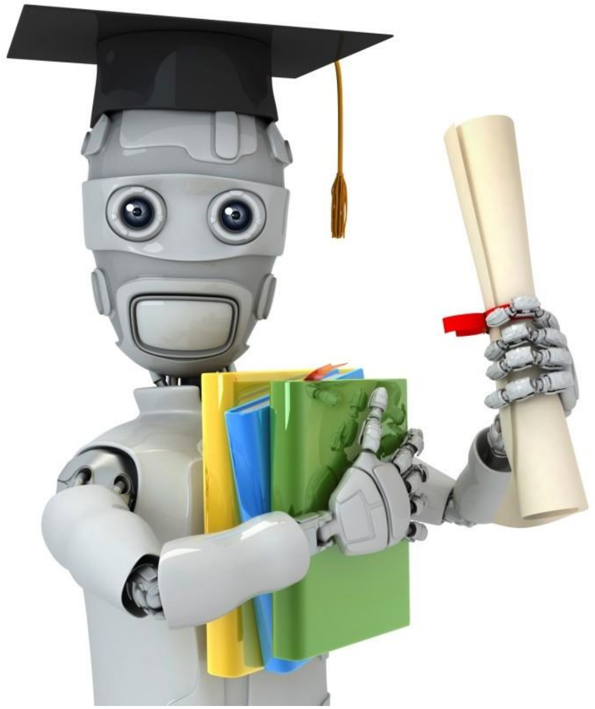


Machine Learning

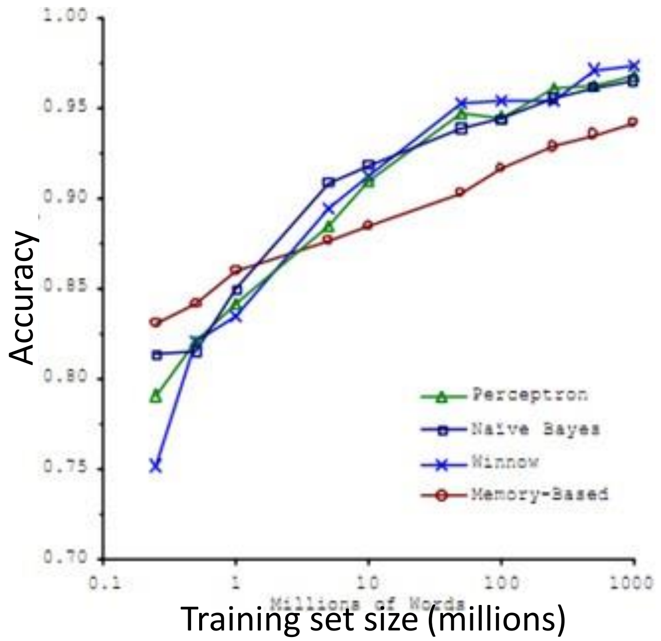
Large scale machine learning

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."

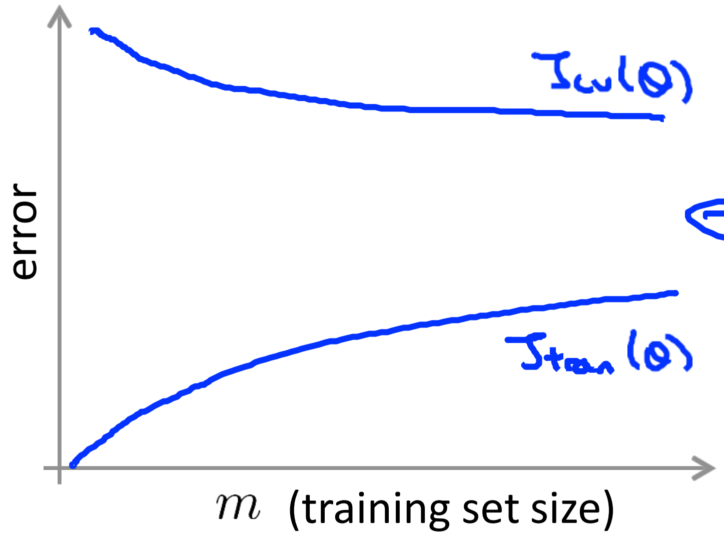
[Figure from Banko and Brill, 2001]

