



Machine Learning

# Large scale machine learning

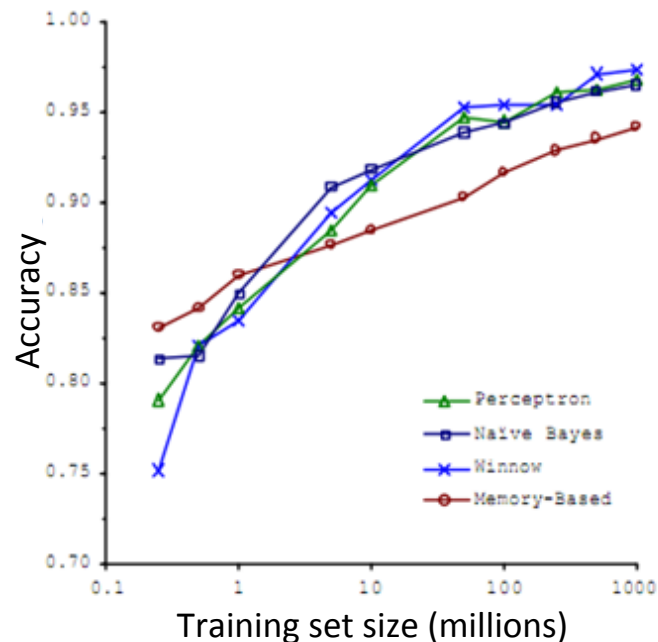
---

## 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.”

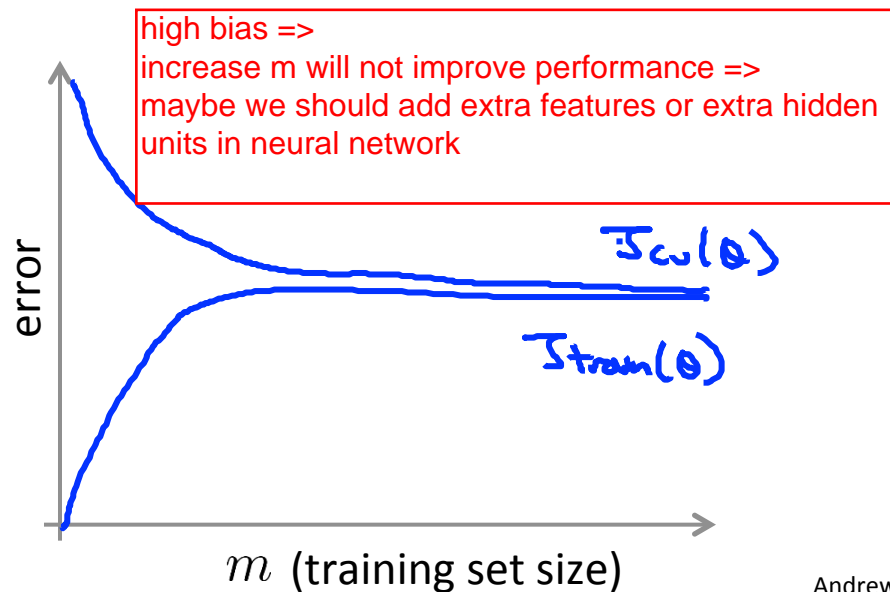
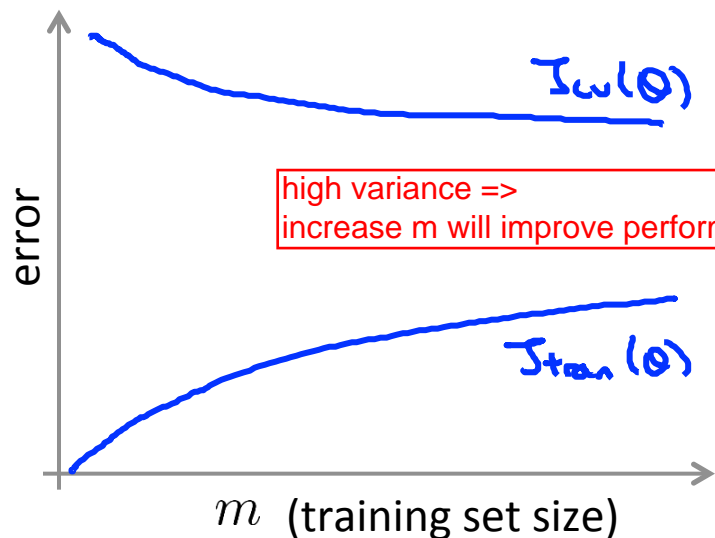
# Learning with large datasets

$m = 100,000,000$

we can randomly pick the subsets of a thousand examples out of a hundred million examples and train our algorithm on just a thousand examples to sanity check and plot the learning curve

$m = 1,000?$

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



Suppose you are facing a supervised learning problem and have a very large dataset ( $m = 100,000,000$ ). How can you tell if using all of the data is likely to perform much better than using a small subset of the data (say  $m = 1,000$ )?

