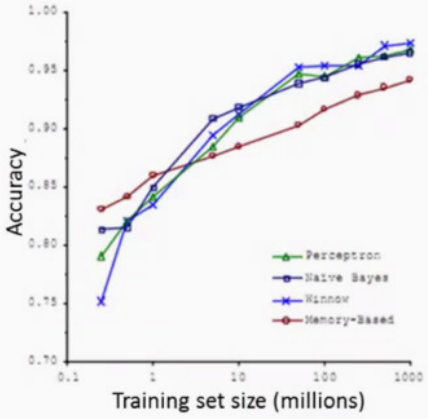
large scale machine learning



learning with large datasets

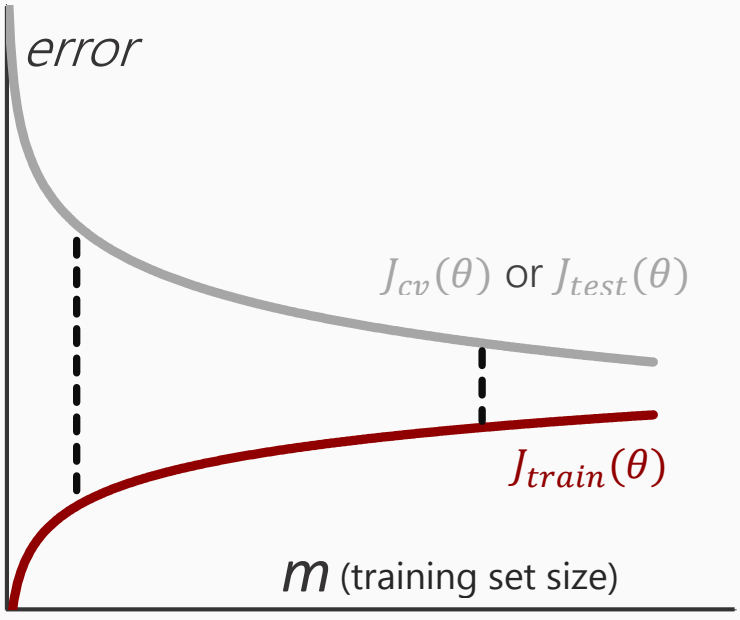
the importance of accessibility of large amounts of data, it has been noted over time that it is not necessarily who has the best algorithm that can solve a problem, but who has the most data

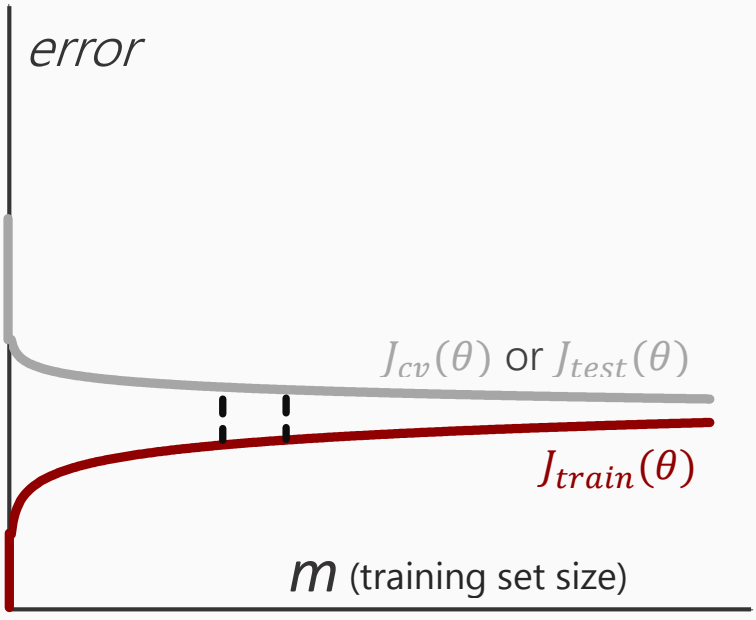
many cases provide datasets with millions of records

m = 100,000,000

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

the model for running iterations of gradients descent consequentially becomes very computationally expensive to implement. a sanity check should determine if the algorithm will work equally well on a much smaller subset of the data → m = 1,000. this can be evaluated by determining that the data still contains a high variance (left illustration) in a smaller sample but does not experience a high bias (right illustration) when m = 1,000





