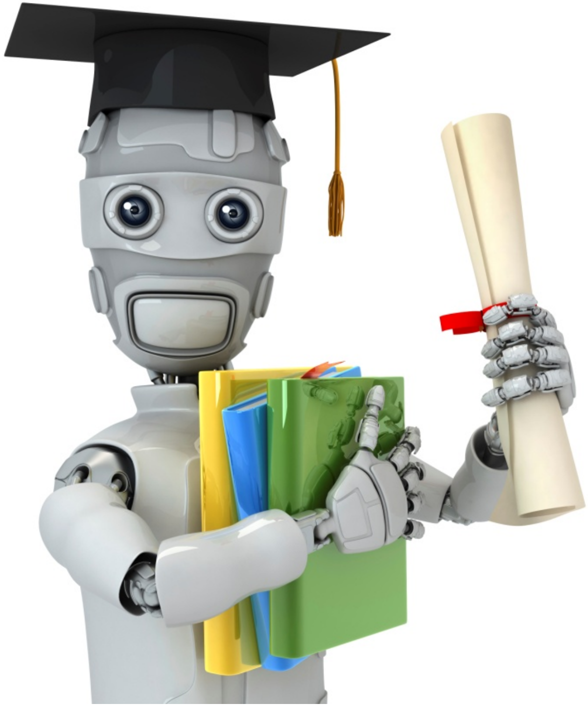


Machine Learning

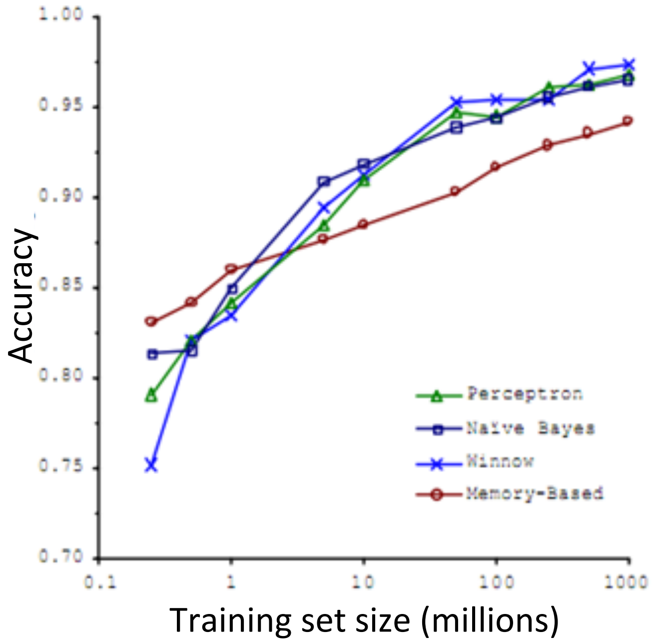
Large scale machine learning

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."

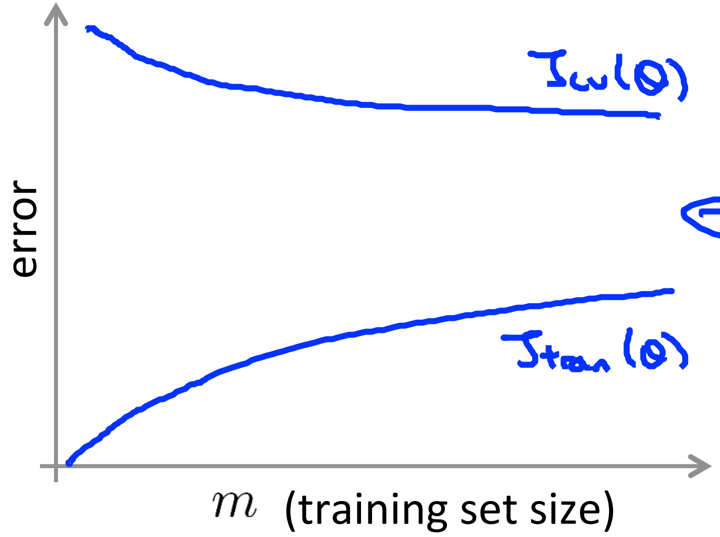
[Figure from Banko and Brill, 2001]