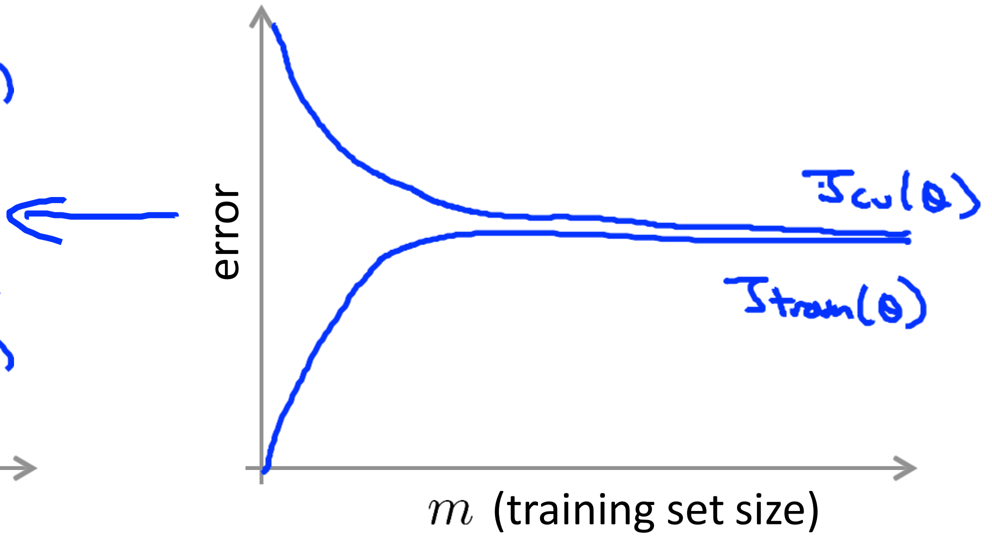
Learning with large datasets

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

$m = 1{,}000?$

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





Andrew Ng

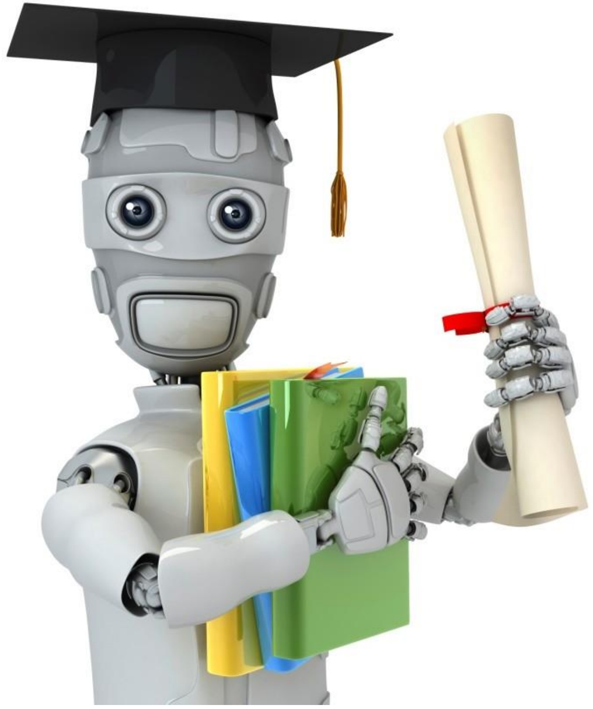
Suppose you are facing a supervised learning problem and have a very large dataset (m = 100,000,000). How can you tell if using all of the data is likely to perform much better than using a small subset of the data (say m = 1,000)?

○ 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.]



Machine Learning

Large scale machine learning

Stochastic gradient descent

Linear regression with gradient descent

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

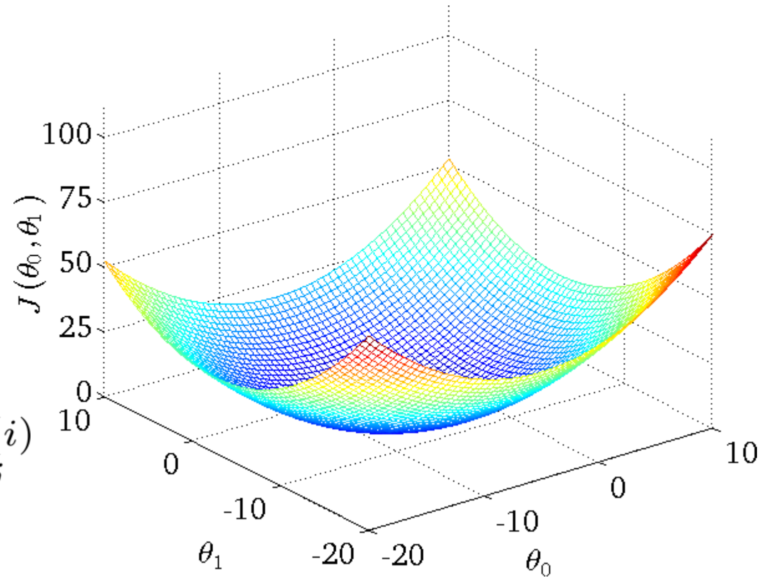
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

