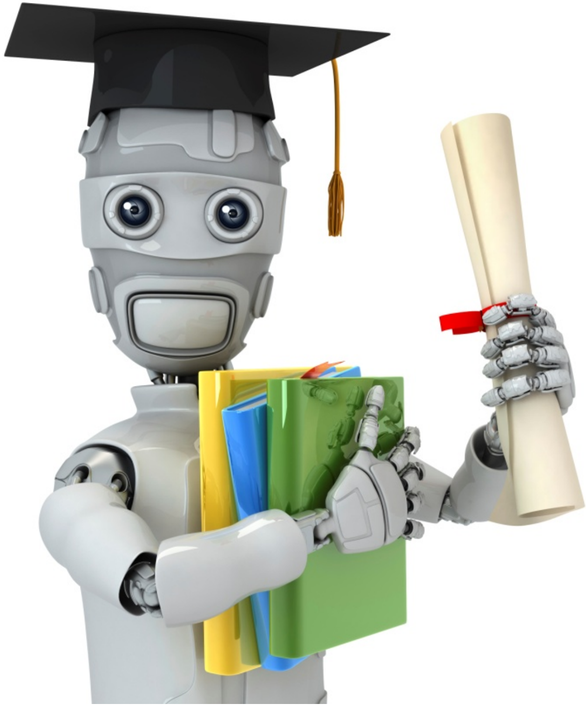


Machine Learning

Large scale machine learning

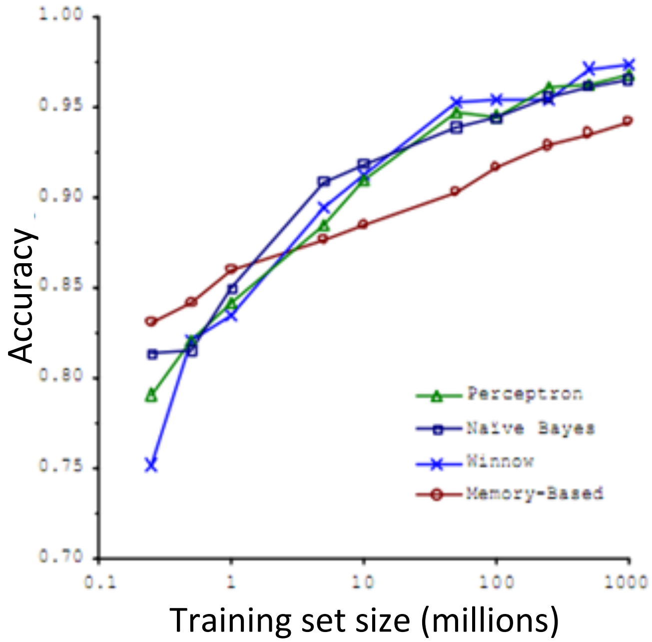
Learning with large datasets

Machine learning and data

Classify between confusable words.
E.g., {to, two, too}, {then, than}.

For breakfast I ate two______ eggs.

We've already seen that one of the best ways to get a high performance machine learning system, is if you take a low-bias learning algorithm, and train that on a lot of data.



"It's not who has the best algorithm that wins.
It's who has the most data."

[Figure from Banko and Brill, 2001]