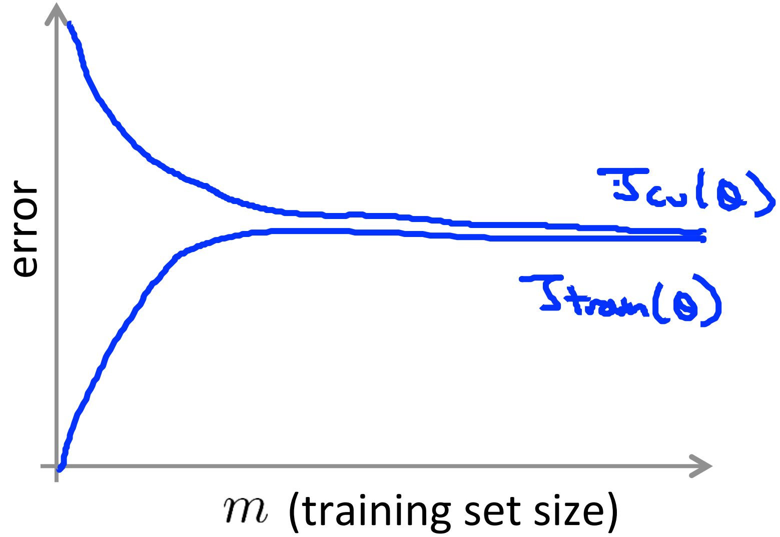
Learning with large datasets

$m = 100{,}000{,}000$

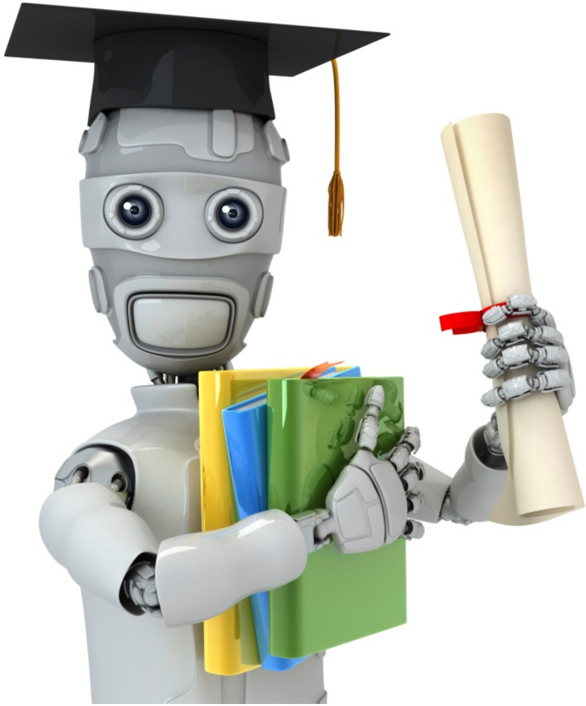
$m = 1{,}000?$

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





Andrew Ng



Machine Learning

Large scale machine learning

Stochastic gradient descent

Linear regression with gradient descent

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

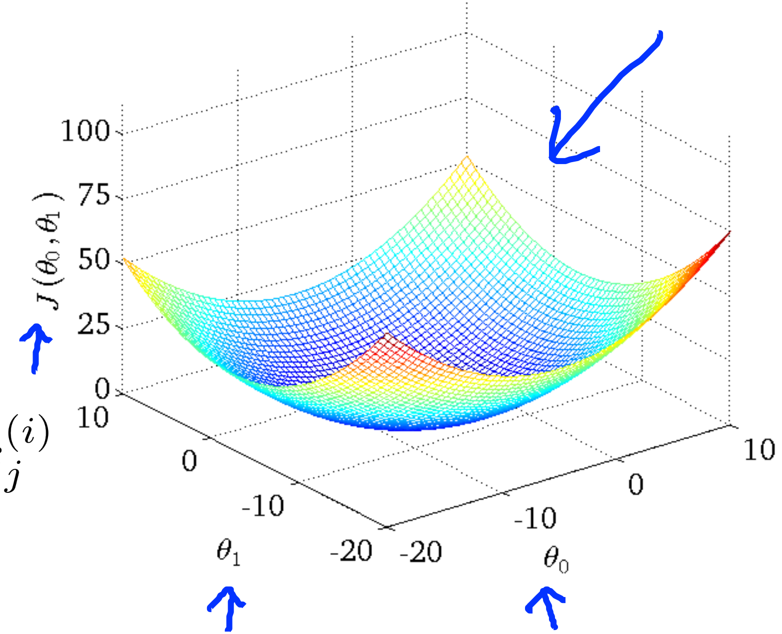
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}