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

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

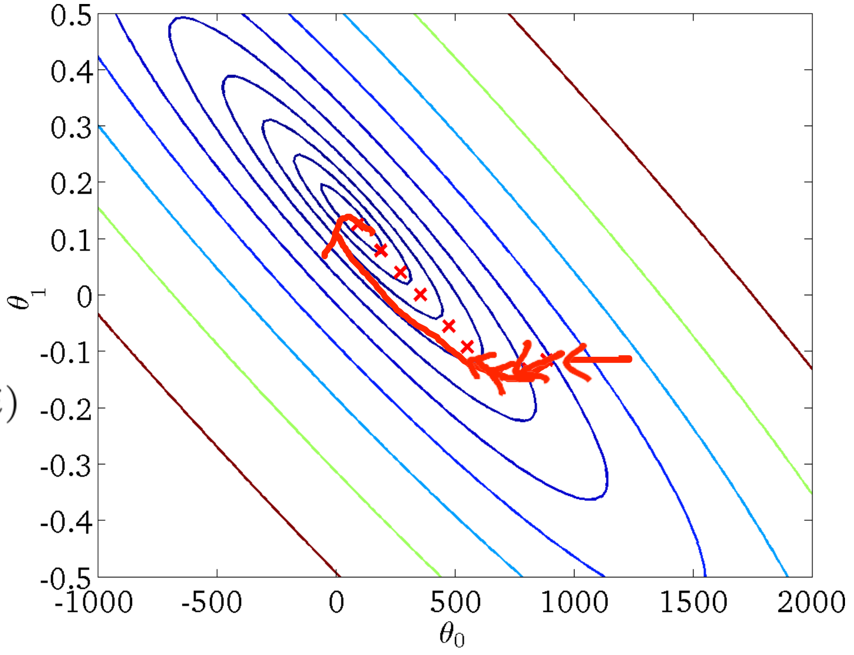
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

(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 {

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

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

}

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

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,...,m {
        $\theta_j := \theta_j - \alpha \boxed{(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}}$
            (for $j=0,\ldots,n$)
    }
}
```

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

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

Andrew Ng

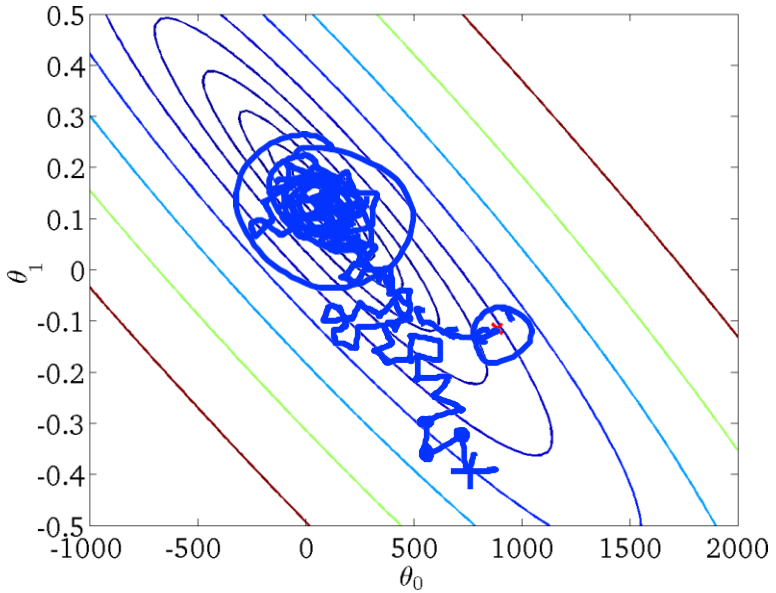
Stochastic gradient descent

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

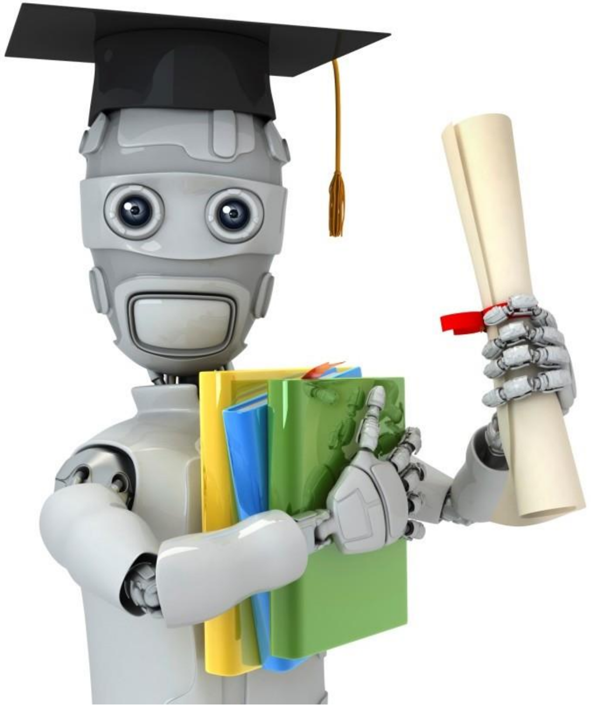
Which of the following statements about stochastic gradient descent are true? Check all that apply.

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

- ☐ 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 α) but not necessarily with stochastic gradient descent.

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

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



Machine Learning

Large scale machine learning

Mini-batch gradient descent

Example: Text categorization

Example by Leon Bottou:

- **Reuters RCV1** document corpus
  - Predict a category of a document
    - One **vs.** the rest classification
- ***n* = 781,000** training examples (documents)
- 23,000 test examples
- ***d* = 50,000** features
  - One feature per word
  - Remove stop-words
  - Remove low frequency words

Example: Text categorization

Questions:

(1) Is **SGD** successful at minimizing ***f(w,b)***?

(2) How quickly does **SGD** find the min of ***f(w,b)***?

(3) What is the error on a test set?

	Training time	*Value of f(w,b)*	*Test error*
Standard SVM	23,642 secs	0.2275	6.02%
"Fast SVM"	66 secs	0.2278	6.03%
SGD SVM	1.4 secs	0.2275	6.02%

(1) SGD-SVM is successful at minimizing the value of ***f(w,b)***

(2) SGD-SVM is super fast

(3) SGD-SVM test set error is comparable

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. $b = 10$. $\underline{2 - 100}$

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 $b = 10, m = 1000$.