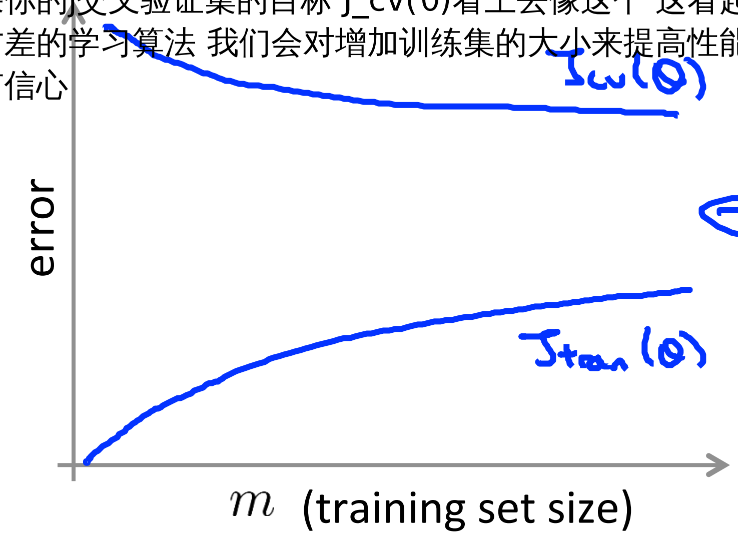
Learning with large datasets

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

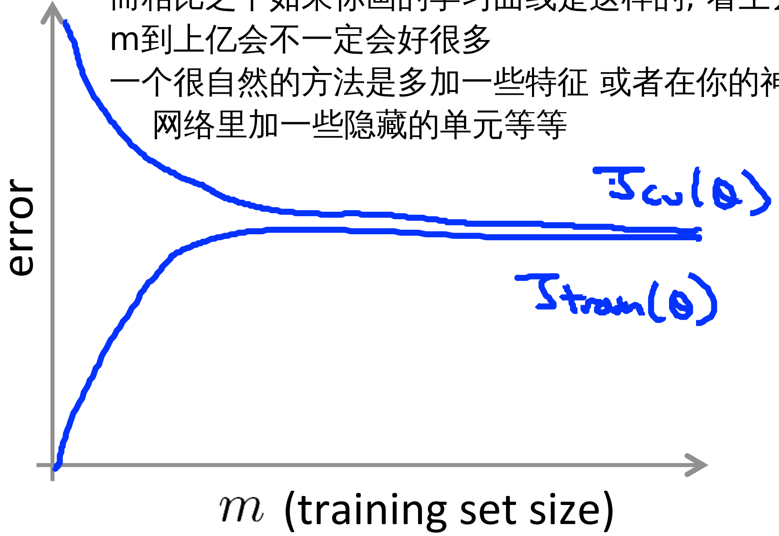
$m = 1{,}000?$

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

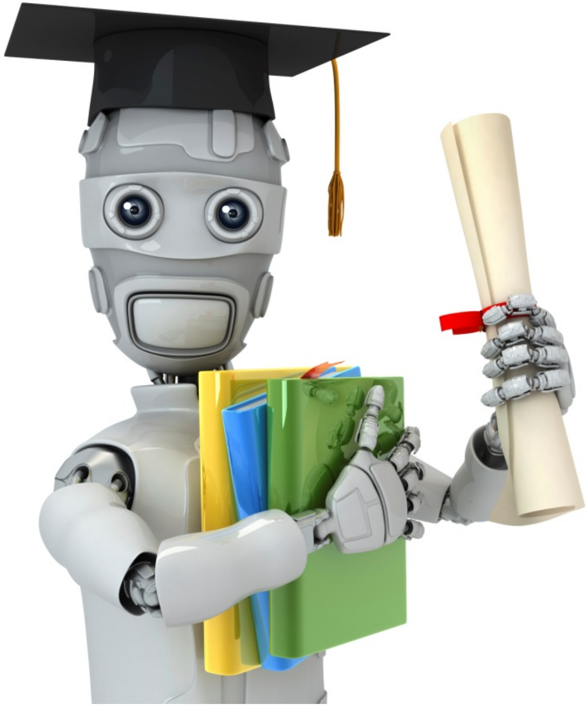
如果你的j交叉验证集的目标 J_cv(θ)看上去像这个 这看起来像高方差的学习算法 我们会对增加训练集的大小来提高性能更有信心



而相比之下如果你画的学习曲线是这样的，看上去增加m到上亿会不一定会好很多
一个很自然的方法是多加一些特征 或者在你的神经网络里加一些隐藏的单元等等



Andrew Ng



Machine Learning

Large scale machine learning

Stochastic gradient descent

Linear regression with gradient descent

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

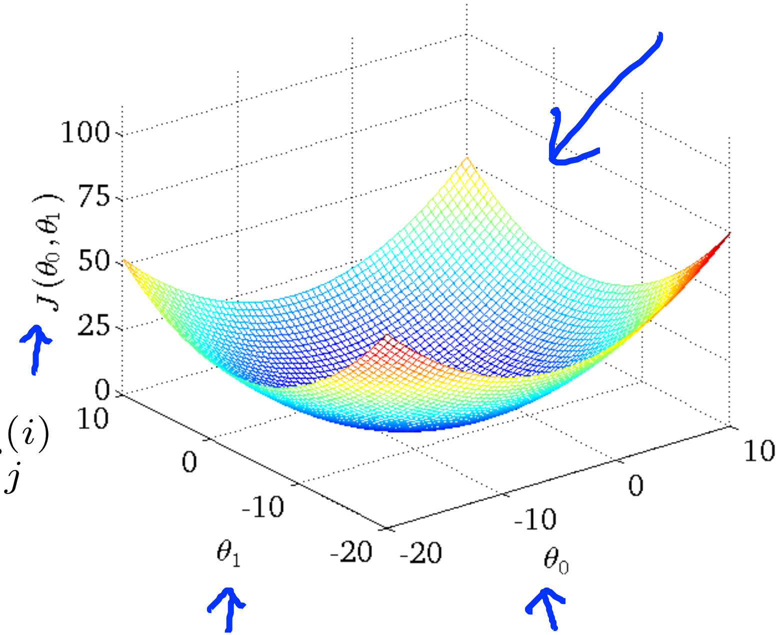
$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