Andrew Ng

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 {

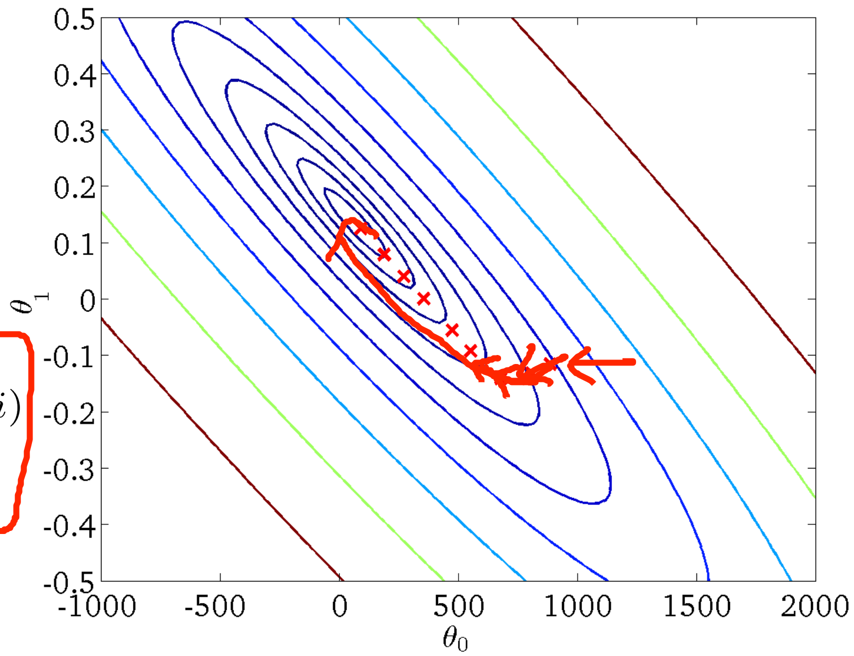
$$\theta_j := \theta_j - \alpha \boxed{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}}$$

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

}

M = 300,000,000

Batch gradient descent



Batch gradient descent

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

Repeat {

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

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

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

}

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

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 $i = 1, \ldots, m$ {

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

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

}

}

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

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

Stochastic gradient descent

1. Randomly shuffle (reorder) training examples

2. Repeat { 1-10x

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

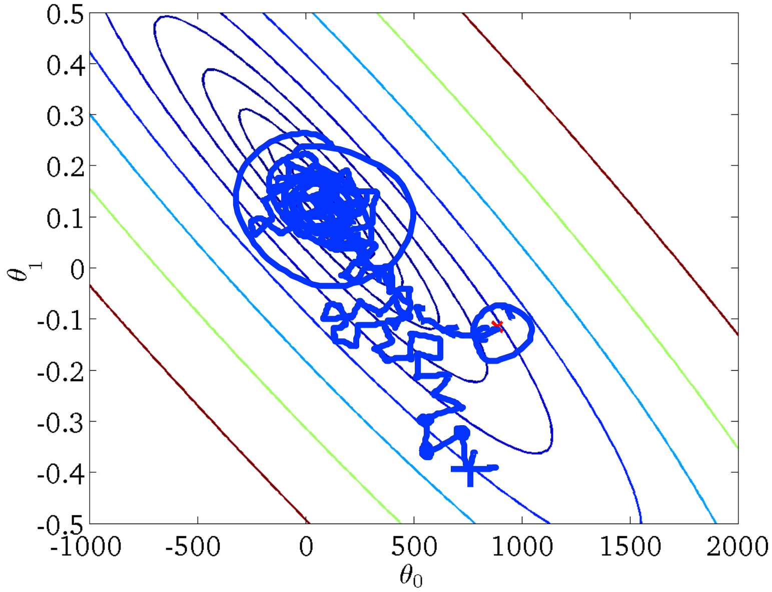
   $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