---

- ☐ There is no need to verify this; using a larger dataset always gives much better performance.
- ☐ Plot  $J_{\text{train}}(\theta)$  as a function of the number of iterations of the optimization algorithm (such as gradient descent).
- ☐ Plot a learning curve ( $J_{\text{train}}(\theta)$  and  $J_{\text{CV}}(\theta)$ , plotted as a function of  $m$ ) for some range of values of  $m$  (say up to  $m = 1,000$ ) and verify that the algorithm has bias when  $m$  is small.
- ☒ Plot a learning curve for a range of values of  $m$  and verify that the algorithm has high variance when  $m$  is small.

Correct



Machine Learning

# Large scale machine learning

---

## Stochastic gradient descent

when you have a large data set, gradient decent becomes  
computationally expensive =>  
use stochastic gradient descent instead

# Linear regression with gradient descent

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

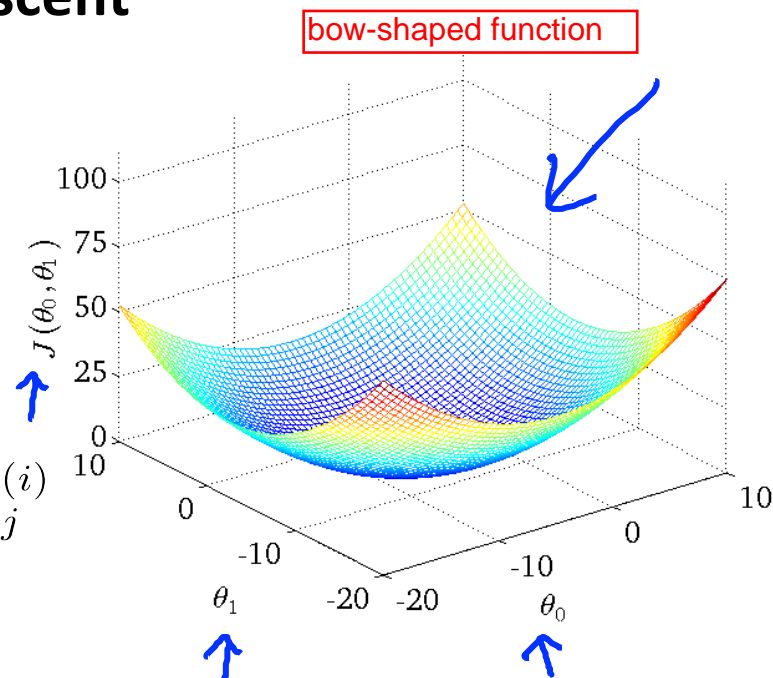
$$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

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  $j = 0, \dots, n$ )

}

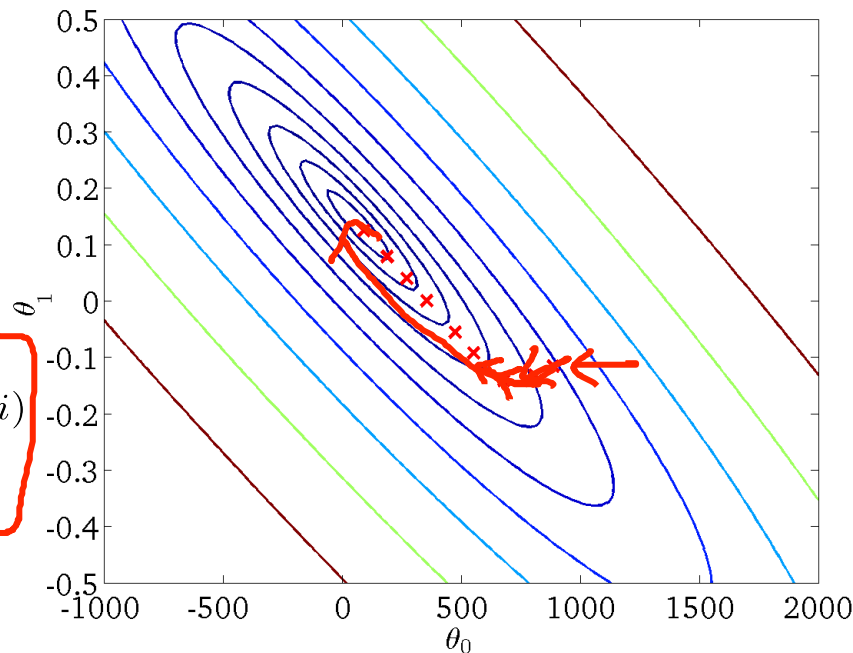
if large m, this becomes  
computationally expensive