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

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

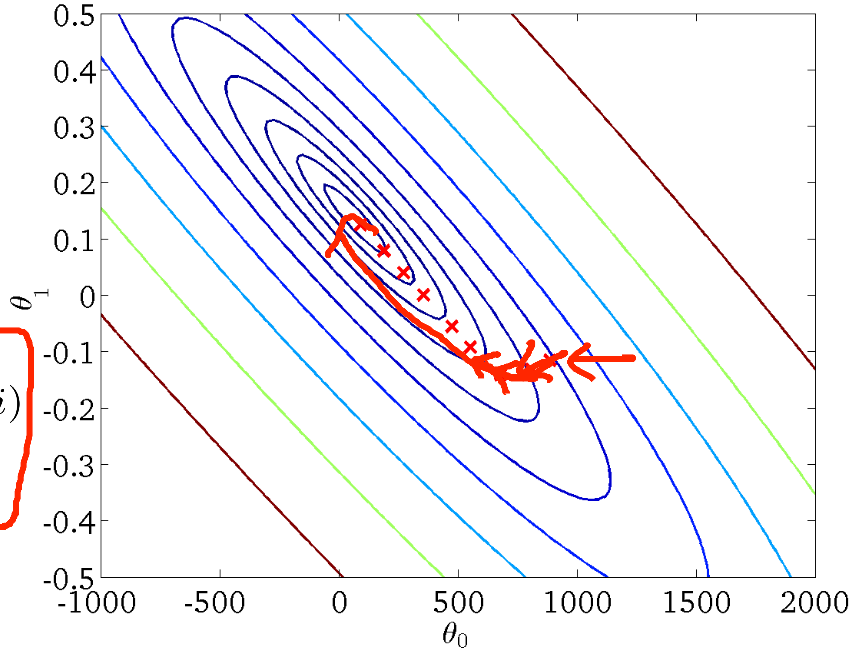
$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

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 (即普通的對所有收据求和的方法叫bactch grad...)

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 $j = 0, \ldots, n$)

}

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

外层循环(即那個repeat)应该执行多少次呢
这取决于训练样本的大小(下文意: 大的話就次數取少些)
通常一次就够了 最多到10次 是比较典型的

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.

将所有数据打乱(即重新排列)

2. Repeat {

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

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

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

}

}

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

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

Andrew Ng

Stochastic gradient descent

1. Randomly shuffle (reorder) training examples

2. Repeat { 1-10x

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

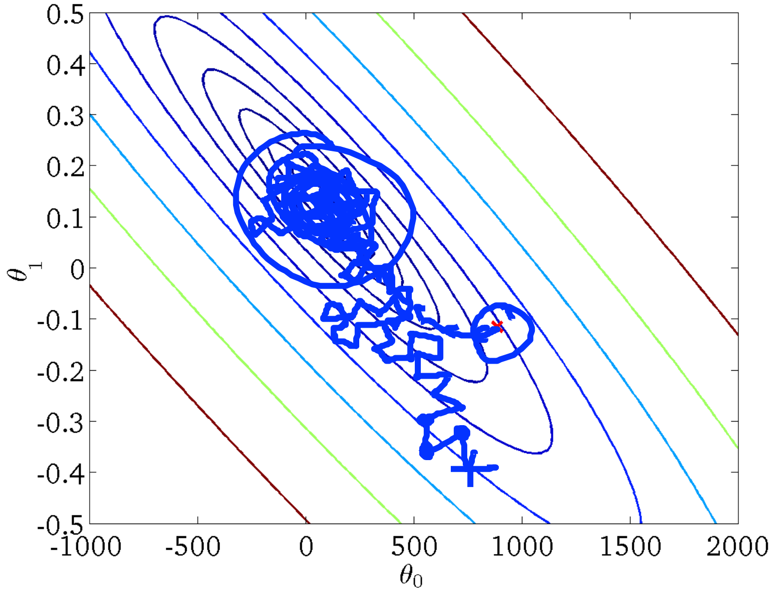
    $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

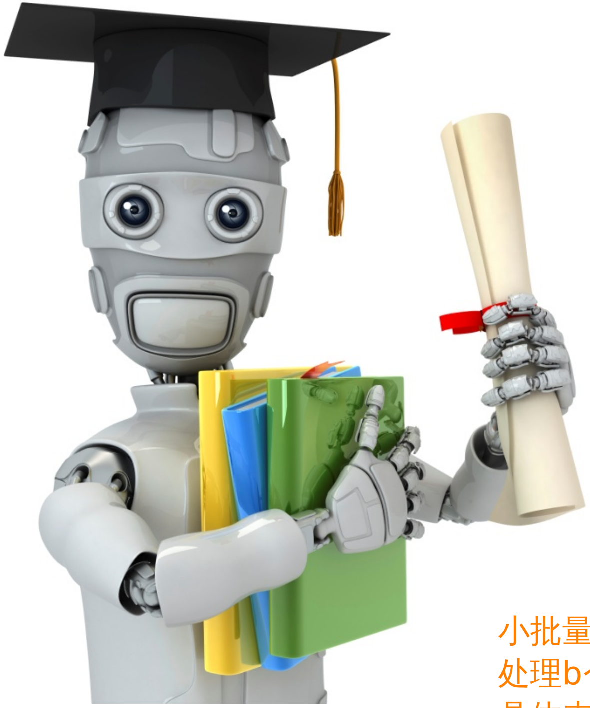
    (for every $j = 0, \ldots, n$)