```
Repeat {
    for $i = 1, 11, 21, 31, \ldots, 991$ {
        $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$
                (for every $j = 0, \ldots, n$)
    }
}
```

$m = 300,000,000$

b examples

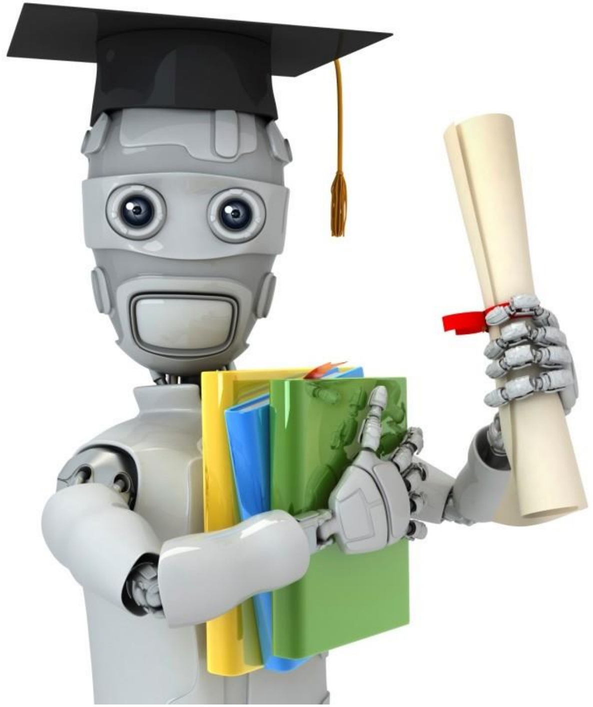
1 example

Vectorization

$b = 10$

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]
- ○ None of the above



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:

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

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

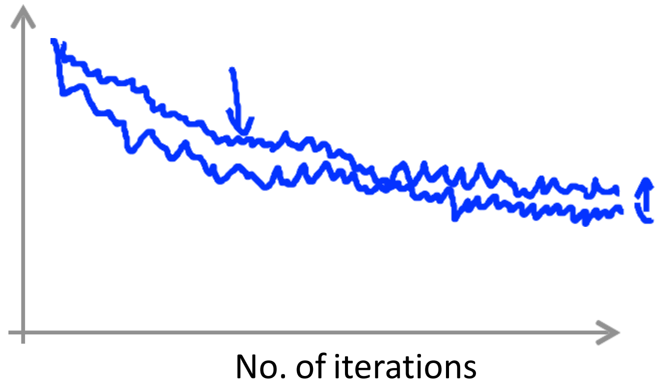
During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.

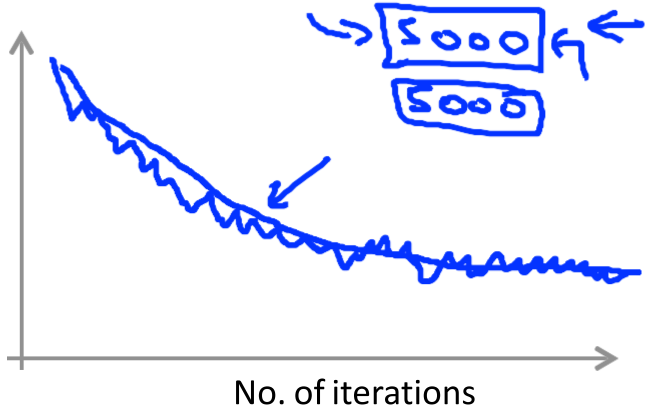
Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.

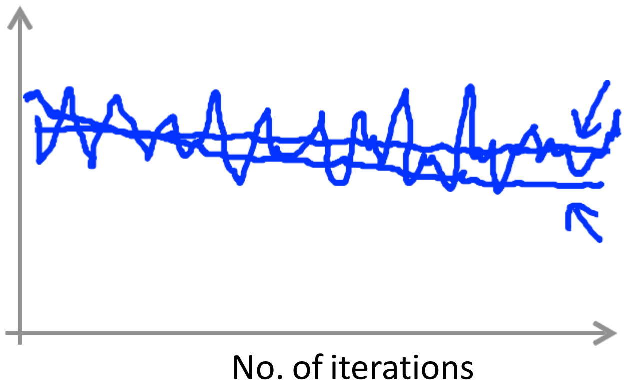
Andrew Ng

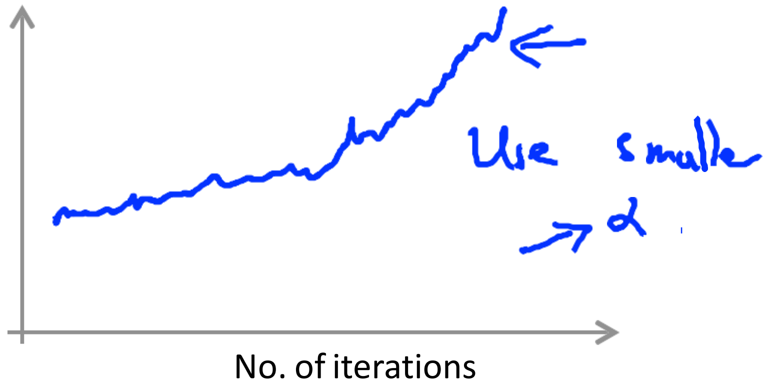
Checking for convergence

