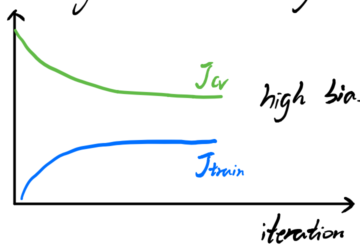
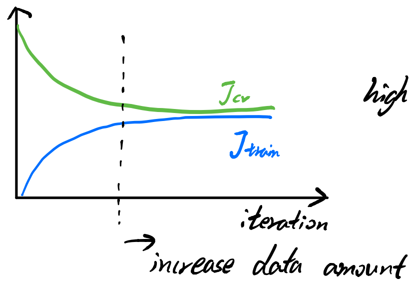
Learning with large dataset

when algorithm has high bias, more data won't benefit

when algorithm has high variance, more data lead to more accurate



high bias



high variance

Stochastic Gradient Descent:

Batch gradient descent VS **stochastic gradient descent**

Batch gradient descent

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

Repeat {

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

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

$\frac{\partial}{\partial \theta_j} J_{train}(\theta)$

}

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. random shuffle dataset
2. Repeat { 1~10 times

   for $i = 1, 2, \cdots, m$ {

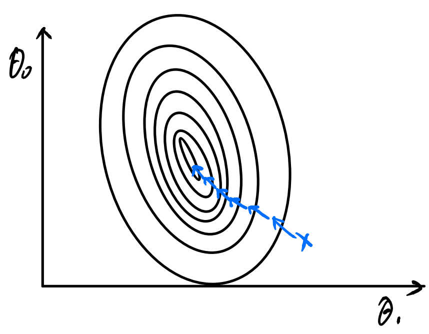
   $$\theta_j := \theta_j - \alpha \boxed{(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}}$$

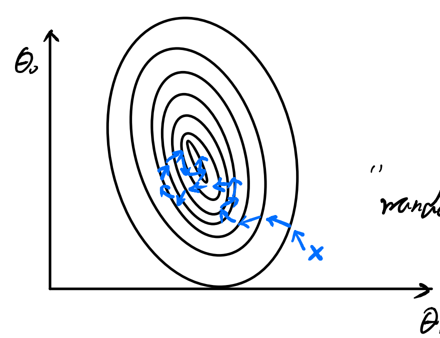
   (for $j = 0, 1, \cdots, n$)

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

   }

}





"wander around"

stochastic gradient descent is more efficient to large dataset

Mini-Batch Gradient Descent

even faster than stochastic gradient descent, instead using only one example each iteration, mini-batch use some in-between number of examples b ($1<b<m$) $b: 2 \sim 100$

e.g. $b=10$, $m=1000$

Repeat {

for $i = 1, 11, \cdots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)} \quad \text{(for every } j\text{)}$$

}

}

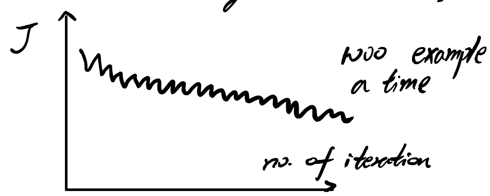
compute more than one example can take advantage of vectorization to speed up.

Stochastic Gradient Descent Convergence

— debug whether stochastic gradient descent is close to global optima

stochastic gradient descent will oscillate and jump around the global mininum, so choose a smaller learning rate.

plot average cost every 1000 examples, the more number of the examples or the smaller learning rate α is, the more smooth the curve can be



slowly decrease α overtime :

$$\alpha = \frac{const1}{iterationNumber + const2}$$

Online Learning

with a continuous steam of training example (e.g. when user click a zone (x, y)), we can run endless loop , using stochastic gradient descent, when an example is used it won't be stored or use again.

You can update θ continuously , adapt to the change of user habit.

Map Reduce and Data parallelism

we can divide up batch gradient descent and dispatch the cost function for a subset of data to many different machines so we can train our algorithm in parallel.

e.g.

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

map

$$t_1 = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$t_2 = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$t_3 = \sum_{i=201}^{300} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$t_4 = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\xrightarrow{\text{reduce}} \theta_j := \theta_j - \alpha \frac{1}{400} (t_1 + t_2 + t_3 + t_4)$$

also can be applied to single computer with multiple core