    }

}

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





Machine Learning

Large scale machine learning

Mini-batch gradient descent

小批量梯度下降和随机梯度下降比较又怎么样呢？ 也就是说 为什么我们想要每次处理b个样本 而不是像随机梯度下降一样每次处理一个样本？ 答案是——向量化！具体来说 小批量梯度下降可能比随机梯度下降好 仅当你有好的向量化实现时在那种情况下 10个样本求和可以用一种更向量化的方法实现 允许你部分并行计算10个样本的和 因此 换句话说 使用正确的向量化方法计算剩下的项 你有时可以使用好的数值代数库来部分地并行计算b个样本 然而如果你是用随机梯度下降每次只处理一个样本 那么你知道 每次只处理一个样本没有太多的并行计算 至少并行计算更少

小批量梯度下降的一个缺点是有一个额外的参数b 你需要调试小批量大小 因此会需要一些时间 但是如果你有一个好的向量化实现这种方法有时甚至比随机梯度下降更快

Mini-batch gradient descent (这种算法有时候甚至比 随机 梯度下降还要快一点)

→ 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 $j = 0, \ldots, n$)

}

}

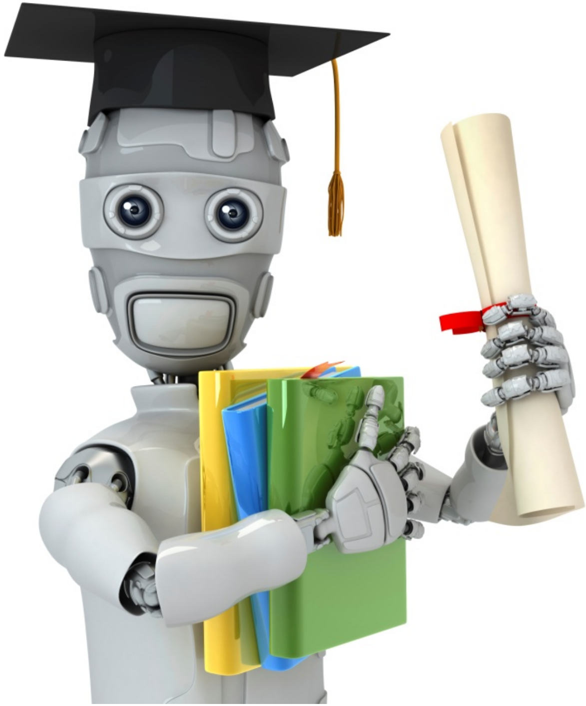
$\rightarrow$ b examples

$\rightarrow$ 1 example

Vectorization

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

$b = 10$



Machine Learning

Large scale machine learning

Stochastic gradient descent convergence

