



Machine Learning

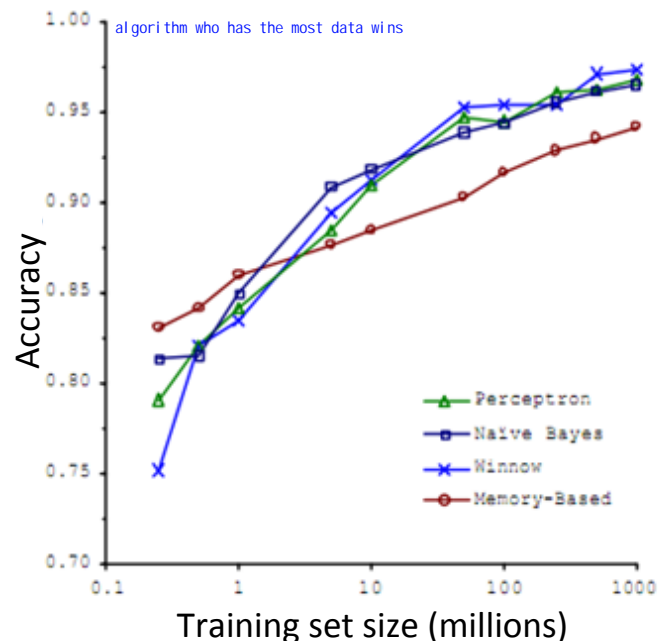
Large scale machine learning

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.**”

Learning with large datasets

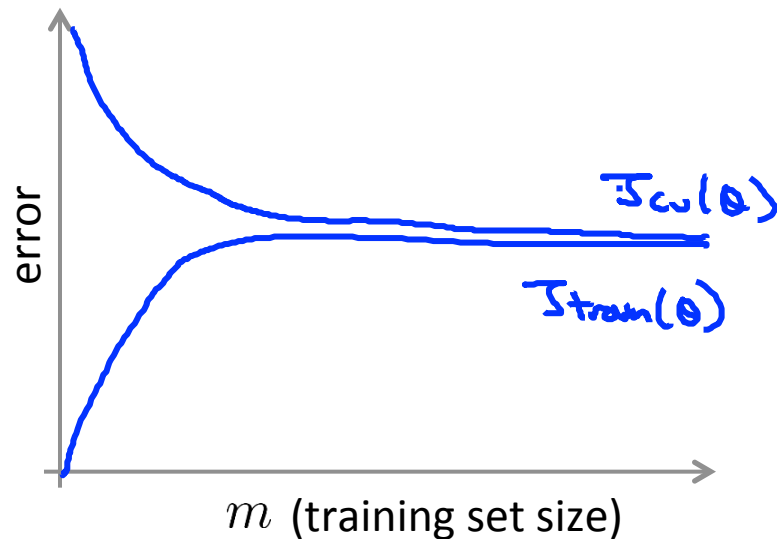
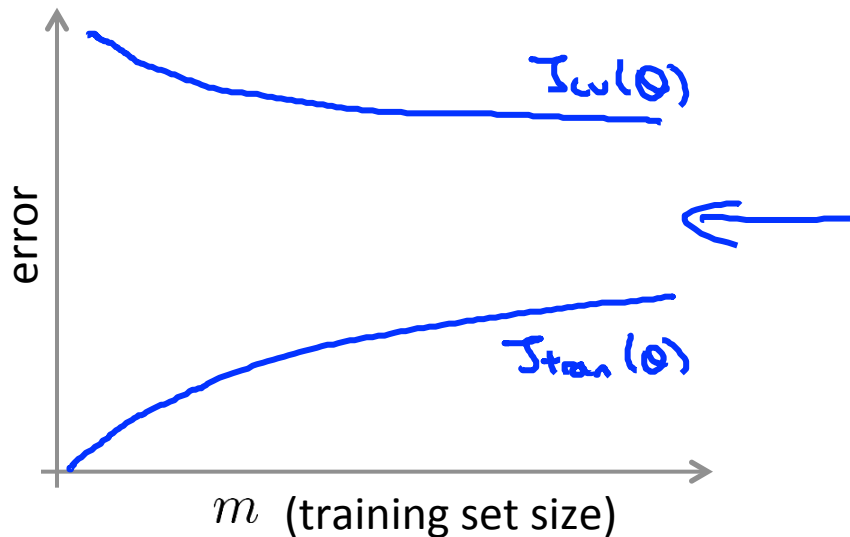
$m = 100,000,000$

