



Machine Learning

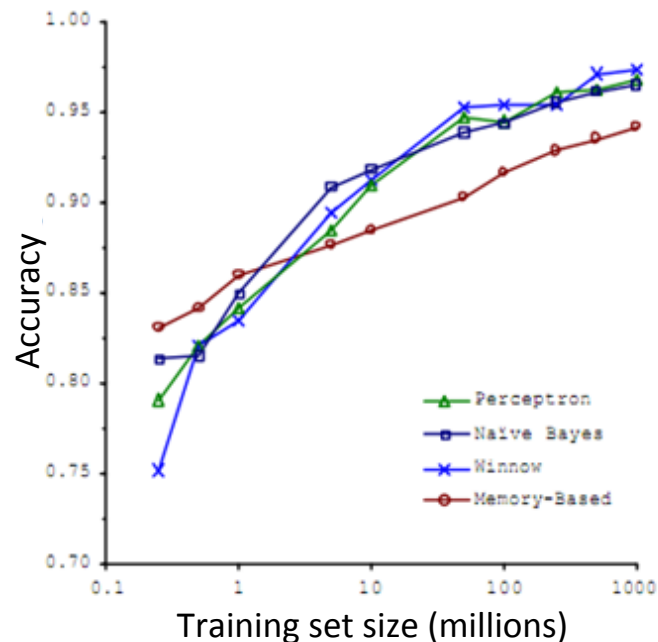
Large scale machine learning

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.



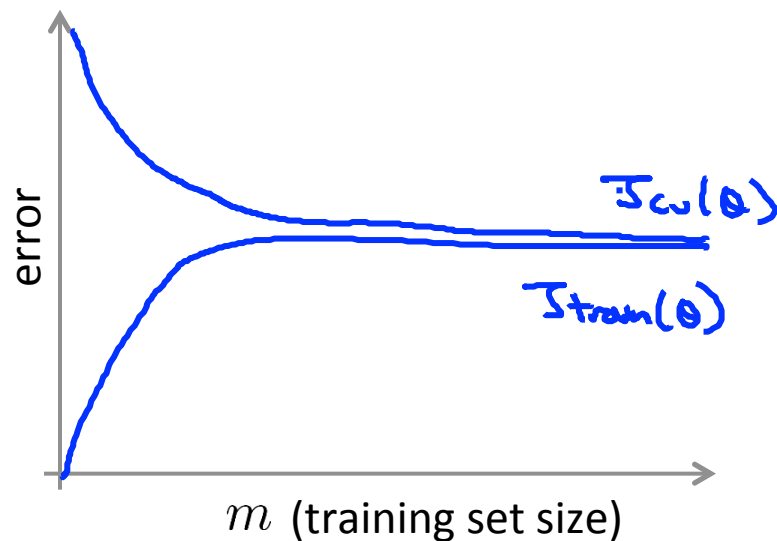
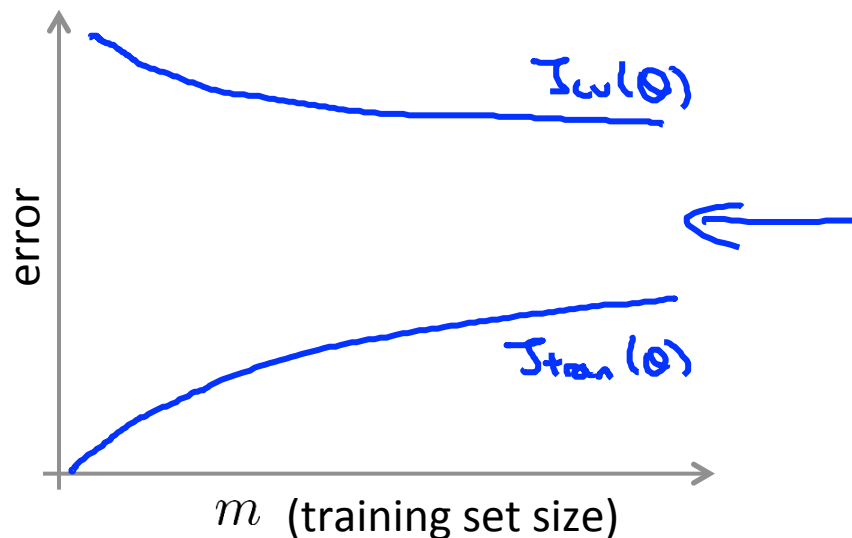
“It’s not who has the best algorithm that wins.
It’s who has the most data.”