$M = 300,000,000$

this version is  
called

Batch gradient descent

batch: looking at all the  
training example



## 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.  $\leftarrow$

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

usually only need to repeat it 1-10 times

→ 2. Repeat { 1-10x

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

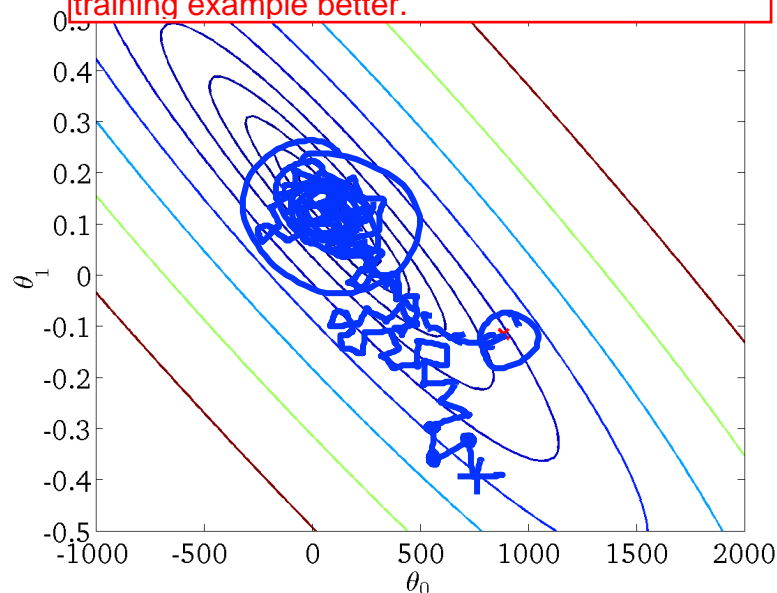
→  $\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$

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

every

→  $m = 300,000,000$

with Stochastic gradient descent every iteration is going to be much faster because we don't need to sum up over all the training examples. But every iteration is just trying to fit single training example better.



as you run Stochastic gradient descent it doesn't actually converge as Batch gradient descent does, it ends up wandering around continuously in some region that's in some region close to the global minimum, it doesn't get to the global minimum and stay there. But in practice its okay as long as the parameters end up close to the global minimum for most practical purposes.

Which of the following statements about stochastic gradient descent are true? Check all that apply.

☒ When the training set size  $m$  is very large, stochastic gradient descent can be much faster than gradient descent.

Correct

☒ The cost function  $J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$  should go down with every iteration of batch gradient descent (assuming a well-tuned learning rate  $\alpha$ ) but not necessarily with stochastic gradient descent.

Correct

☐ Stochastic gradient descent is applicable only to linear regression but not to other models (such as logistic regression or neural networks).

Un-selected is correct

☒ Before beginning the main loop of stochastic gradient descent, it is a good idea to "shuffle" your training data into a random order.

Correct



Machine Learning

# Large scale machine learning

---

## Mini-batch gradient descent

another variation of stochastic gradient descent : Mini-batch gradient descent, they can work sometimes even a bit faster than stochastic 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

typical range :  $b = 2 - 100$ , Andrew usually use  $b=10$

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

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

$b = 10$

$2 - 100$

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

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

$i := i + 10$

# Mini-batch gradient descent

Say  $b = 10$ ,  $m = 1000$ .

mini-batch:

→  $b$  examples

stochastic:

→ 1 example

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  $j = 0, \dots, n$ )

}

}

$m = 300, 600, 900$

↑

Vectorization

Mini-batch gradient descent is likely to outperform Stochastic gradient descent only if you have a good vectorized implementation  
=> the sum of the 10 examples can be performed in a vectorized way  
=>  
so you can use parallelized linear algebra library to speed it up

$b = 10$   
↑

Suppose you use mini-batch gradient descent on a training set of size  $m$ , and you use a mini-batch size of  $b$ . The algorithm becomes the same as batch gradient descent if:

---

- ☐  $b = 1$
- ☐  $b = m / 2$
- ☒  $b = m$

Correct

- ☐ None of the above



Machine Learning

# Large scale machine learning

---

## Stochastic gradient descent convergence

how do you know the code is correct and how to tune the learning rate  $\alpha$  ?

# Checking for convergence

→ Batch gradient descent:

make sure that this cost function is decreasing on every iteration

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

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

when large m you don't want to pause the code to plot J vs m.

$M = 300,000,000$

→ 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  $\theta$  using  $(x^{(i)}, y^{(i)})$ .

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

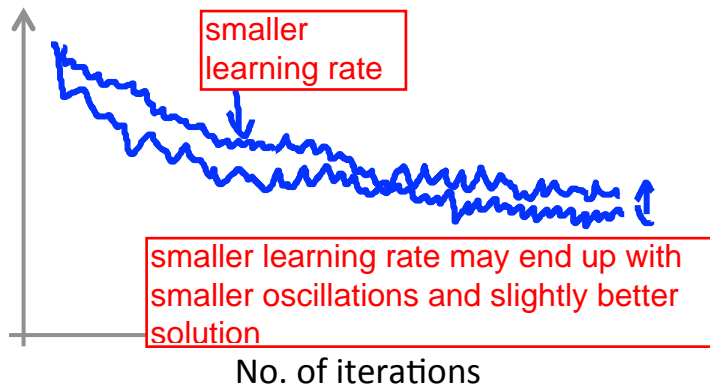
→ Every 1000 iterations (say), plot  $cost(\theta, (x^{(i)}, y^{(i)}))$  averaged over the last 1000 examples processed by algorithm.