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









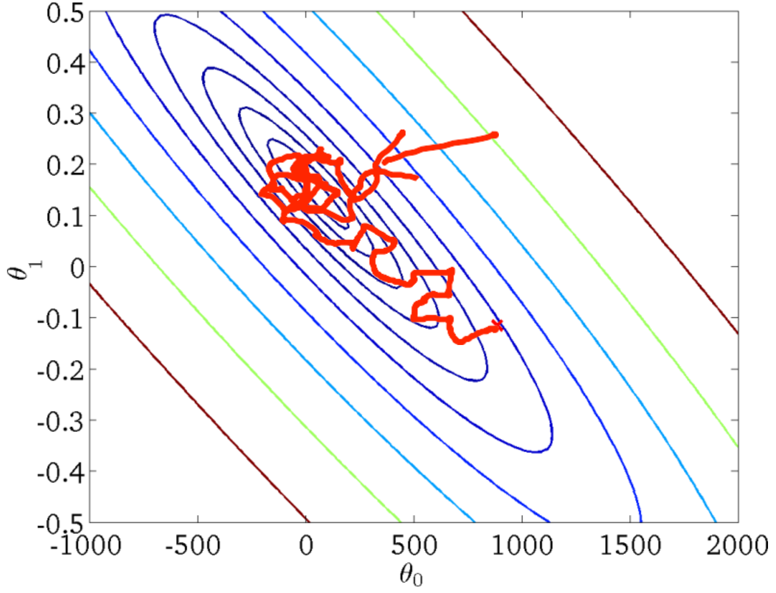
Andrew Ng

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

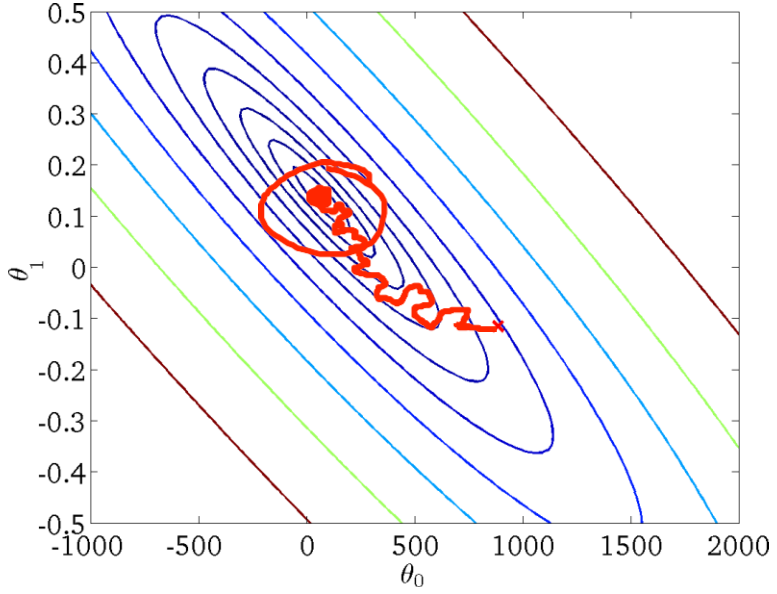
Andrew Ng

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. $\alpha = \frac{\texttt{const1}}{\texttt{iterationNumber + const2}}$) $\underline{\alpha \to 0}$

Andrew Ng

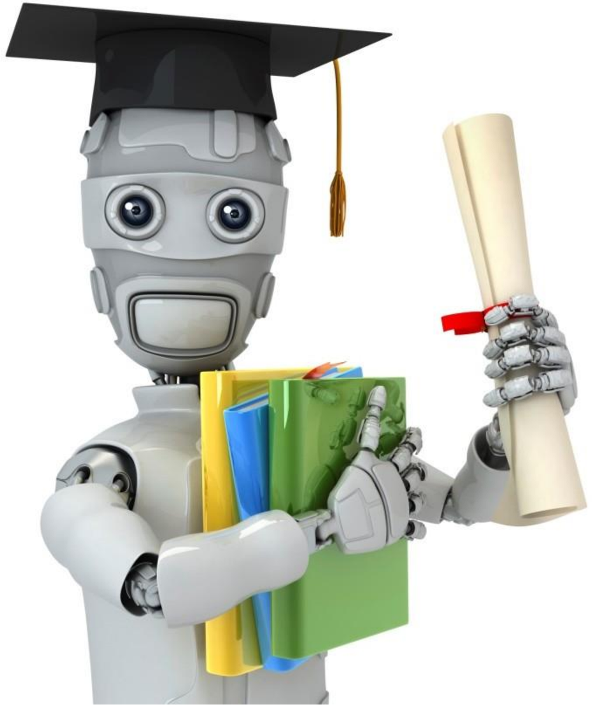
Which of the following statements about stochastic gradient descent are true? Check all that apply.

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

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

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

[✓ 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 α is poorly tuned.]



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 $\underline{p(y = 1|x; \theta)}$ to optimize price.

price

logistic regression