Learning with large datasets

$m = 100,000,000$

$m = 1,000?$

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





Machine Learning

Large scale machine learning

Stochastic gradient descent

Linear regression with gradient descent

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

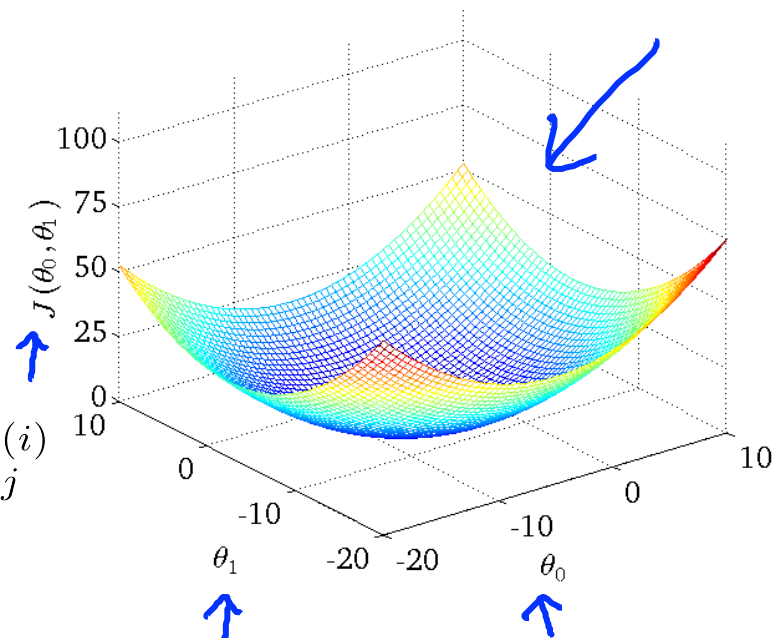
$$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

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 $j = 0, \dots, n$)

}



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 {

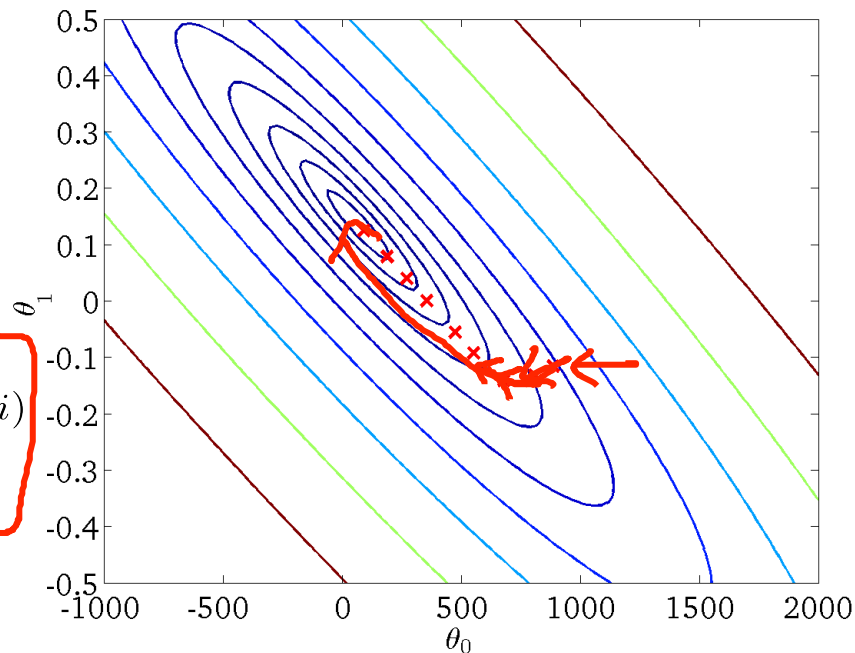
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(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