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

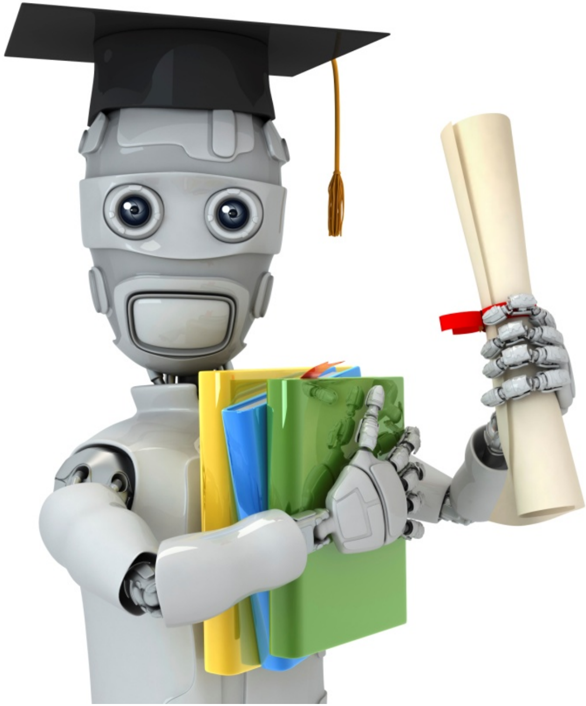
   }

   }

m = 300,000,000



Andrew Ng



Machine Learning

Large scale machine learning

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

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

$i := i + 10;$

Mini-batch gradient descent

Say $\boxed{b = 10}$, $m = \boxed{1000}$.

Repeat {

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

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

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

}

}

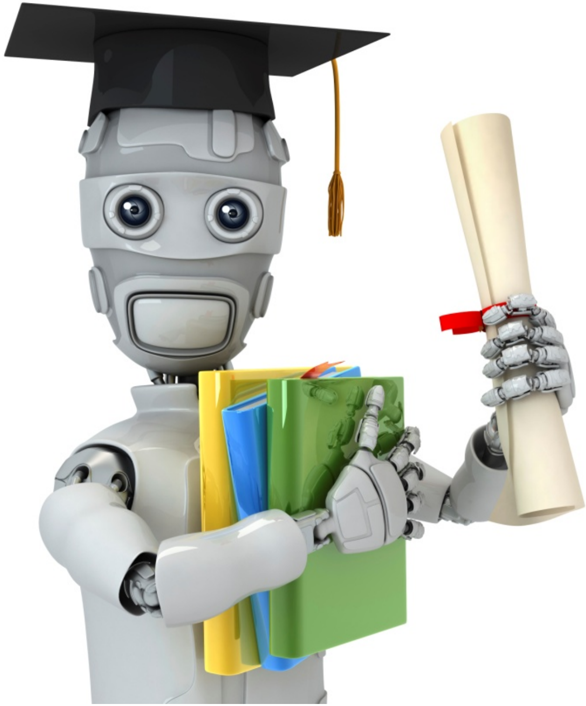
→ b examples

→ 1 example

Vectorization

m = 300,000,000

b = 10



Machine Learning

Large scale machine learning

Stochastic gradient descent convergence

Checking for convergence

- Batch gradient descent:
  - Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.
  - $\boxed{J_{train}(\theta)} = \frac{1}{2m}\boxed{\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2}$

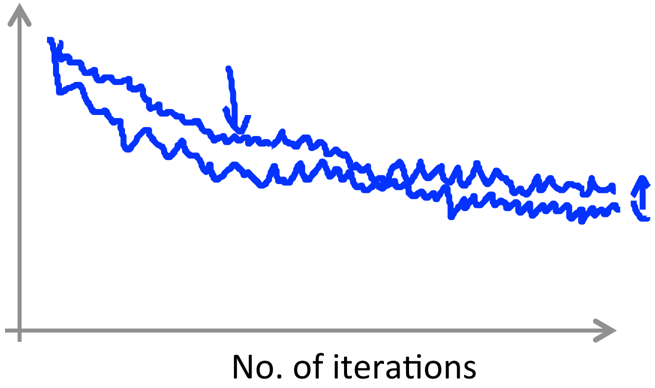
M = 300,000,000

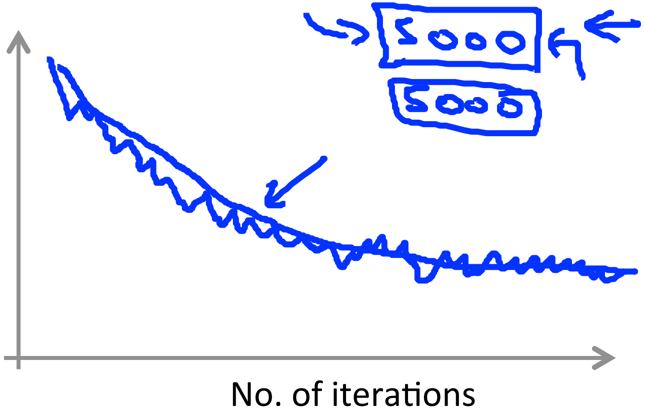
- Stochastic gradient descent:
  - $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$
  - During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.
  - Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged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