# Checking for convergence

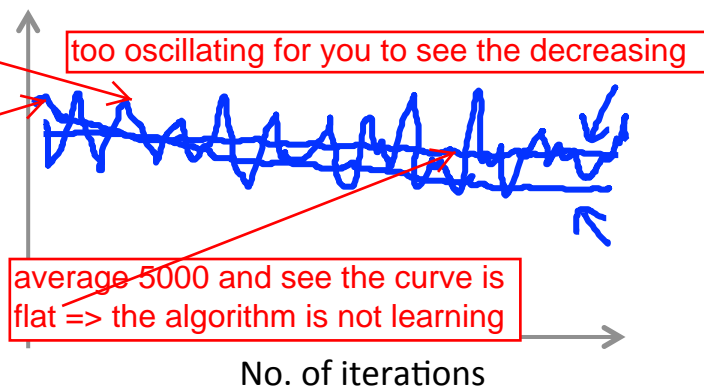
Plot  $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ , averaged over the last 1000 (say) examples

the plot is noisy (not smooth) because it's average over 1000 training examples.

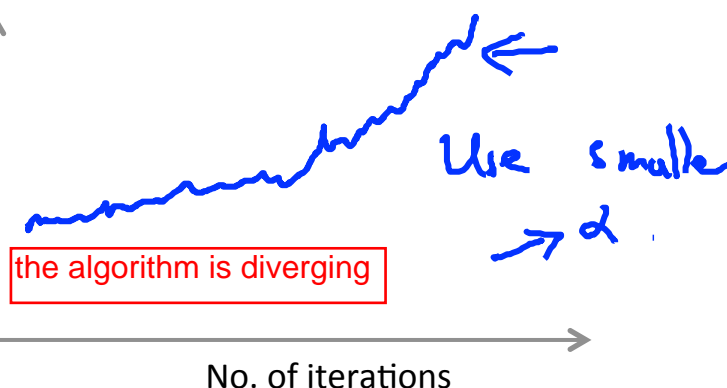
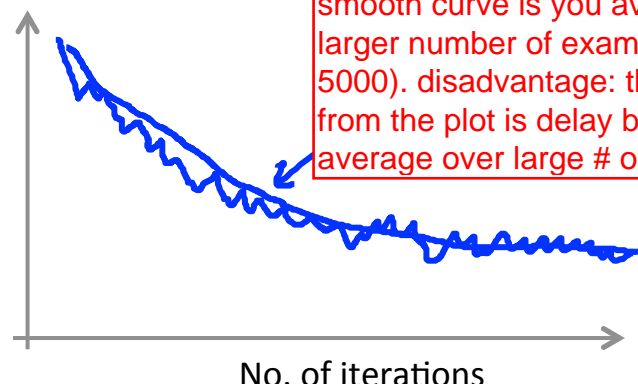


average 1000 and see

average 5000 and see it => you can see it smooth and decreasing.