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

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



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 $j = 0, \dots, n$)

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

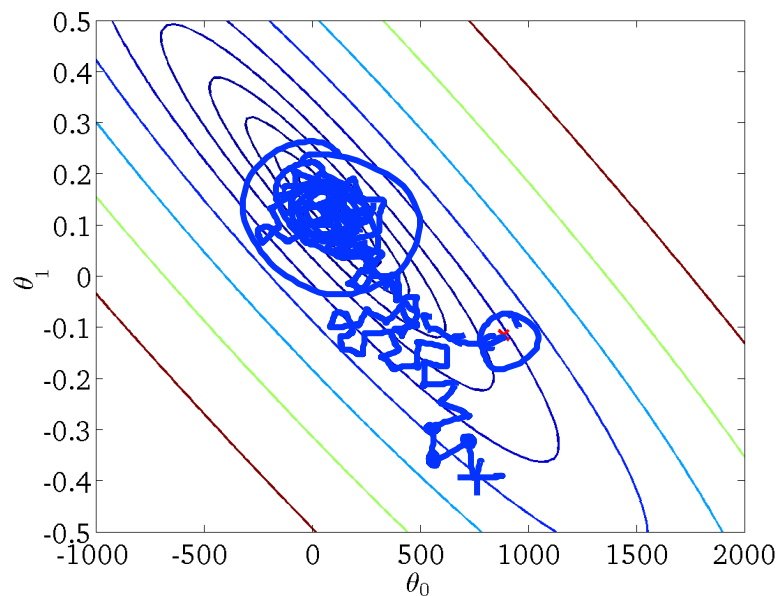
$$\rightarrow (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots$$

Stochastic gradient descent

→ 1. Randomly shuffle (reorder) training examples

→ 2. Repeat { 1-10x
 for $i := 1, \dots, m$ {
 → $\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$
 (for $j = 0, \dots, n$)
 }
every }
}

→ $m = 300,000,000$





Machine Learning

Large scale machine learning

Mini-batch gradient descent

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

$b = \text{mini-batch size}$. $b = 10$. $\frac{2-100}{10}$
Get $\boxed{b=10}$ examples $(x^{(i)}, y^{(i)}) \dots (x^{(i+9)}, y^{(i+9)})$

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

$$i := i + 10$$

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 $j = 0, \dots, n$)

}

}

$m = 300, 600, 900$

↑

→ b examples

→ 1 example

Vectorization

$b = 10$
↑



Machine Learning

Large scale machine learning

Stochastic gradient descent convergence

sự hội tụ

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:

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

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

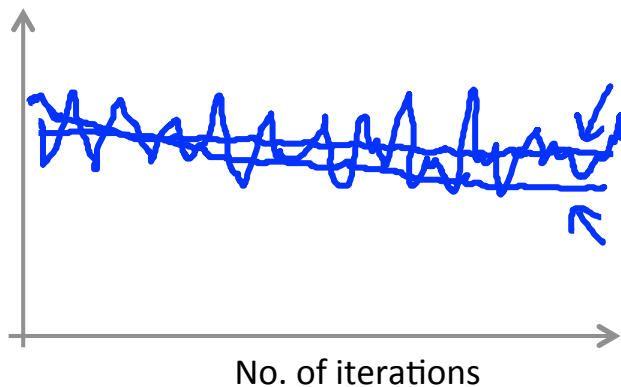
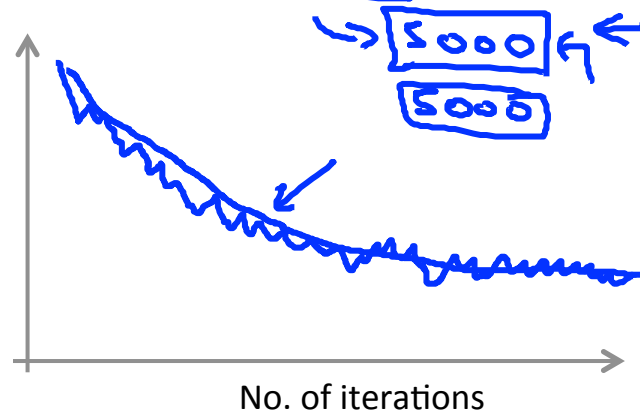
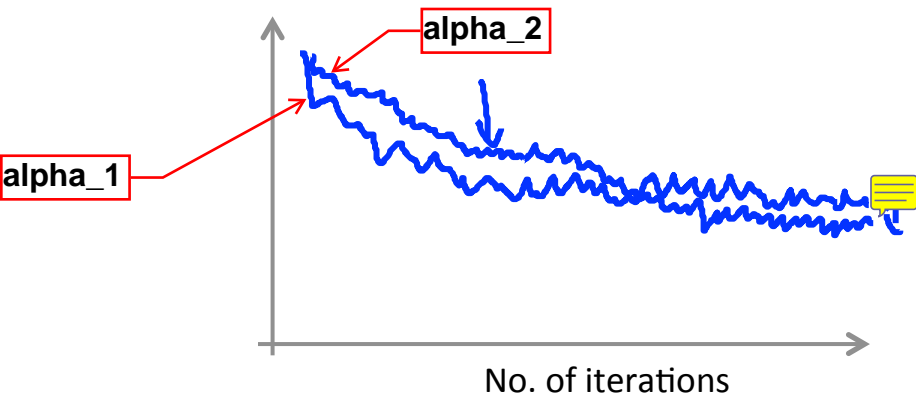
→ During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.

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

Plot for checking the convergence of algorithm

Checking for convergence

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

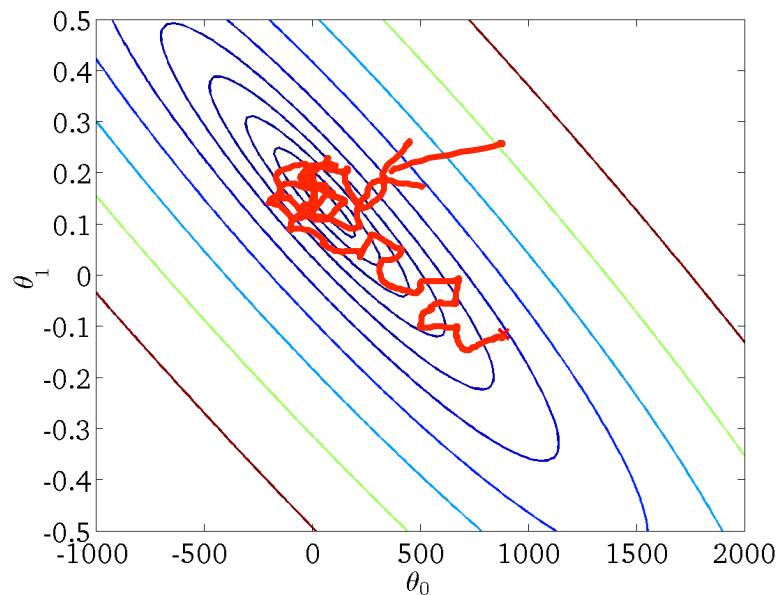


Stochastic gradient descent

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

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

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. $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$)

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. $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$) $\alpha \rightarrow 0$

3. Which of the following statements about online learning are true? Check all that apply.
- ☒ Online learning algorithms are usually best suited to problems where we have a continuous/non-stop stream of data that we want to learn from.
 - ☒ One of the advantages of online learning is that if the function we're modeling changes over time (such as if we are modeling the probability of users clicking on different URLs, and user tastes/preferences are changing over time), the online learning algorithm will automatically adapt to these changes.
 - ☐ When using online learning, you must save every new training example you get, as you will need to reuse past examples to re-train the model even after you get new training examples in the future.
 - ☐ Online learning algorithms are most appropriate when we have a fixed training set of size m that we want to train on.