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

Checking for convergence

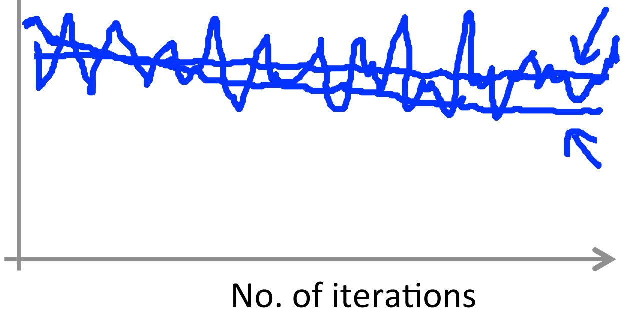
Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples

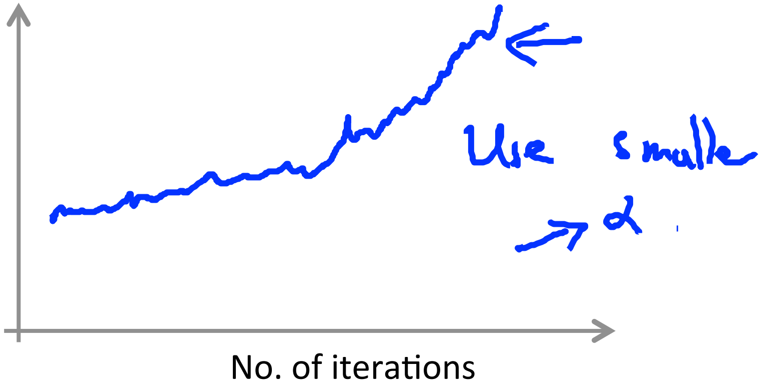




first three: average over more samples to lower the variance of cost, a.k.a., increase steps per epoch
last: lower the learning rate

Increasing 1000 to 5000 may let you see the trends of the cost





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 {

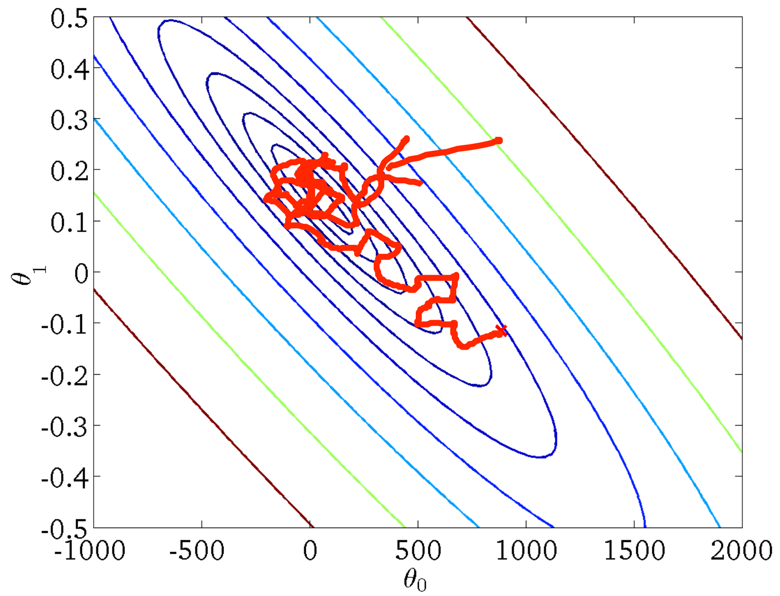
    for $i := 1, \ldots, m$ {

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

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

    }

}



If you don't change alpha, it justs oscillate around the minimum.
Lower the learning rate while training may give you more closer point to the minumum