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 Response

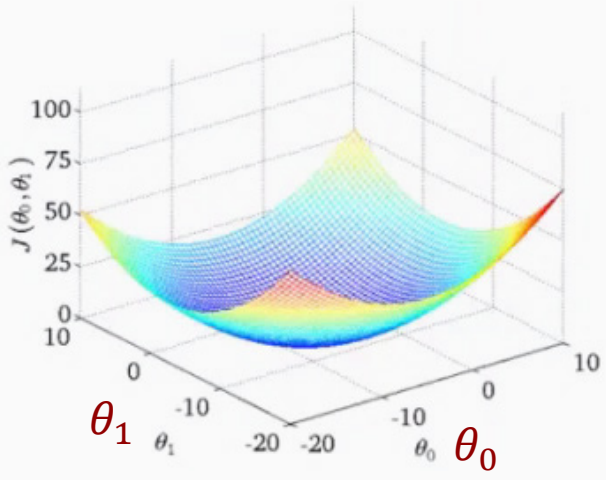
stochastic gradient descent

stochastic gradient descents allows an algorithm to scale to larger datasets

linear regression with gradient descent

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

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

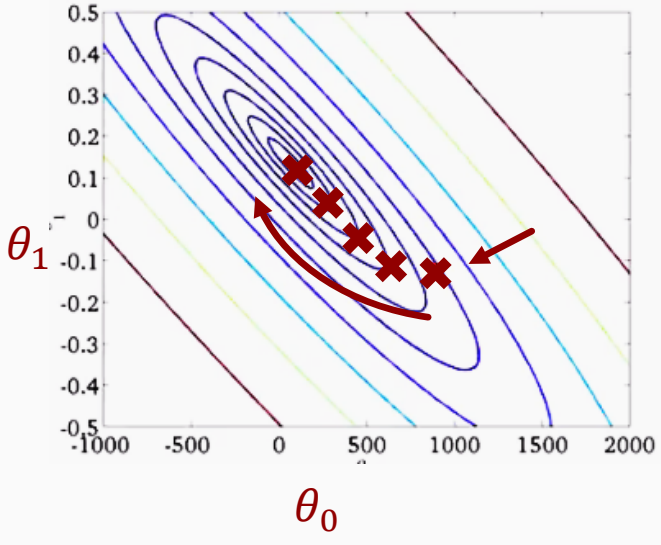


repeat {

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

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

}



reiterating to find the global minimum

in the circumstance where $m = 300{,}000{,}000$, the method of **'batch gradient descent'** implies that all 300,000,000 iterations would be stored in the machine's memory in order to take the first step towards finding the global minimum in gradient descent

a separate algorithm that applies a single training example in one iteration as opposed to every training example in all iterations addresses computational cost of the above

batch gradient descent

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

repeat {

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

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

}

stochastic gradient descent

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

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

1. randomly shuffle dataset
2. repeat {

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

$$\theta_j := \theta_j - a\underbrace{\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}}_{\frac{\partial}{\partial\theta_j}\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)}$$

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

}

}

stochastic gradient descent scans through the training examples, first looking at the first example $\left(x^{(1)}, y^{(1)}\right)$ and taking a gradient descent step only in respects to the first example. the algorithm will proceed to the second training example $\left(x^{(2)}, y^{(2)}\right)$ and perform the same process of advancing another step in respects to the cost of the second example. similarly, the algorithm will proceed to the third training example $\left(x^{(3)}, y^{(3)}\right)$ in attempt to fit it slightly better, and so on... (random shuffling necessary)

the algorithm essentially takes the form of summation in batch gradient descent $\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$ and instead applies the metric in a single iteration $\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$ through stochastic gradient descent to benign making improvements to the algorithm with each individual iteration.

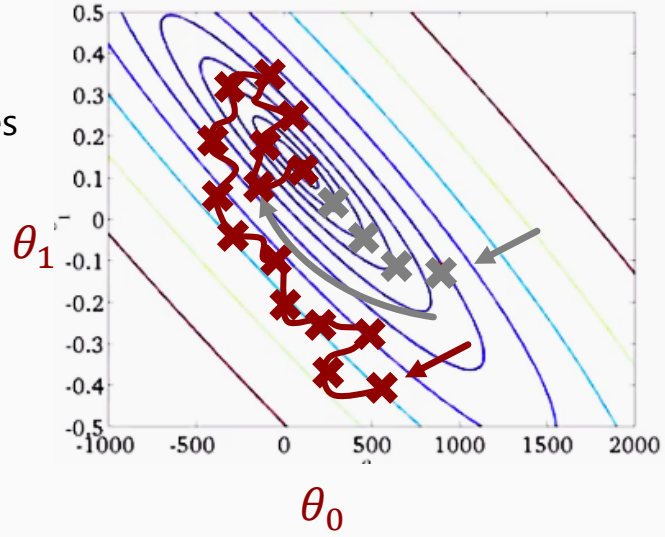
stochastic gradient descent

1. randomly shuffle (reorder) training examples
2. repeat {
   for $i = 1, \ldots, m$ {