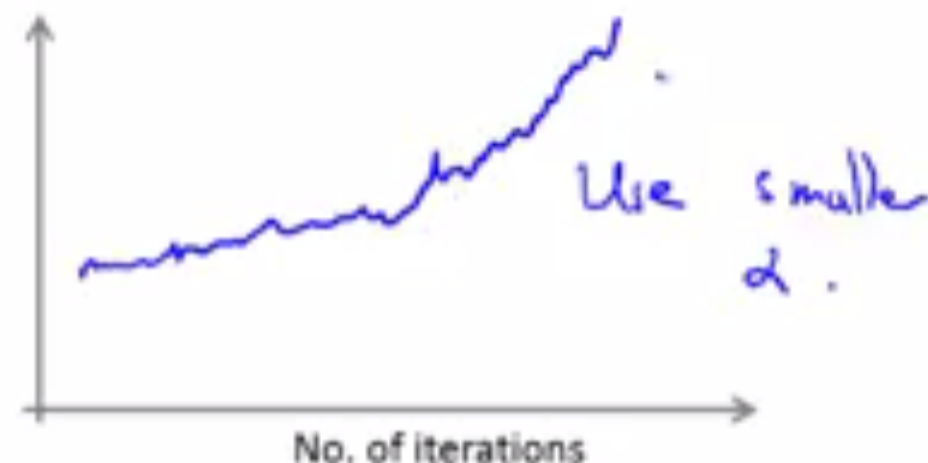
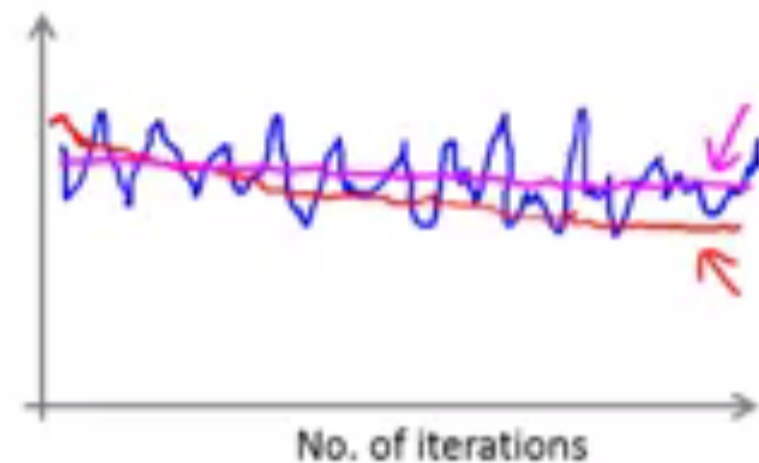
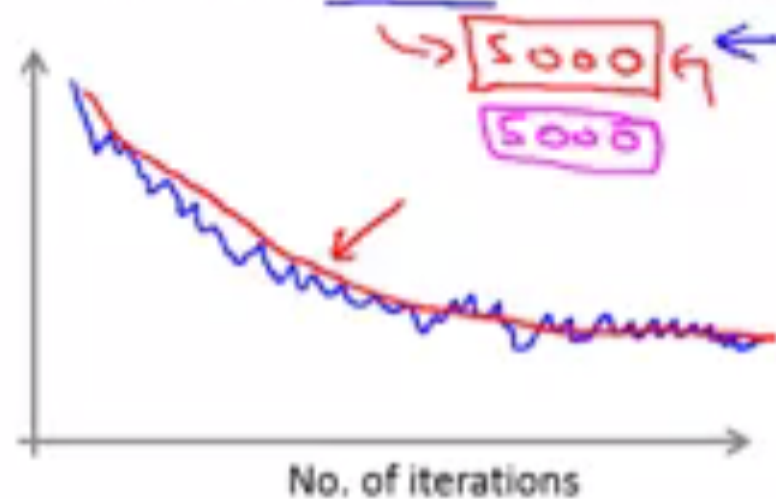
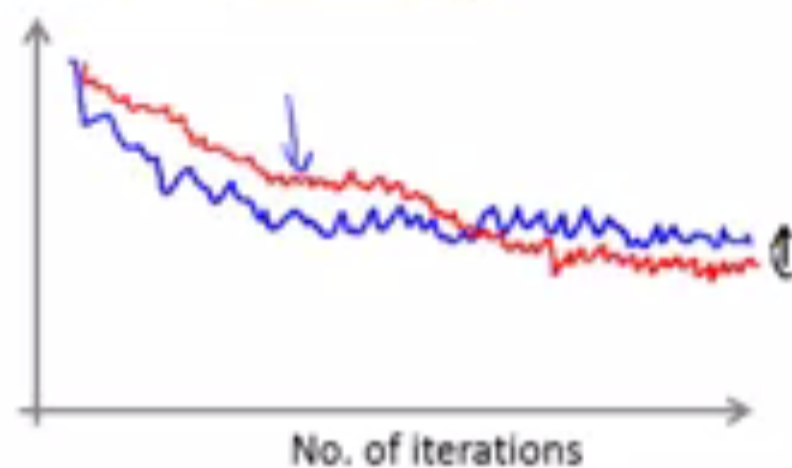


smooth curve is you average over larger number of examples (say, 5000). disadvantage: the feedback from the plot is delay because you average over large # of examples



# Checking for convergence

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

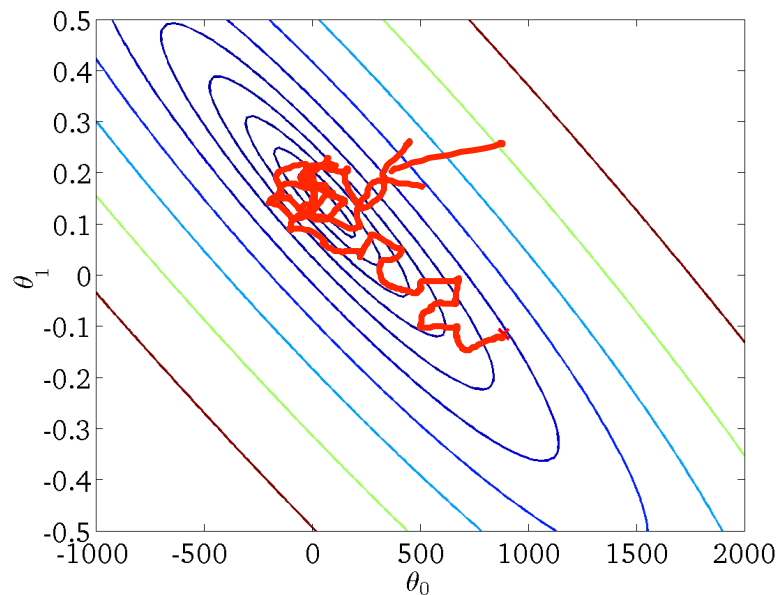


# Stochastic gradient descent

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

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

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  $j = 0, \dots, n$ )  
    }



One of the reasons people tend not to do this is because you need to spend time playing with 2 extra parameters and this makes the algorithm more finicky, But if you manage to tune the parameters well, as it gets closer you're decreasing the learning rate and ends up giving a solution closer to global minimum.