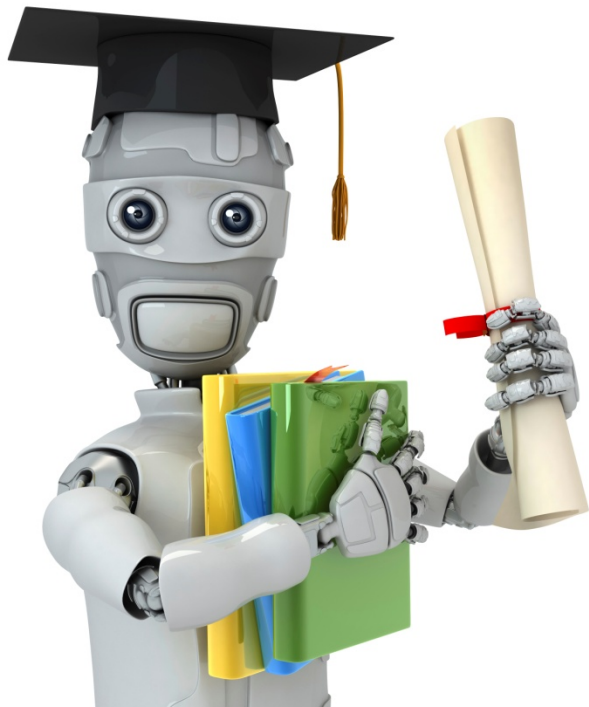
$m = 1,000?$

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

Need a lot computation power!

We would like to get some computational efficient ways to speed up your learning algorithm!





Machine Learning

Large scale machine learning

Stochastic
gradient descent

regular gradient descent is computation expensive!