你如何确保调试过程已经完成 并且能正常收敛呢？

我们确定梯度下降已经收敛的一个标准方法 是画出最优化的代价函数关于迭代次数的变化 这就是代价函数 我们要保证这个代价函数在每一次迭代中 都是下降的

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

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

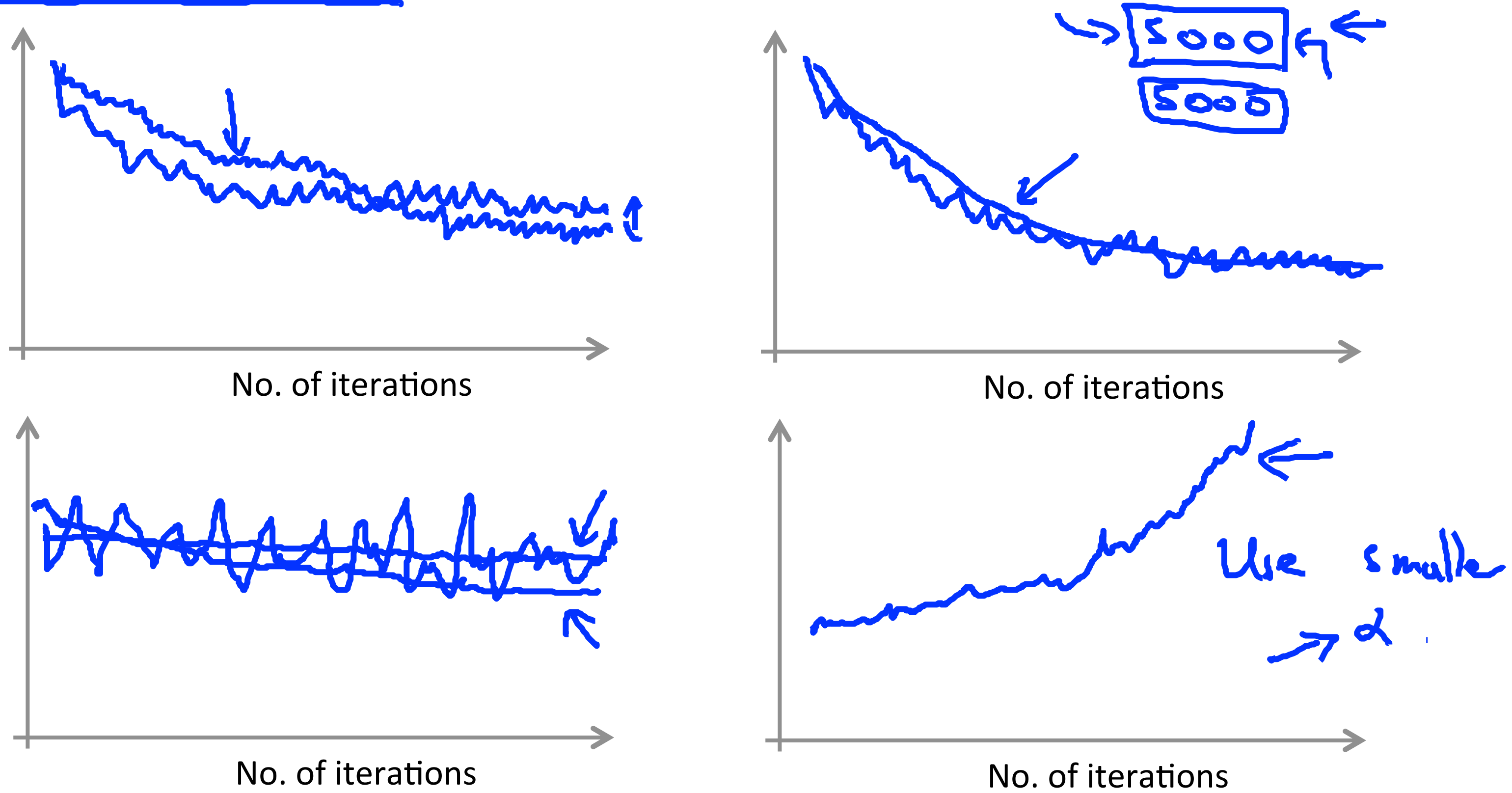
- Every 1000 iterations (say), plot $cost(\theta,(x^{(i)},y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.

Andrew Ng

如果曲线看起来噪声较大 或者老是上下振动 那就试试增大你要平均的样本数量 这样应该就能得到比较好的变化趋势
如果你发现代价值在上升 那么就换一个小一点的α值

Checking for convergence

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

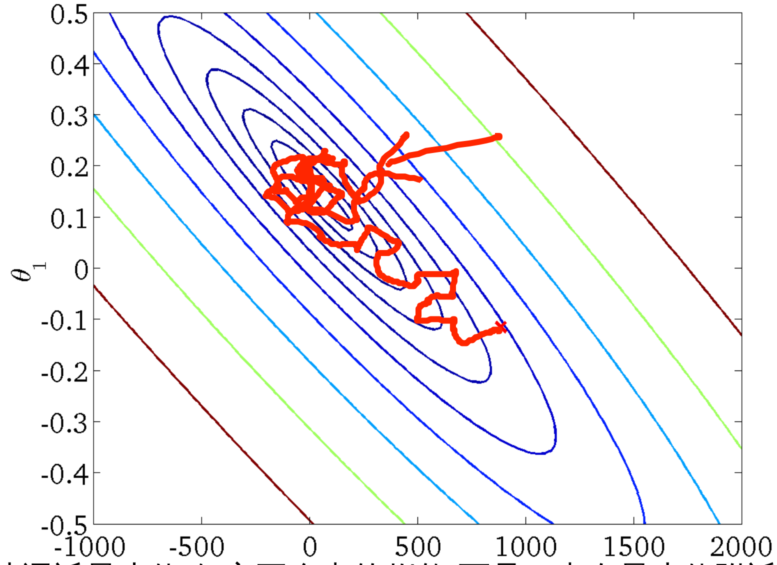


别忘了 随机梯度下降不是直接收敛到全局最小值 而是在局部最小附近反复振荡 所以使用一个更小的学习速率 最终的振荡就会更小

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 $j = 0, \ldots, n$)

    }

}

当运行随机梯度下降时 算法会从某个点开始 然后曲折地逼近最小值 但它不会真的收敛 而是一直在最小值附近徘徊 因此你最终得到的参数 实际上只是接近全局最小值 而不是真正的全局最小值 在大多数随机梯度下降法的典型应用中 学习速率α一般是保持不变的 如果你想让随机梯度下降确实收敛到全局最小值 你可以随时间的变化