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

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 {

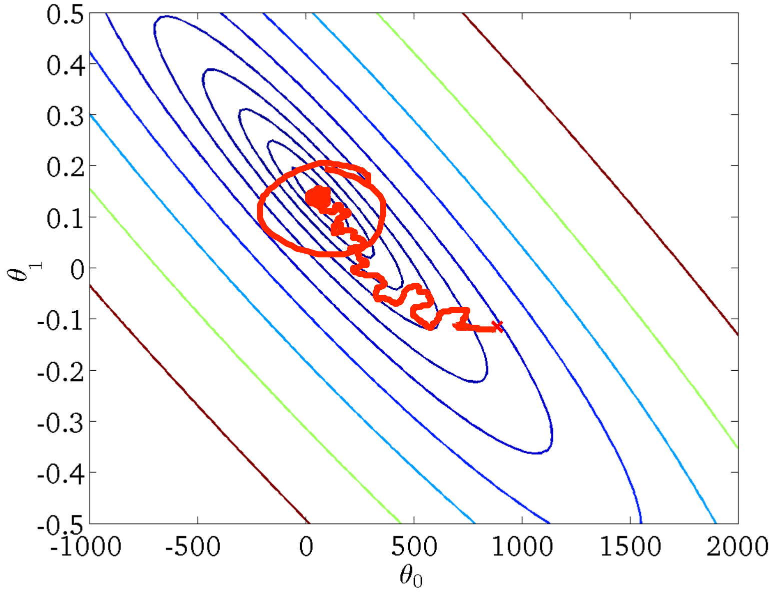
    for $i := 1, \ldots, m$ {

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

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

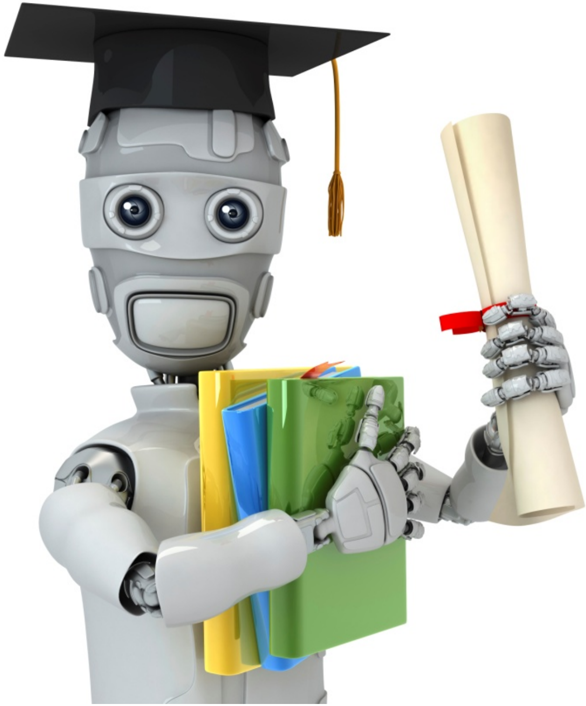
    }

}



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

Andrew Ng



Machine Learning

Large scale machine learning

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.

price

logistic regression

Repeat forever {

Get $\boxed{(x, y)}$ corresponding to user.

Update θ using $\boxed{(x,y)}$: ~~$(x^{(i)}, y^{(i)})$~~

$\rightarrow \theta_j := \theta_j - \alpha (h_\theta(x) - y) \cdot x_j \quad (j = 0, \ldots, n)$

}

$\rightarrow$ Can adapt to changing user preference.

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.

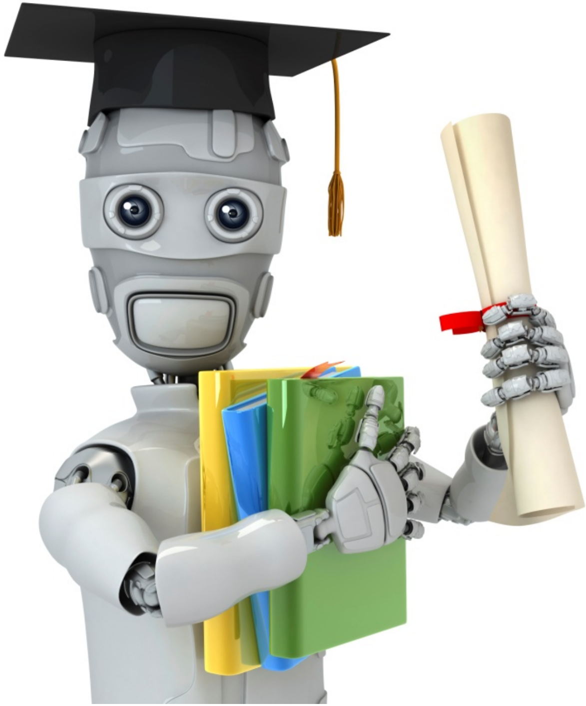
(x, y)

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

Learn $\boxed{p(y = 1|x; \theta)}$. predicted CTR

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; ...