Linear regression with gradient descent

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

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

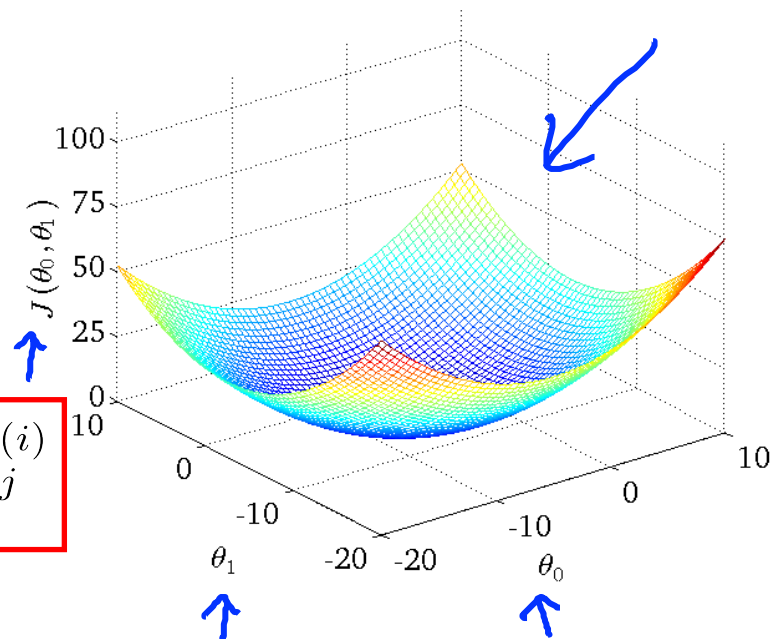
Repeat {

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

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

}

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 {

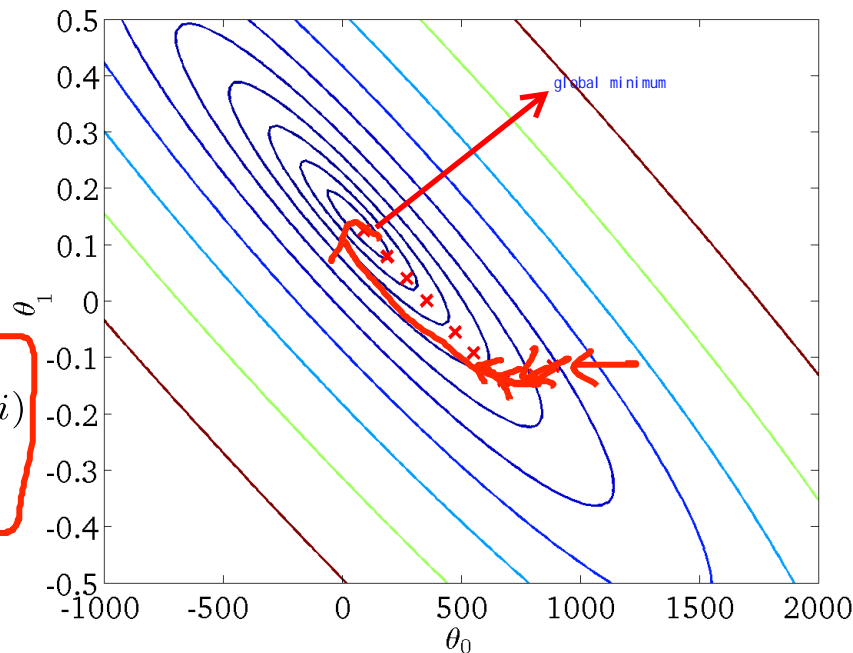
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

}

$M = 300,000,000$

Batch gradient descent



a different name

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

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

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

1. Randomly shuffle dataset. ← first step

2. Repeat {

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

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

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

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

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

just look at a single training example at a time

Stochastic gradient descent

- 1. Randomly shuffle (reorder) training examples

→ 2. Repeat { 1-10x

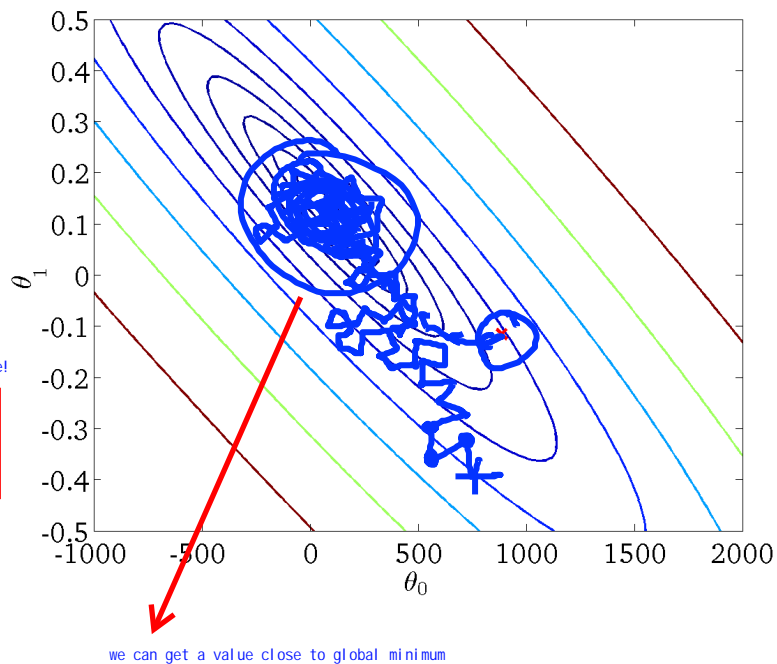
for $i := 1, \dots, m$ { each time step we only look at one training example!

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

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

every }

→ $m = 300,000,000$





Machine Learning

Large scale machine learning

Mini-batch
gradient descent

Another idea sometimes can even work faster than stochastic gradient descent algorithm

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

$(x^{(i)}, y^{(i)}) \dots (x^{(i+9)}, y^{(i+9)})$

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

$i := i + 10$

perform gradient descent update using these 10 examples!

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 $j = 0, \dots, n$)

}

}

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

↑

→ b examples
→ 1 example
Vectorization

Use 10 examples at a time!

mini > stochastic when you have a good vectorization implementation.

we can make progress by looking at the just 10 examples!

$$b = \underline{10}$$

↑



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.

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

$$M = 300,000,000$$

→ 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 $(x^{(i)}, y^{(i)})$.