```
Repeat forever {
    Get $\boxed{(x,y)}$ corresponding to user.
    Update $\theta$ using $\boxed{(x,y)}$: ~~$(x^{(i)}, y^{(i)})$~~
        → $\theta_j := \theta_j - \alpha (h_\theta(x) - y) \cdot x_j$    $(j = 0, \ldots, n)$
    }
}
```

→ Can adapt to changing user preference.

Andrew Ng

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 $p(y = 1|x; \theta)$. 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; ...

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

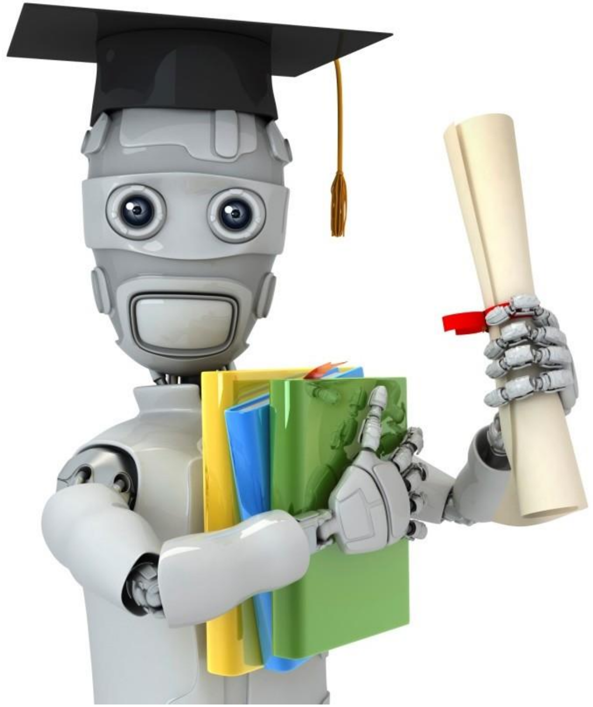
- [✓ It can adapt to changing user tastes (i.e., if $p(y|x;\theta)$ changes over time).]
- There is no need to pick a learning rate α.
- [✓ It allows us to learn from a continuous stream of data, since we use each example once then no longer need to process it again.]
- It does not require that good features be chosen for the learning task.

Recursive Least Squares