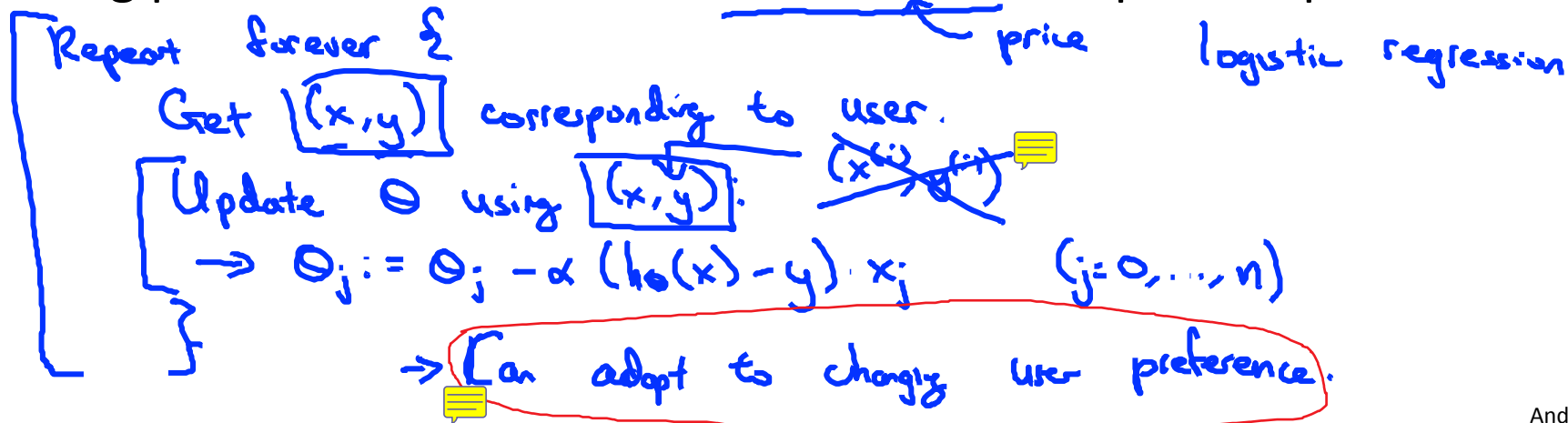
Machine Learning

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.



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. $y = 0$

(x, y) ←
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; ...

5. Which of the following statements about map-reduce are true? Check all that apply.
- ☒ When using map-reduce with gradient descent, we usually use a single machine that accumulates the gradients from each of the map-reduce machines, in order to compute the parameter update for that iteration.
 - ☐ Linear regression and logistic regression can be parallelized using map-reduce, but not neural network training.
 - ☒ Because of network latency and other overhead associated with map-reduce, if we run map-reduce using N computers, we might get less than an N -fold speedup compared to using 1 computer.
 - ☒ If you have only 1 computer with 1 computing core, then map-reduce is unlikely to help.



Machine Learning

Scale learning

Map-reduce and data parallelism

Map-reduce

Batch gradient descent:

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

$$m = 400 \leftarrow$$

$$m = 400,000,000$$

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

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

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

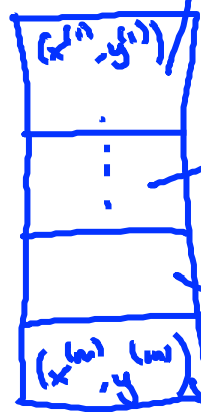
$$\text{temp}_j^{(4)} = \sum_{i=301}^{400} (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Use only 1 machine to accumulate result for performing gradient descent