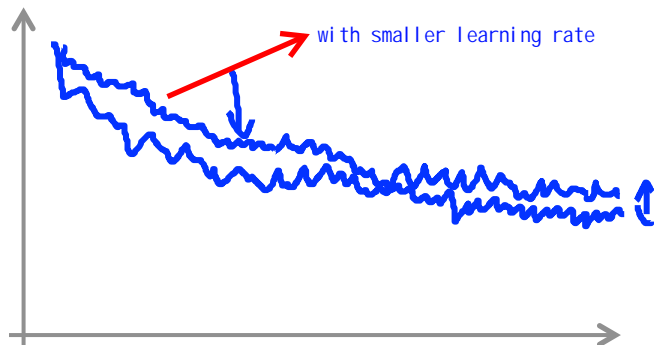
$$\Rightarrow (x^{(i)}, y^{(i)}), (x^{(i+1)}, y^{(i+1)}), \dots$$

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

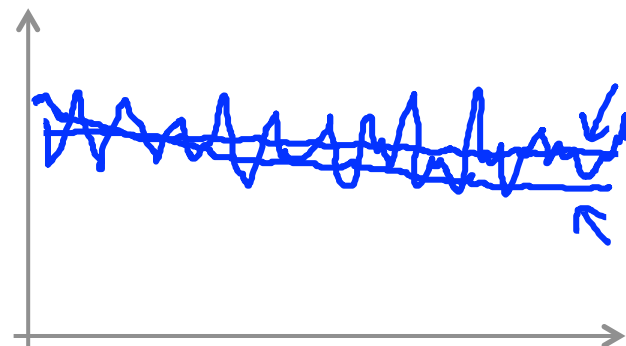
This gives you an idea about how the last 1000 examples doing!

Checking for convergence

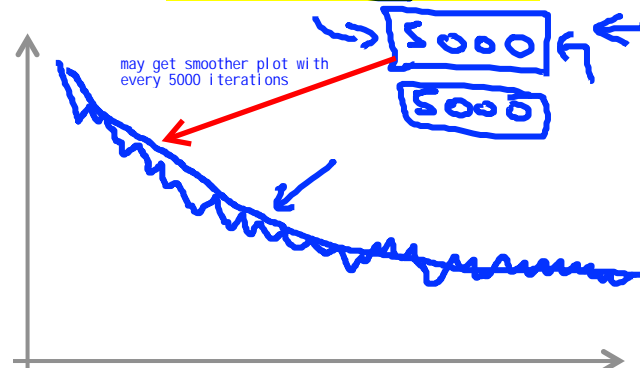
Plot $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples



No. of iterations



No. of iterations



No. of iterations



No. of iterations

Stochastic gradient descent

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

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

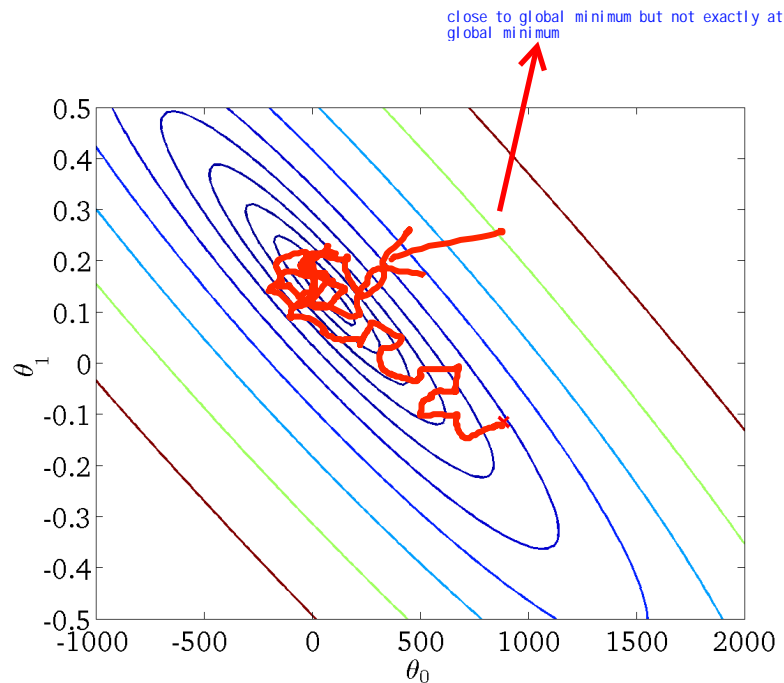
1. Randomly shuffle dataset.
2. Repeat {