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

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

   }
}



The nature of **stochastic gradient descent** does **not** converge in the same sense that **batch gradient descent** is expected to. stochastic gradient descent will tend to wander around the parameter space in efforts to find the global minimum as opposed to moving in a more linear fashion. Regardless, the algorithm will still arrive at a respectable hypothesis

if *m* is extraordinarily large (e.g. *m* = 300,000,000), the first iteration of stochastic gradient descent might actually be enough to form a suitable hypothesis forming a parameter that arrives at the global minimum (opposed to multiple reiterations)

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

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

Correct Response

- [x] 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.

Correct Response

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

Correct Response

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

Correct Response

mini · batch gradient descent

batch gradient descent: use all m examples in each iteration

stochastic gradient descent use 1 single example in each iteration

mini-batch gradient descent: use b examples in each iteration

$b = \text{mini} - \text{batch size}$ typically batches range from 2 – 100

in the case of 10 examples ($b = 10$)

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

$$\theta_j := \theta_j - a\frac{1}{10}\sum_{k=i}^{i+9}\left(h_\theta(x^{(k)}) - y^{(k)}\right)x_j^{(k)}$$

$$i := i + 10$$

when comparing stochastic gradient descent (1 example) to mini-batch gradient descent (b examples), the reason behind using mini-batch gradient descent is when **vectorization** is possible.

this allows the user of a numerical linear algebra library to **parallelize** the

formally: $b = 10$, $m = 1{,}000$

1. repeat {

for $i = 1, 11, 21, 31, \dots, 991$ {

$$\theta_j := \theta_j - a\frac{1}{10}\sum_{k=i}^{i+9}\left(h_\theta(x^{(k)}) - y^{(k)}\right)x_j^{(k)}$$

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

}

}

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 Response

- None of the above

stochastic gradient descent convergence

checking for gradient descent convergence

batch gradient descent:

plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent:

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

stochastic gradient descent:

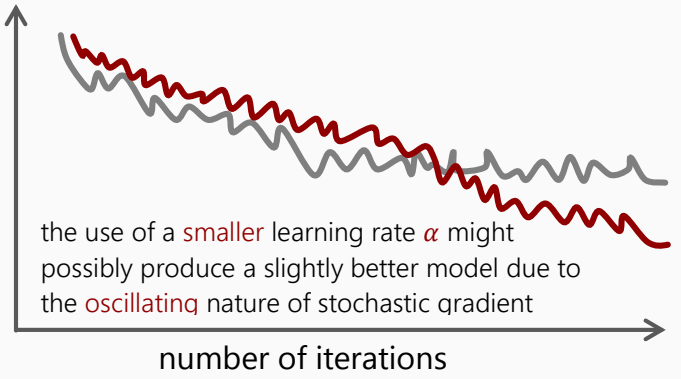
$$\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

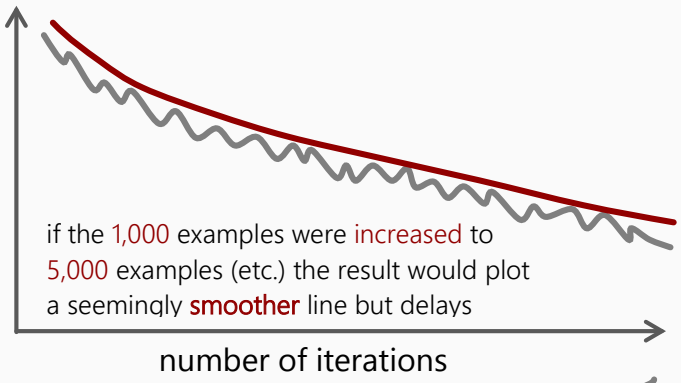
while the algorithm is learning, compute $\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)$ prior to updating θ using the training example $\left(x^{(i)}, y^{(i)}\right)$

