Introduction to Data Science

(Lecture 24: Optional)

Dr. Mohammad Pourhomayoun

Assistant Professor

Computer Science Department

California State University, Los Angeles

Machine Learning for Big Data

Large-Scale Machine Learning

- One of the main reasons that machine learning algorithms achieve much better results recently compared to say 5-10 years ago is **the massive datasets that we have for training now**.

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

Andrew Ng,
Prof. of Stanford University
Machine Learning Expert

Large-Scale Machine Learning

- However, learning from large datasets is **very challenging**. We need to handle computational complexity!

- **Example**: Suppose that we want to train a **linear (or logistic) regression** using gradient descent algorithm on ***m = 100 Million* samples**. We need to calculate a summation over 100 Million samples in EACH iteration of gradient descent!

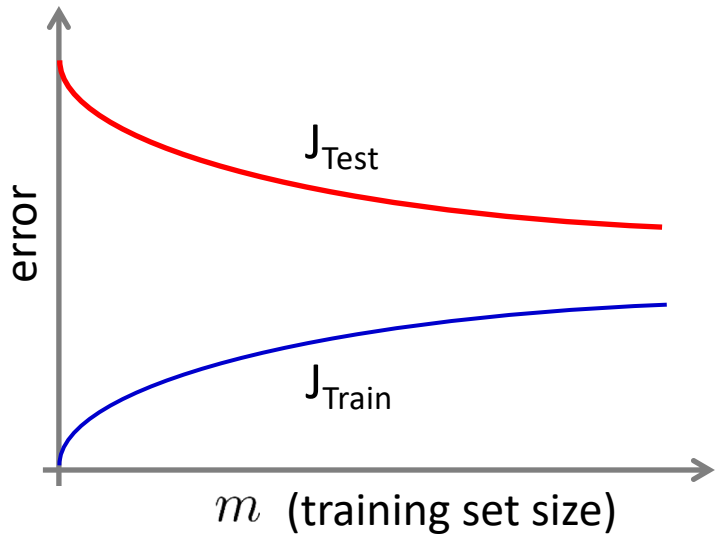
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)})\, x_j^{(i)}$$

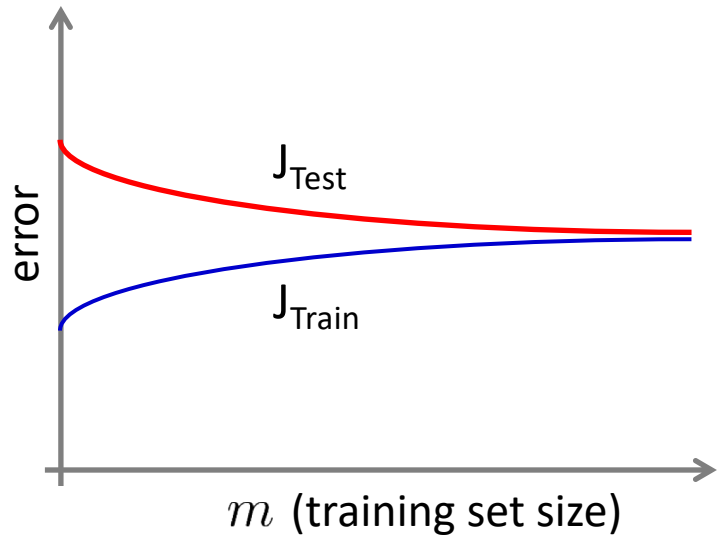
Large-Scale Machine Learning

- There are several approaches to handle learning from Big Data:
  1. Sampling the big dataset and only use the samples.
  2. Modifying the learning algorithms (e.g. Stochastic or Mini-batch Gradient Descent).
  3. Parallel Computing (MapReduce for ML).
  4. Dimensionality Reduction (or Feature Selection).

How Many Samples?