减

速

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

但由于确定这两个常数需要更多的工作量 并且我们通常也对 能够很接近全局最小值的参数已经很满意了 因此我们很少采用逐渐减小α的值的方法

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 {

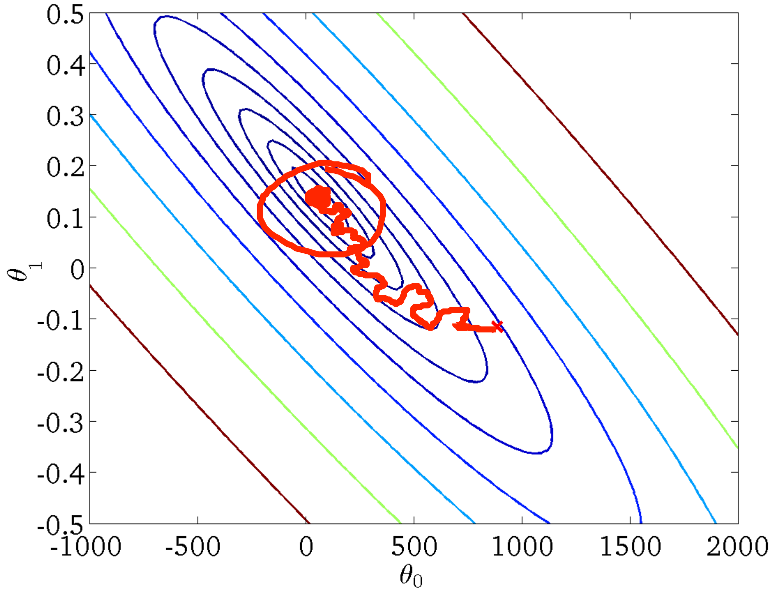
    for $i := 1, \ldots, m$ {

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

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

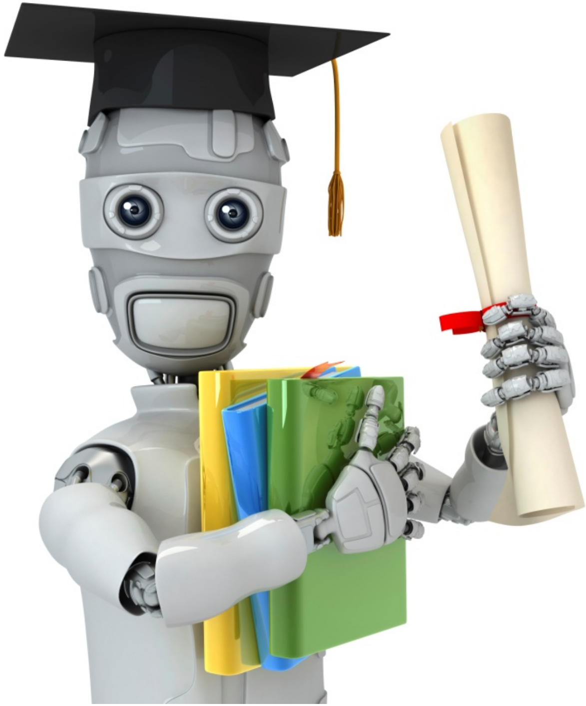
    }

}



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

Andrew Ng



Machine Learning

Large scale machine learning

Online learning

在拥有连续一波数据 或连续的数据流涌进来 而我们又需要 一个算法来从中学习的时候来模型化问题

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 ($y = 0$).

根据 你开给用户的这个价格 用户有时会接受这个运输服务 那么这就是个正样本

有时他们会走掉 然后他们拒绝购买你的运输服务

Features x capture properties of user, of origin/destination and asking price. We want to learn $p(y = 1|x;\theta)$ to optimize price.

這才是price

對於概率, think of logistic reg 中的概率

p表示概率，而不是price

price

logistic regression

Repeat forever {

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

我们先来考虑逻辑回归

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

如果你的用户群(或其喜好)变化了 那么参数θ的变化与更新 会逐渐调适到 你最新的用户群所应该体现出来的 参数

→ $\theta_j := \theta_j - \alpha(h_\theta(x) - y) \cdot x_j$ $(j = 0, \dots, n)$

我们从样本中学习

我们再丢弃它

}

j表示分量

Repeat forever 这只是代表着我们的网站将会一直继续保持在线学习 (即對每一個來的用戶，都更新θ)

Can adopt to changing user preference

→ Can adopt to changing user preference.

Andrew Ng

我们想要 使用一种学习机制来学习如何 反馈给用户好的搜索列表
你有一个在线 卖电话的商铺

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.

概率

→ Learn $\boxed{p(y = 1|x; \theta)}$.