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

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

 }

 }



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

dynamic learning rate

as iteration number increases alpha decreases

Stochastic gradient descent

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

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

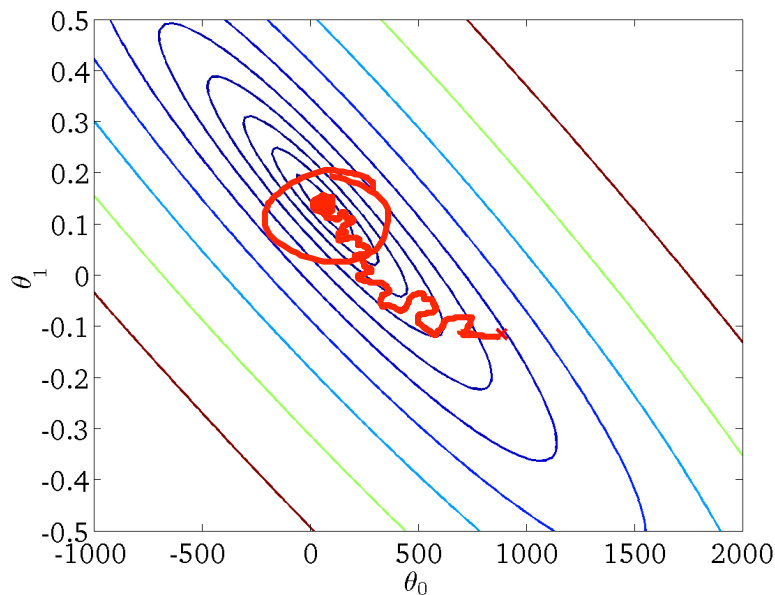
1. Randomly shuffle dataset.
2. Repeat {

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

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

 }

 }



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



Machine Learning

Large scale machine learning

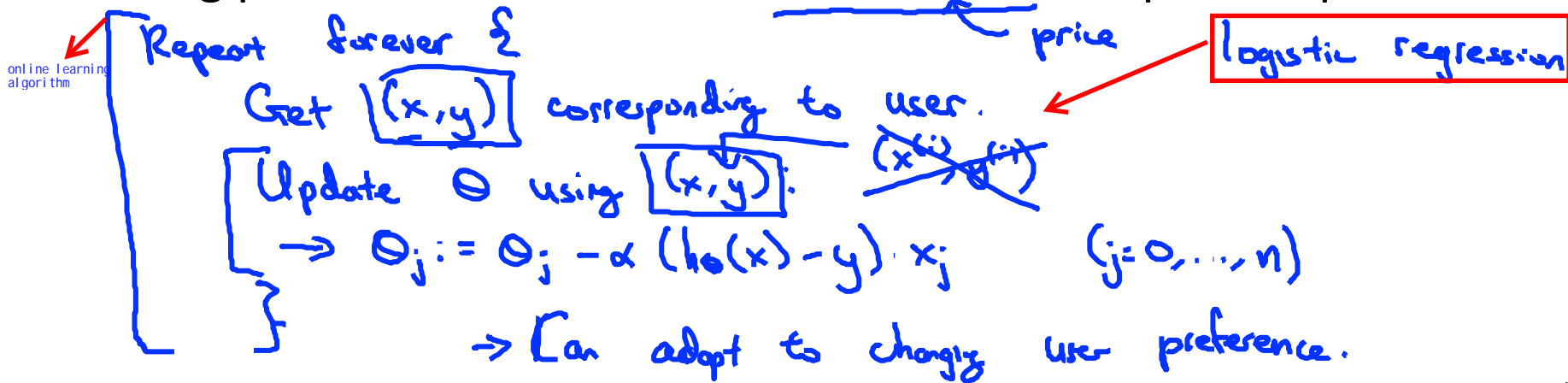
Online learning

when we have a continuous stream of data come in!

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.



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

click through rate

→ 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

Sometimes, some ML problems are just cannot run on machine.

Map-reduce

Batch gradient descent:

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

$$m = 400 \leftarrow$$

$$m = 400,000,000 \rightarrow \text{real world samples}$$

Machine 1: Use $(x^{(1)}, y^{(1)}), \dots, (x^{(100)}, y^{(100)})$. \rightarrow use first 1/4 training samples