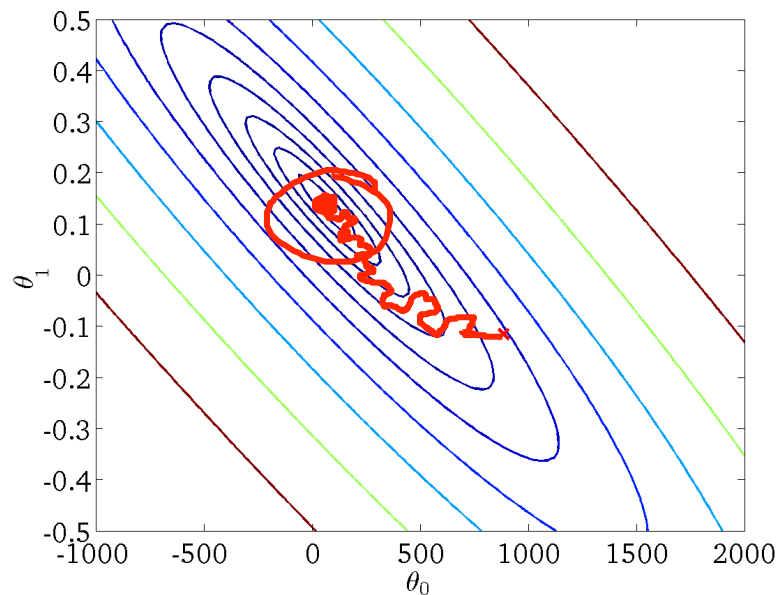
Learning rate  $\alpha$  is typically held constant. Can slowly decrease  $\alpha$  over time if we want  $\theta$  to converge. (E.g.  $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$ )

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

Which of the following statements about stochastic gradient descent are true? Check all that apply.

☐ Picking a learning rate  $\alpha$  that is very small has no disadvantage and can only speed up learning.

Un-selected is correct

☒ If we reduce the learning rate  $\alpha$  (and run stochastic gradient descent long enough), it's possible that we may find a set of better parameters than with larger  $\alpha$ .

Correct

☐ If we want stochastic gradient descent to converge to a (local) minimum rather than wander or "oscillate" around it, we should slowly increase  $\alpha$  over time.

Un-selected is correct

☒ If we plot  $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$  (averaged over the last 1000 examples) and stochastic gradient descent does not seem to be reducing the cost, one possible problem may be that the learning rate  $\alpha$  is poorly tuned.

Correct

To summarize we talk about a way for approximately monitoring how the stochastic gradient descent is doing in terms for optimizing the cost function. And this is a method that does not require scanning over the entire training set periodically to compute the cost function on the entire training set. But instead it looks at only the last thousand examples or so. And you can use this method both to make sure the stochastic gradient descent is okay and is converging or to use it to tune the learning rate  $\alpha$ .



Machine Learning

# Large scale machine learning

---

## Online learning

The online learning setting allows us to model problems where we have a continuous flood or a continuous stream of data coming in and we would like an algorithm to learn from that.

if you have a continuous stream of data generated by a continuous stream of users coming to your website, what you can do is to use online learning algorithm to learn user preferences from the stream of data and use that to optimize some of the decisions on your website.

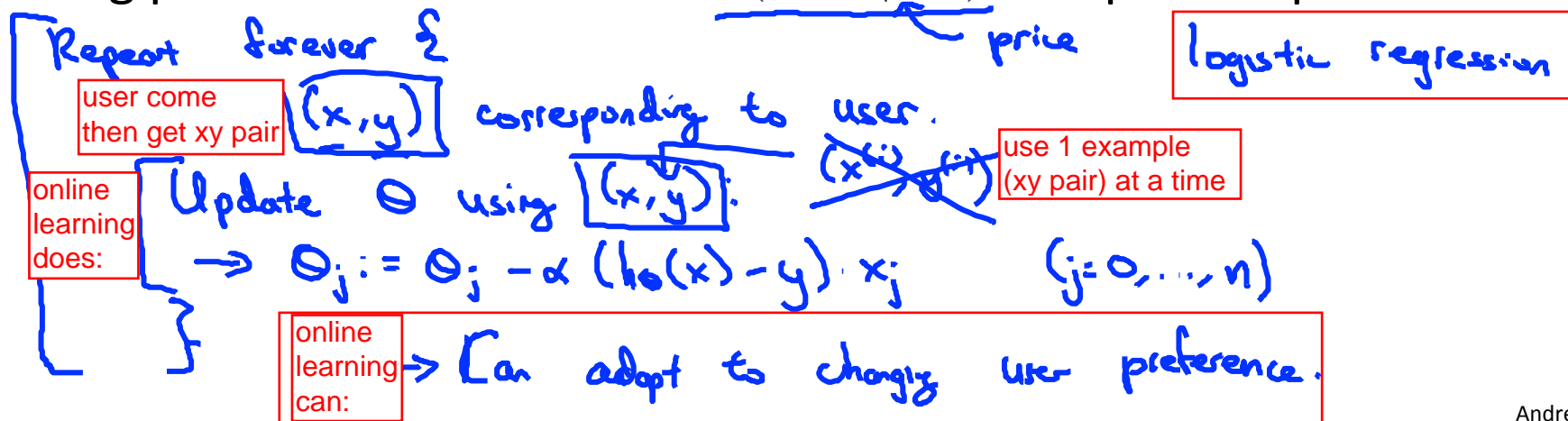
And so if we could estimate the chance that they'll agree to use our service for any given price, then we can try to pick a price so that they have a pretty high probability of choosing our website while simultaneously hopefully offering us a fair return, offering us a fair profit for shipping their package.