这个概率是
指用户将会
点进 某一个特定的手机的链接

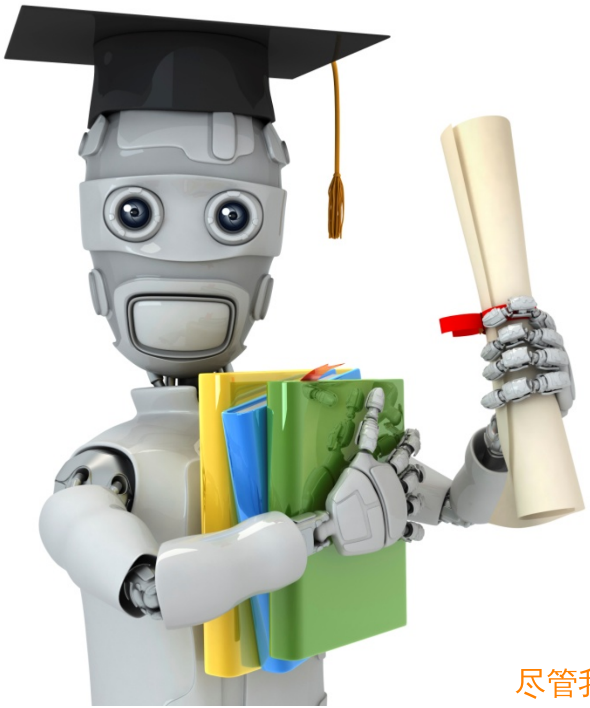
predicted

CTR

如果你能够估计 任意一个特定手机的点击率
我们可以做的就是 利用这个来 给用户展示十个
他们最有可能点击的手机

→ 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

尽管我们 用了多个视频讲解 随机梯度下降算法 而我们将只用少量时间
介绍MapReduce. 但是请不要根据 我们所花的时间长短 来判断哪一种技术
更加重要 事实上 许多人认为MapReduce方法至少是 同等重要的
还有人认为映射化简方法 甚至比梯度下降方法更重要

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

Map-reduce

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

$(x^{(1)}, y^{(1)})$

$\vdots$

$(x^{(m)}, y^{(m)})$

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

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

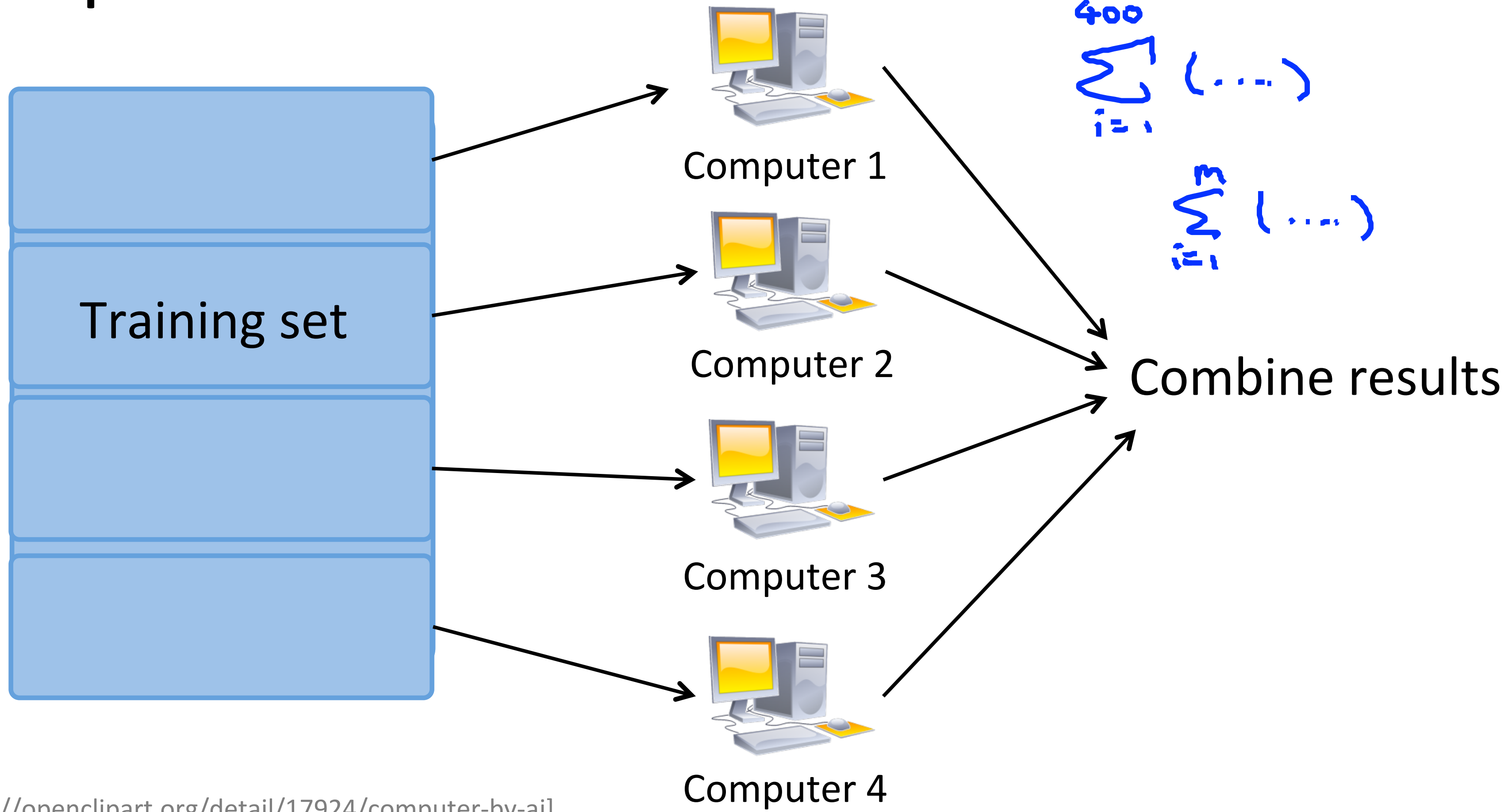
Combine:

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

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

[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]

Andrew Ng

如果你打算用MapReduce, 你需要问自己一个很关键的问题: 你的机器学习算法 是否可以表示为训练样本的某种 *求和* 事实证明

很多机器学习算法 的确可以
表示为关于训练样本的函数

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:

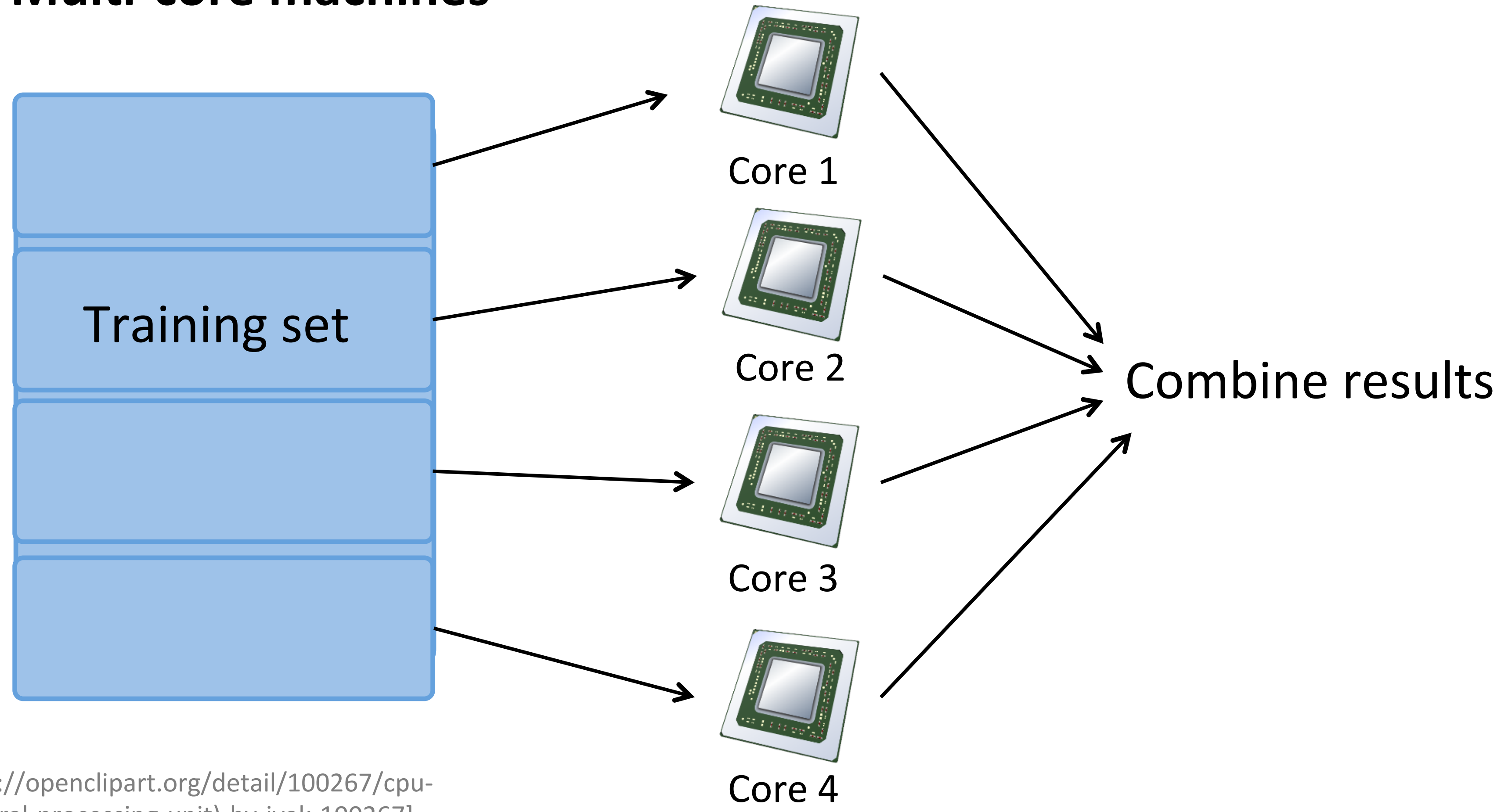
$$\rightarrow \underline{J_{train}(\theta)} = -\frac{1}{m}\sum_{i=1}^{m} \underline{y^{(i)} \log h_\theta(x^{(i)}) - (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))}$$

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

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

有时即使我们只有一台计算机 我们也可以运用这种技术
具体来说 现在的许多计算机 都是多核的 你可以有多个CPU 而每个CPU 又包括多个核

Multi-core machines



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