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

similarly, we use the 2nd quarter of data

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

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

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

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

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

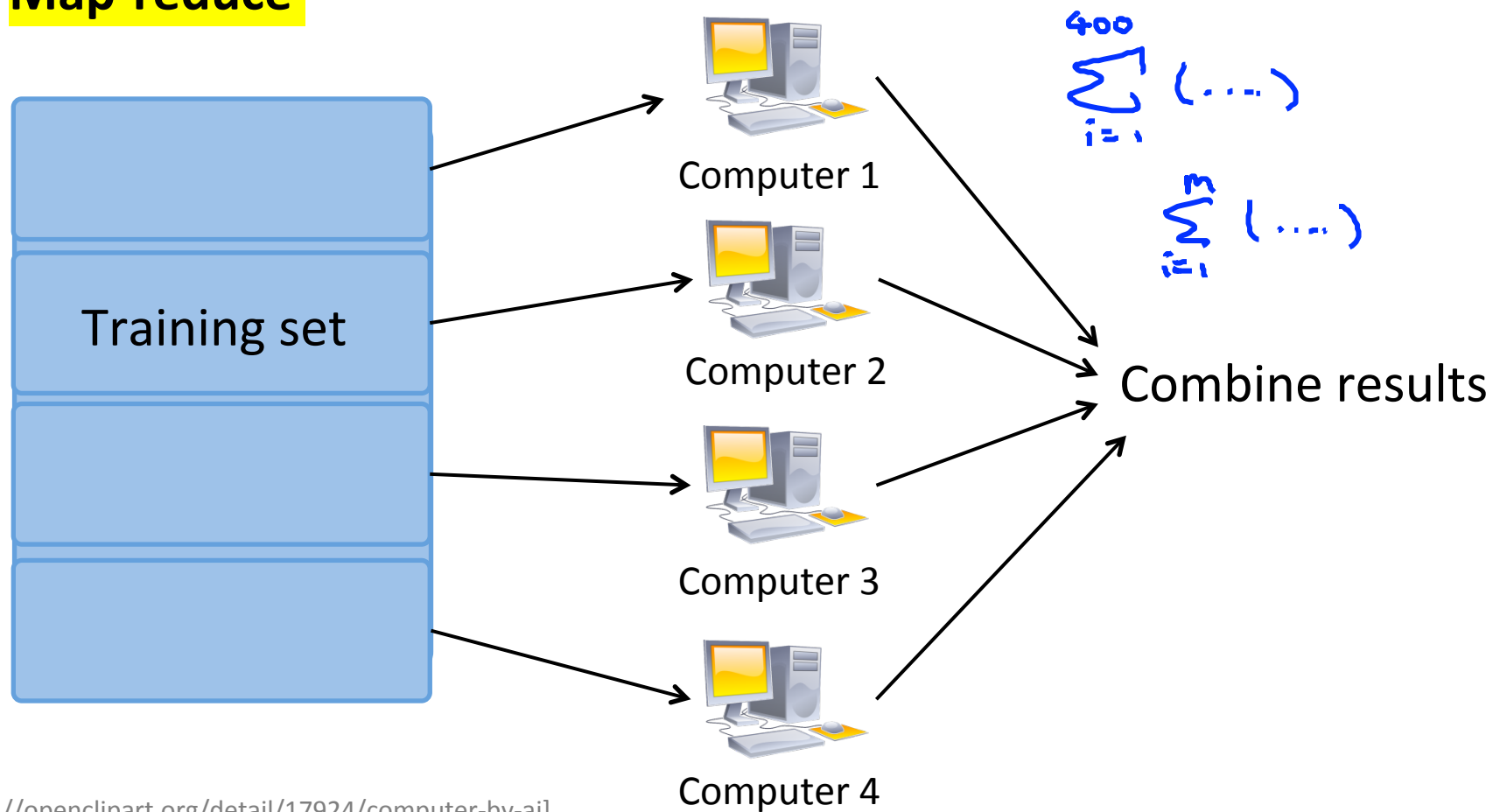
$$\text{temp}_j^{(4)} = \sum_{i=301}^{400} (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Combine:

combine all the results in a centralized master server

$$\begin{aligned} \theta_j &:= \theta_j \\ &- \alpha \frac{1}{400} (\\ &\quad \text{temp}_j^{(1)} + \text{temp}_j^{(2)} \\ &\quad + \text{temp}_j^{(3)} + \text{temp}_j^{(4)}) \\ &(j = 0, \dots, n) \end{aligned}$$

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.

then MapReduce can be a candidate for accelerating your learning algorithm

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

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

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

$temp^{(i)}$

$temp_j \leftarrow$

we can do this summation in a centralized server

Multi-core machines