## 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 ( $y = 0$ ).

positive example

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

its the problem of learning the predicted click-through rate, the predicted CTR

→ Use to show user the 10 phones they're most likely to click on.

⇒ give 10  $x, y$  pairs

Other examples: Choosing special offers to show user; customized selection of news articles; product recommendation; ...

Some of the advantages of using an online learning algorithm are:

- ☒ It can adapt to changing user tastes (i.e., if  $p(y|x;\theta)$  changes over time).

Correct

- ☐ There is no need to pick a learning rate  $\alpha$ .

Un-selected is correct

- ☒ It allows us to learn from a continuous stream of data, since we use each example once then no longer need to process it again.

Correct

we can throw away the data after we use it

- ☐ It does not require that good features be chosen for the learning task.

Un-selected is correct



Machine Learning

# Large scale machine learning

---

## Map-reduce and data parallelism

using these ideas you might be able to scale learning algorithms to even far larger problems than is possible using stochastic gradient descent.

# Map-reduce

Batch gradient descent:

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

map reduce:

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  $(x^{(101)}, y^{(101)}), \dots, (x^{(200)}, y^{(200)})$ .

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

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

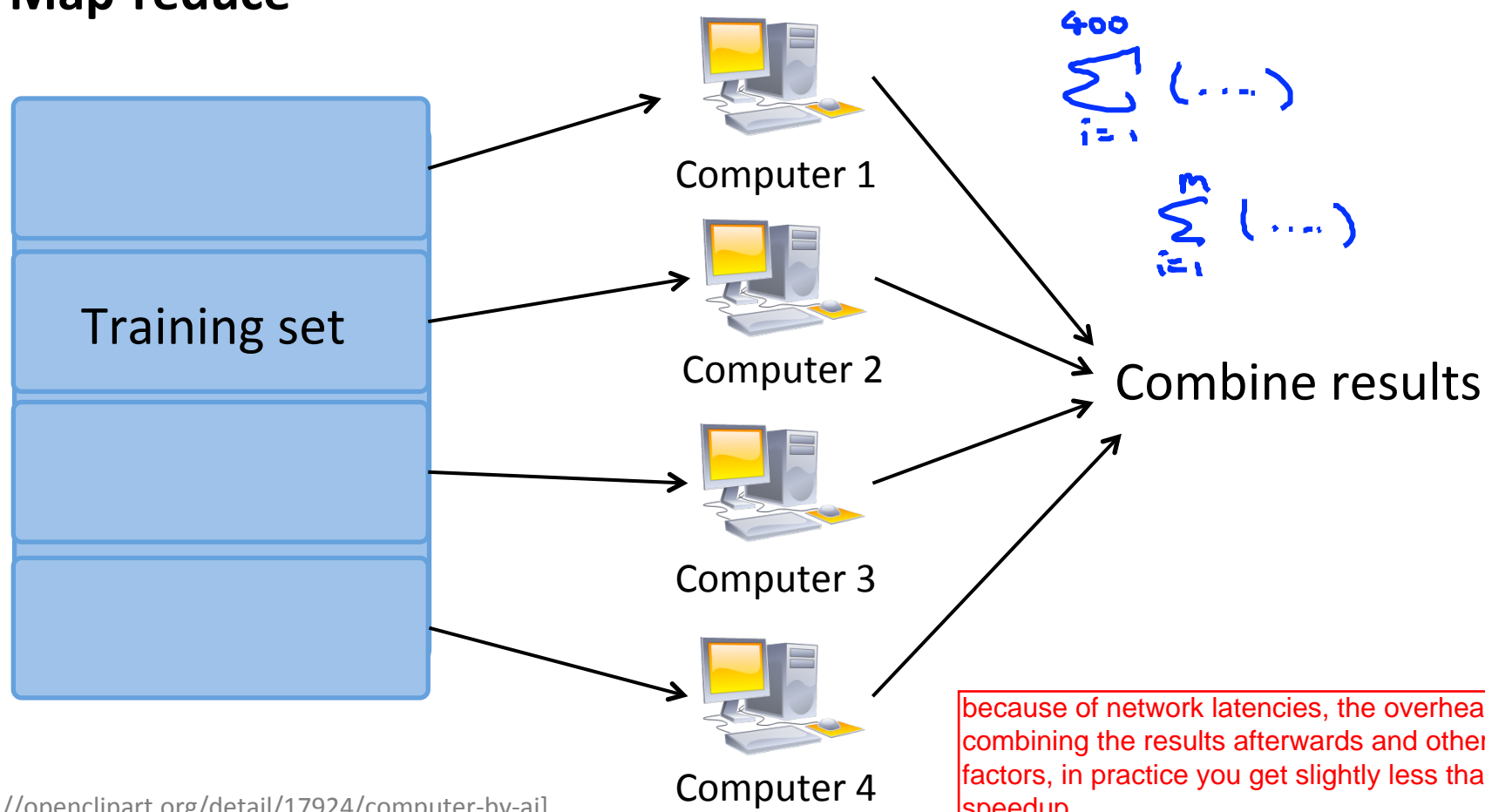
$$\text{temp}_j^{(4)} = \sum_{i=301}^{400} (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Combine:

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

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


# 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:

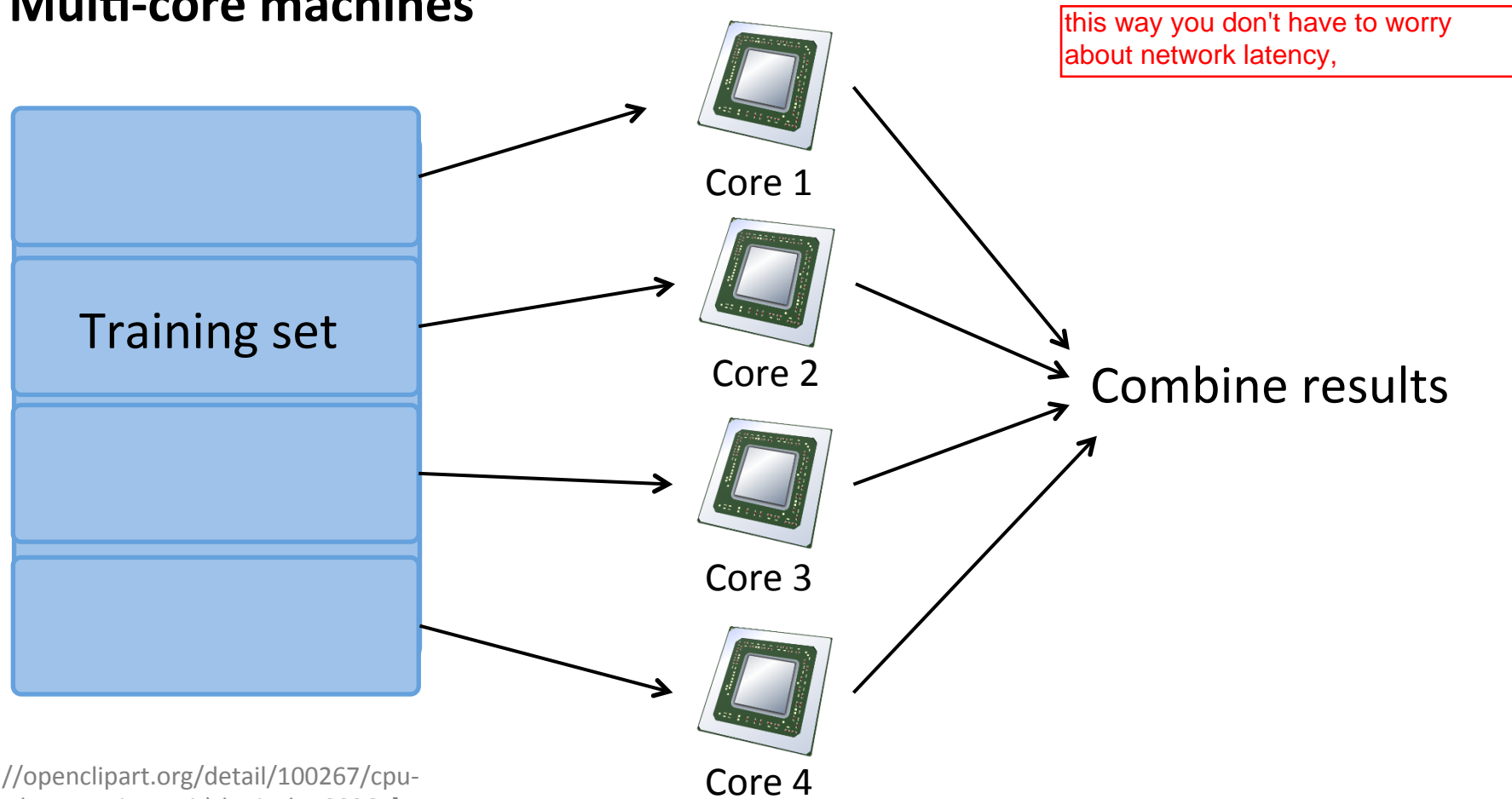
$$\rightarrow \underline{J_{train}(\theta)} = -\frac{1}{m} \sum_{i=1}^m \underline{y^{(i)} \log h_{\theta}(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))}$$


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

$temp^{(i)}$

$temp_j^{(i)} \leftarrow$

# Multi-core machines



Suppose you apply the map-reduce method to train a neural network on ten machines. In each iteration, what will each of the machines do?

- ☐ Compute either forward propagation or back propagation on  $1/5$  of the data.
- ☒ Compute forward propagation and back propagation on  $1/10$  of the data to compute the derivative with respect to that  $1/10$  of the data.

Correct

- ☐ Compute only forward propagation on  $1/10$  of the data. (The centralized machine then performs back propagation on all the data).
- ☐ Compute back propagation on  $1/10$  of the data (after the centralized machine has computed forward propagation on all of the data).