1. Initialize $\Gamma_0 = (I + \lambda I)^{-1}, w_0 = \mathbf{0}$;
2. Online iteration:

$$\Gamma_i = \Gamma_{i-1} - \frac{\Gamma_{i-1} x_i x_i^T \Gamma_{i-1}}{1 + x_i^T \Gamma_{i-1} x_i}$$

$$w_i = w_{i-1} - \Gamma_i x_i (x_i^T w_{i-1} - y_i)$$



Machine Learning

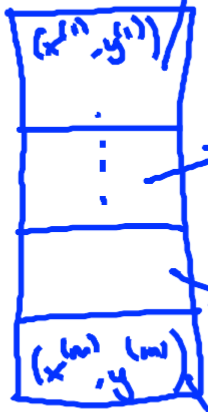
Large scale machine learning

Map-reduce and data parallelism

Map-reduce

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

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



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

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

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

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

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

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

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

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

Combine:

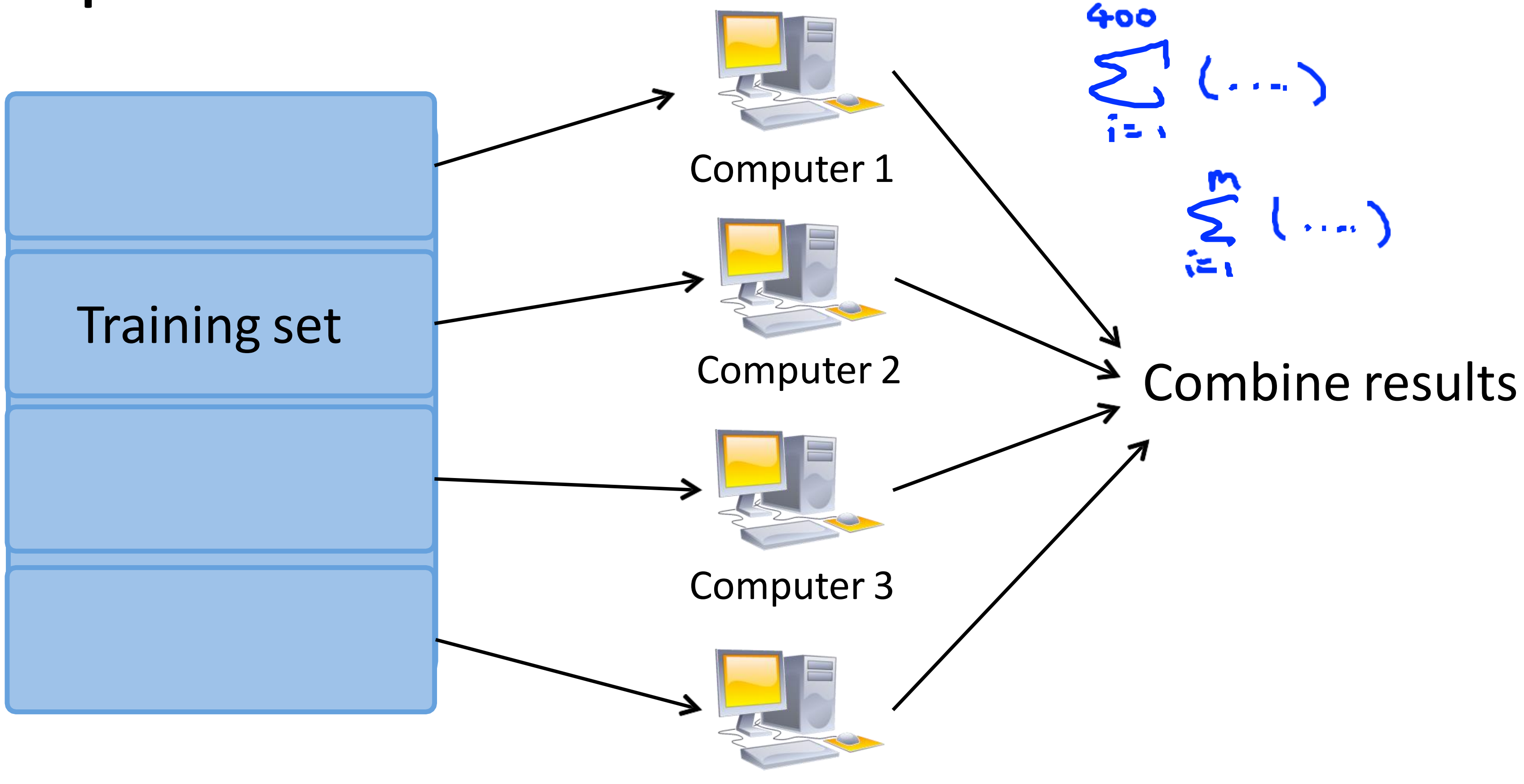
$$\boxed{\theta_j := \theta_j - \alpha \frac{1}{400} \left(temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)} \right)}$$

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

[Jeffrey Dean and Sanjay Ghemawat]

Andrew Ng

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:

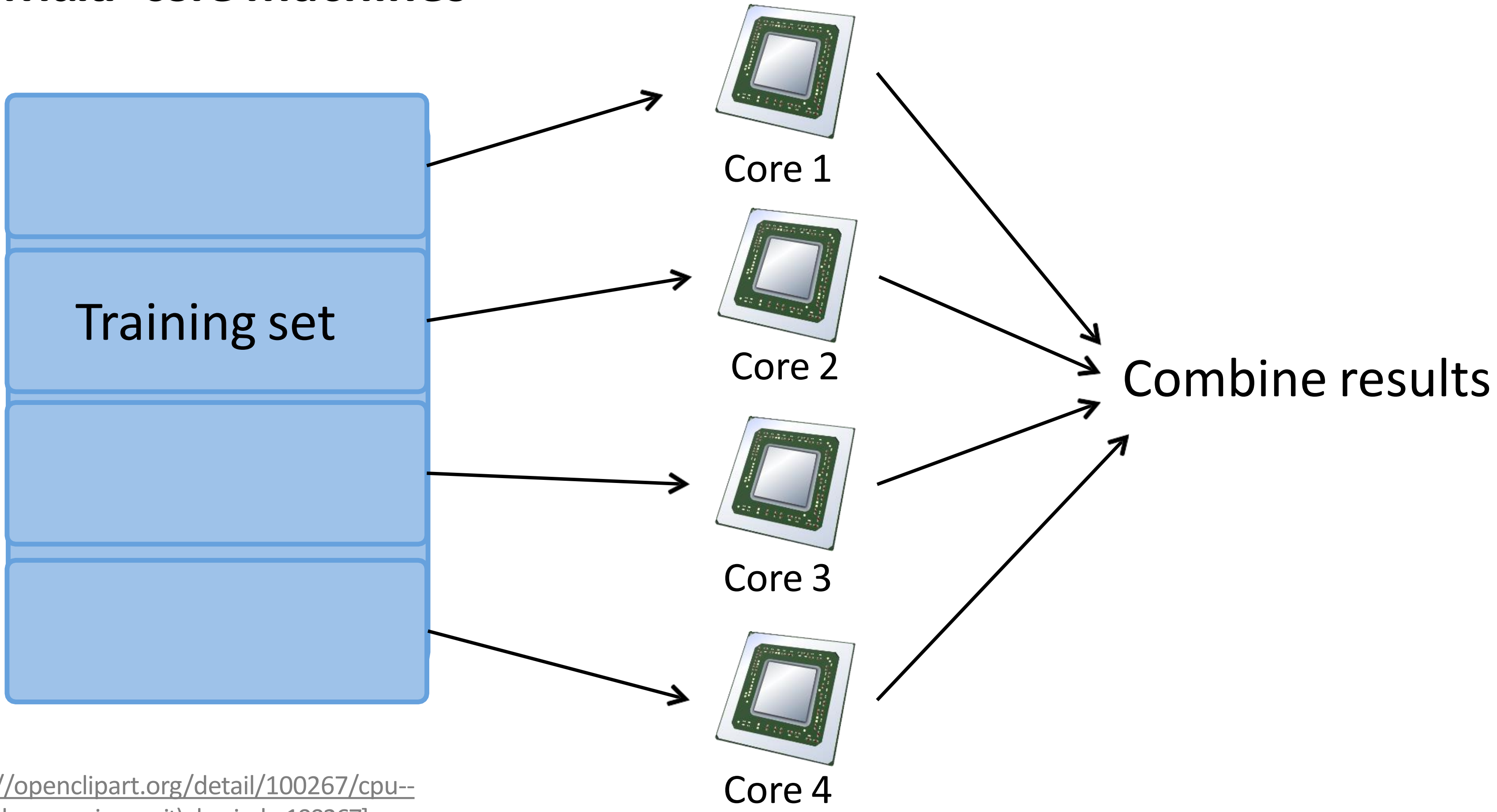
$$\rightarrow \underline{J_{train}(\theta)} = -\frac{1}{m}\sum_{i=1}^{m} \underline{y^{(i)}\log h_\theta(x^{(i)}) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))}$$

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

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

Andrew Ng

Multi-core machines



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

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.]
- ○ 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).