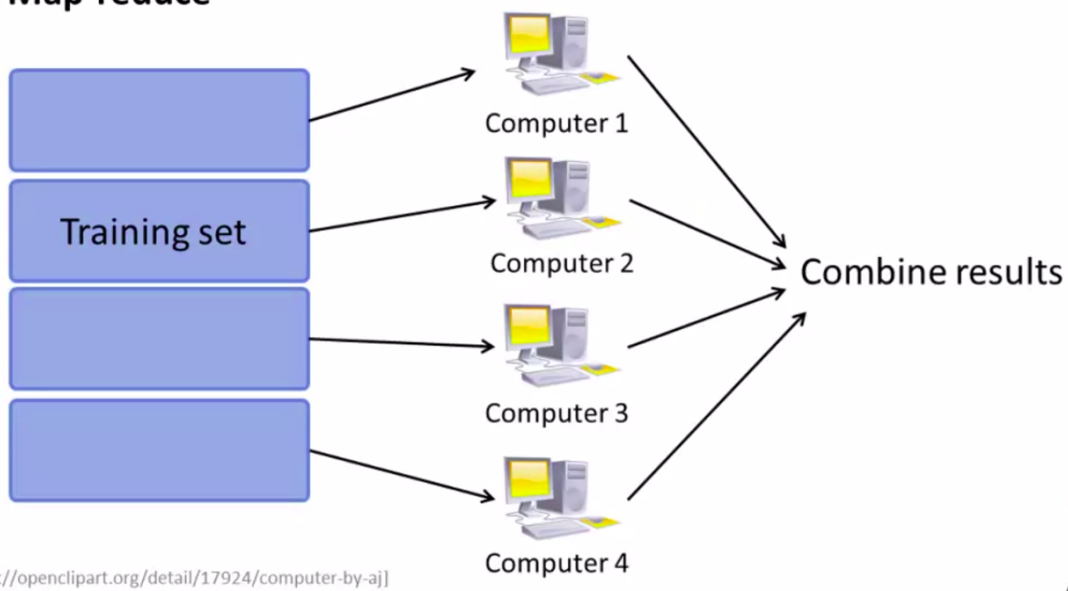

**Example:** Predict the *Click Through Rate* (map the clicks of users accessing a website)




**Other examples:** choosing special offers to show the user; customized selection of news articles; product recommendation; etc.

## Map Reduce and Data Parallelism

## Map-reduce




[http://openclipart.org/detail/17924/computer-by-aj]

Andrew Ng

## Map-reduce

Batch gradient descent: $\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

$m = 400$ $\quad m = 400{,}000{,}000$

Machine 1: Use $(x^{(1)}, y^{(1)}), \ldots, (x^{(100)}, y^{(100)})$.

$temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Machine 2: Use $(x^{(101)}, y^{(101)}), \ldots, (x^{(200)}, y^{(200)})$.

$temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Machine 3: Use $(x^{(201)}, y^{(201)}), \ldots, (x^{(300)}, y^{(300)})$.

$temp_j^{(3)} = \sum_{i=201}^{300} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Machine 4: Use $(x^{(301)}, y^{(301)}), \ldots, (x^{(400)}, y^{(400)})$.

$temp_j^{(4)} = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Combine:

$\theta_j := \theta_j - \alpha \frac{1}{400} (temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$

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

[ Jeffrey Dean and Sanjay Ghemawat]

Andrew Ng

## Map-reduce and summation over the training set

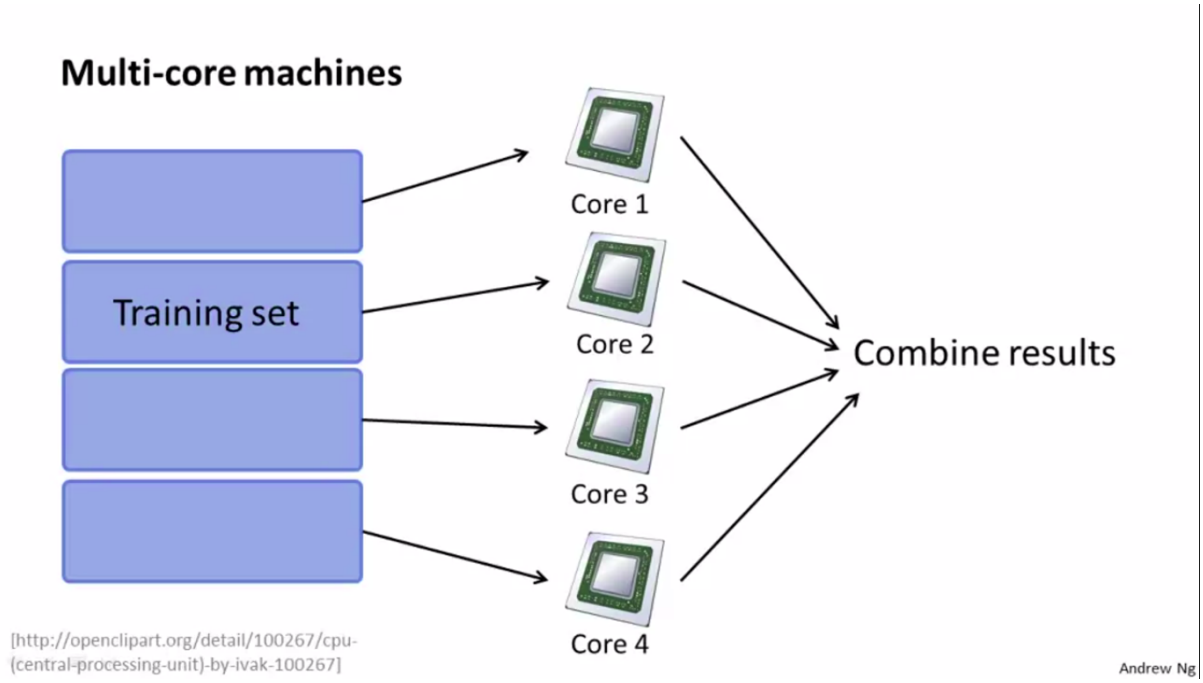
Many learning algorithms can be expressed as computing sums of functions over the training set.

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

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

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

Andrew Ng




## References

[1] Machine Learning - Stanford University (https://www.coursera.org/learn/machine-learning)