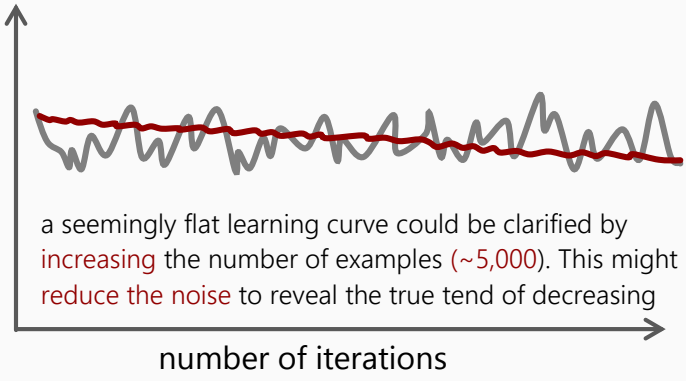
every 1,000 iterations (example), plot $\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)$ averaged over the last 1,000 examples processed by the algorithm

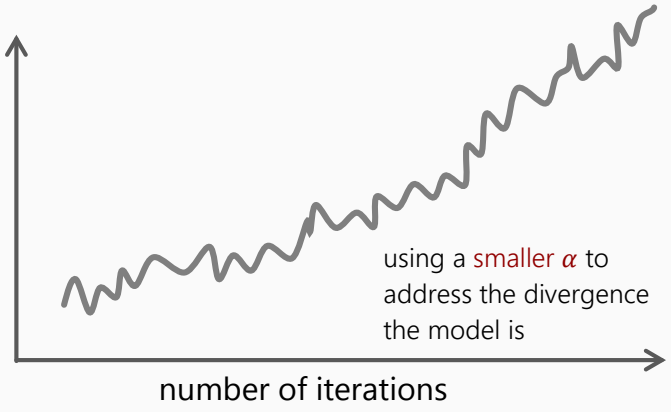
as stochastic gradient descent is scanning through the training set, before θ is updated using a particular training example $\left(x^{(i)}, y^{(i)}\right)$, compute how well the hypothesis is performing on training example $\left(x^{(i)}, y^{(i)}\right)$

plot $\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)$, averaged over the last 1,000 (example number) examples:



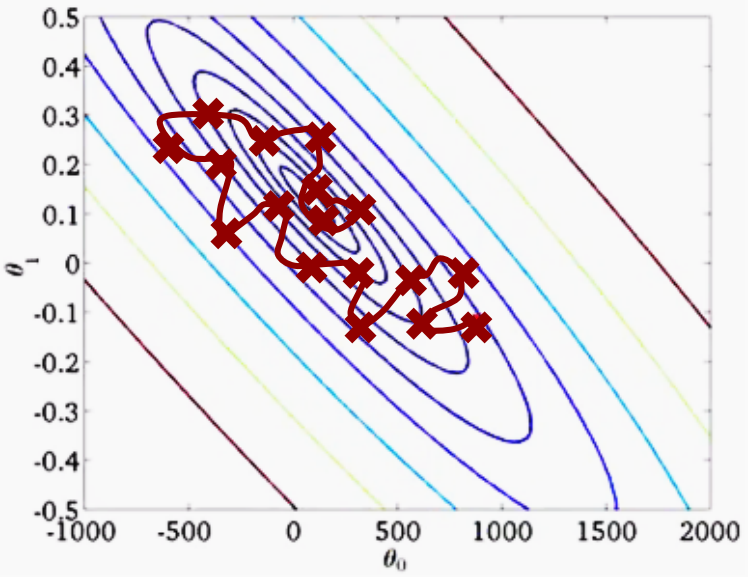






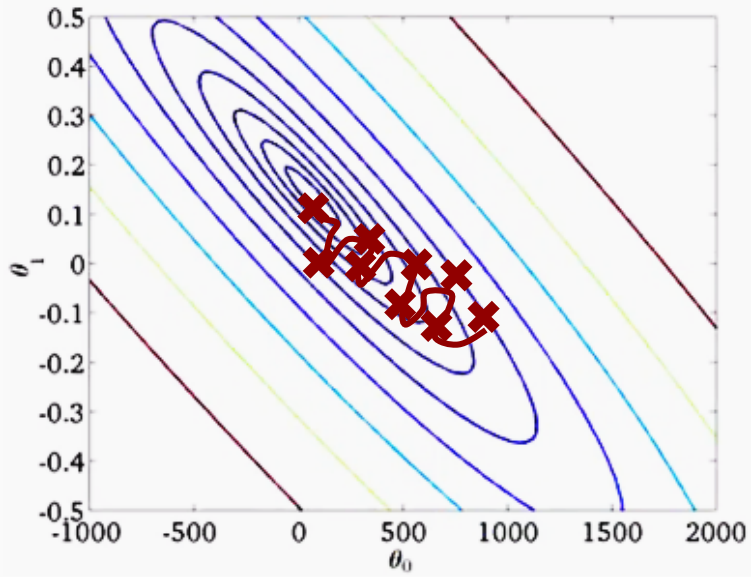
stochastic gradient descent

typical behavior of stochastic gradient descent is representative of the model wandering around the parameter space while converging but never actually settles on the global minimum. instead, the algorithm will essentially wander around the global minimum indefinitely (close convergence but not quite exact)



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

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



1. randomly shuffle (reorder) training examples
2. repeat {
   for $i = 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }

learning rate α is typically held constant. α can be slow decreased over time if θ is needing to converge (e.g. $\alpha = \frac{const1}{interationNumber + const2}$)

in effect, the algorithm with a properly adjusted α will experience smaller and smaller oscillations under convergence and arrive at an ultimately suitable global minimum

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.

Correct Response

- [x] 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.

Correct Response

- [ ] 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.

Correct Response

- [x] 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.

Correct Response

online learning

shipping example

shipping service website where a user comes, specifies the origin and destination for shipping. The package is offered to ship for a certain asking price, and the user sometimes choose to user the shipping service $(y = 1)$ or not $(y = 0)$

features x capture properties of the user, the origin, the destination, and the asking price. the aim is to learn $p(y = 1|x; \theta)$ to optimize the shipping price offered

logistic regression approach

repeat indefinitely {
obtain (x, y) corresponding to the user
update θ using (x, y): $(x^{(i)}, y^{(i)})$
$\theta_j := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \quad (j = 0, \dots, n)$
}

the notion of using (x, y) opposed to the typically denoted $(x^{(i)}, y^{(i)})$ is due to the presence of streaming data and no fixed training set being used. Each example in the expression is used for learning and subsequently discarded for the next examples received in real-time, or near real-time.

The significance of this type of online learning algorithm is the **adaptability** to changing user preferences.

product search example (learning to search)

users search for "android phone 1080p camera"

100 phones in the online store of which will **return 10 results** (*10 training examples per search)*

x = features of phone, how many words in user query match description, etc…

$y = 1$ if user clicks on link, $y = 0$ otherwise

learn $p(y = 1|x; \theta)$ → predicted click-through-rate (CTR)

algorithm used to show the user 10 phones they are most likely to click on

other examples

special offers to show user; news article selection; product recommendation, etc…

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

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

Correct Response

- [ ] There is no need to pick a learning rate α.

Correct Response

- [x] 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 Response

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

Correct Response

map reduce and data parallelism

map reduce

assuming a sample of $m = 400$ (actual application would imply a sample more relative to $m = 400{,}000{,}000$; reduced for formula clarity in the following example)

batch gradient descent:

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

machine 1: use $\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(100)}, y^{(100)}\right)$ $\quad \text{temp}_j^{(1)} = \sum_{i=1}^{100}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

machine 2: use $\left(x^{(101)}, y^{(101)}\right), \ldots, \left(x^{(200)}, y^{(200)}\right)$ $\quad \text{temp}_j^{(2)} = \sum_{i=101}^{200}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

machine 3: use $\left(x^{(201)}, y^{(201)}\right), \ldots, \left(x^{(300)}, y^{(300)}\right)$ $\quad \text{temp}_j^{(3)} = \sum_{i=201}^{300}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

machine 4: use $\left(x^{(301)}, y^{(301)}\right), \ldots, \left(x^{(400)}, y^{(400)}\right)$ $\quad \text{temp}_j^{(4)} = \sum_{i=301}^{400}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

combine:

$$\theta_j := \theta_j - a\frac{1}{400}\sum_{i=1}^{400}\left(\text{temp}_j^{(1)} + \text{temp}_j^{(2)} + \text{temp}_j^{(3)} + \text{temp}_j^{(4)}\right) \qquad (j = 0, \ldots, n)$$

map reduce is an applicable technique when the learning algorithm can be expressed as computing sums of functions over the training set