Machine Learning

Large scale machine learning

Map-reduce and data parallelism

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

Map-reduce

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

$(x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})$

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 $(x^{(301)}, y^{(301)}), \ldots, (x^{(400)}, y^{(400)})$.

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

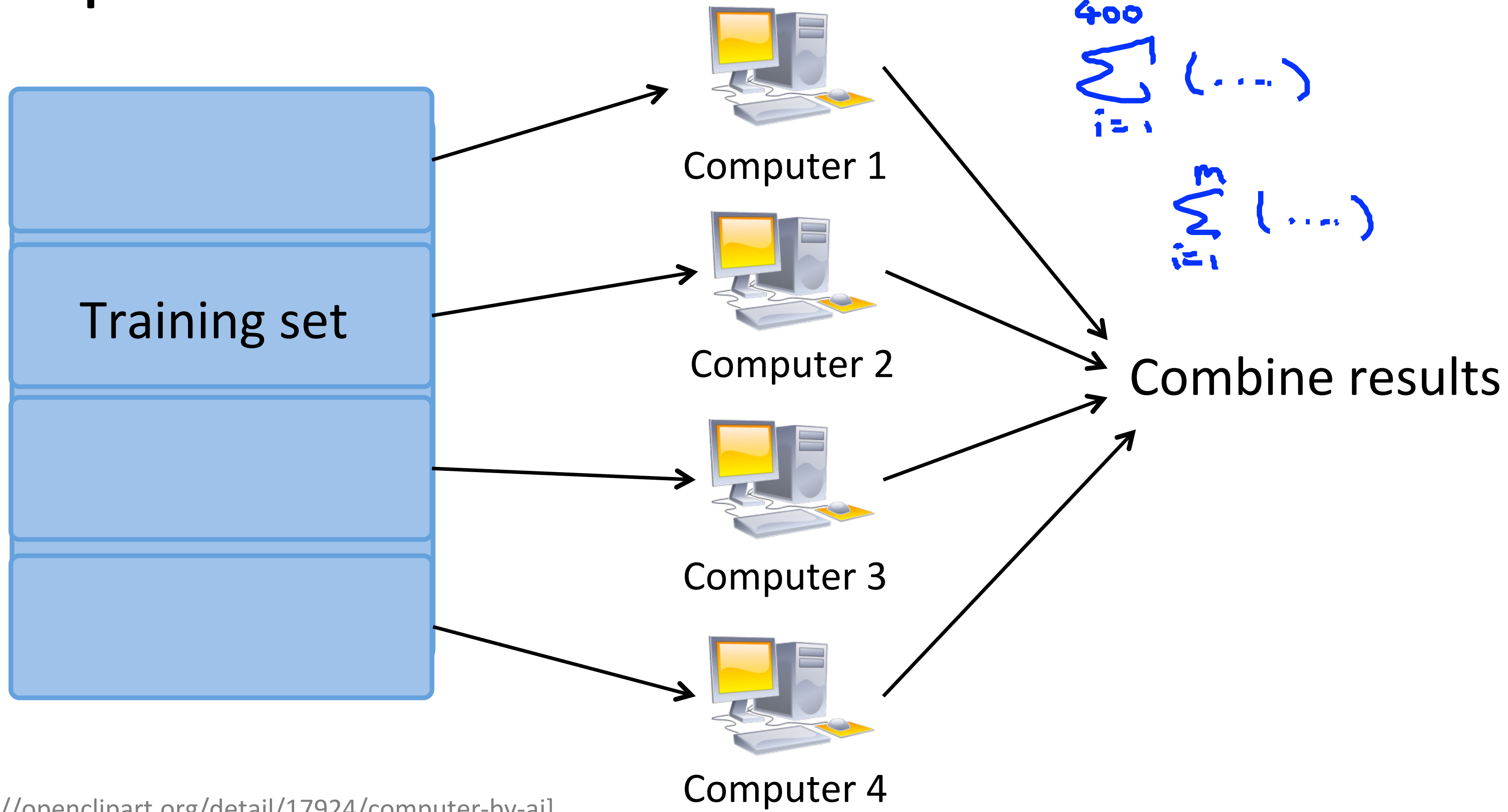
Combine:

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

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

[Jeffrey Dean and Sanjay Ghemawat]

Map-reduce



$\sum_{i=1}^{400} (\ldots)$

$\sum_{i=1}^{m} (\ldots)$

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

Map-reduce and summation over the training set

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

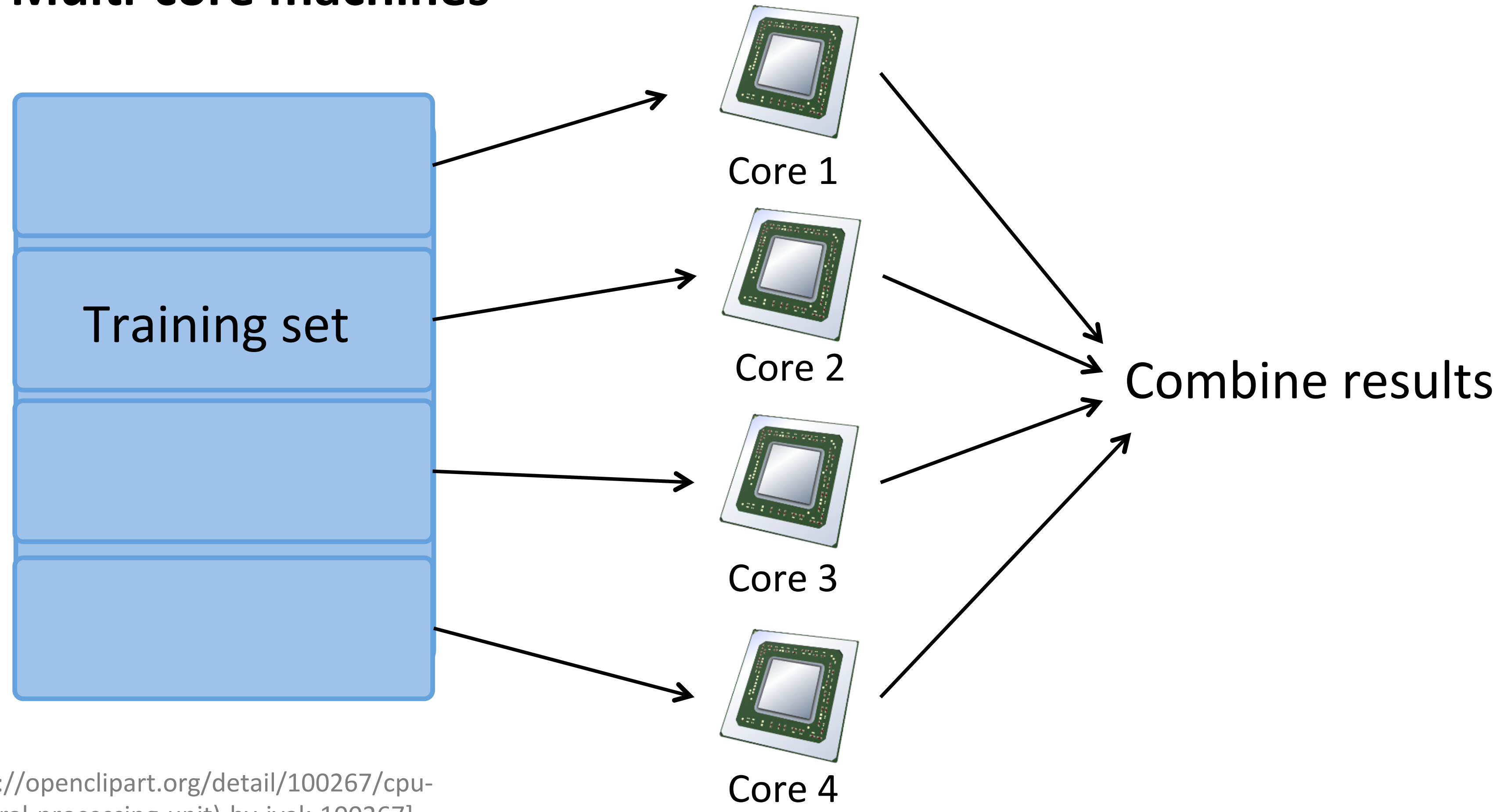
E.g. for advanced optimization, with logistic regression, need:

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

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

$temp^{(i)}$ $temp_j^{(i)}$

Multi-core machines



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