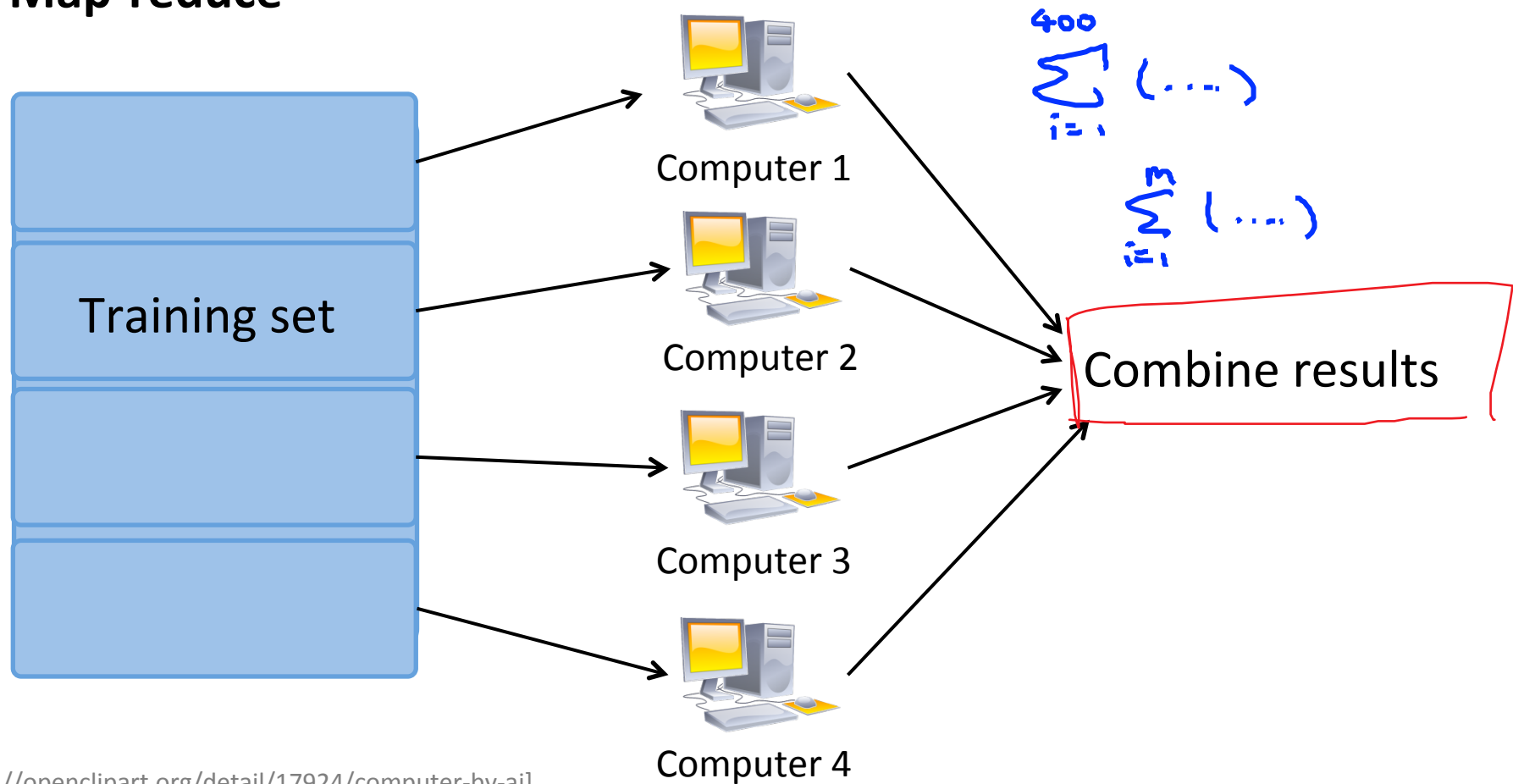
Combine:

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

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



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:

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


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

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

Multi-core machines