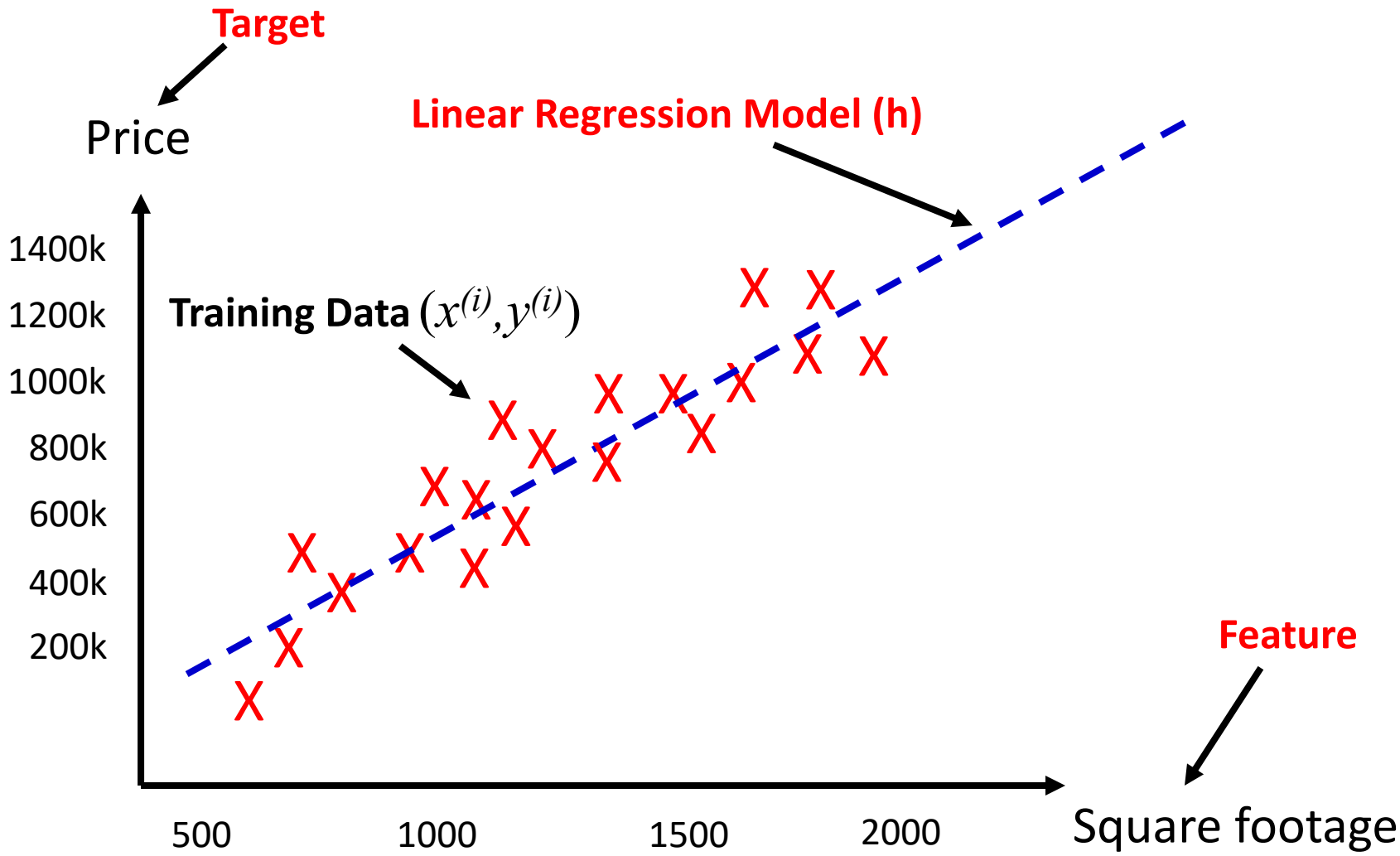
- **Sampling**: The simplest (but not the best) approach!
- If we want to randomly sample the dataset, and only use those samples in training rather than the entire dataset, how many samples should we select?





Stochastic Gradient Descent for Big Data

Review: Linear Regression

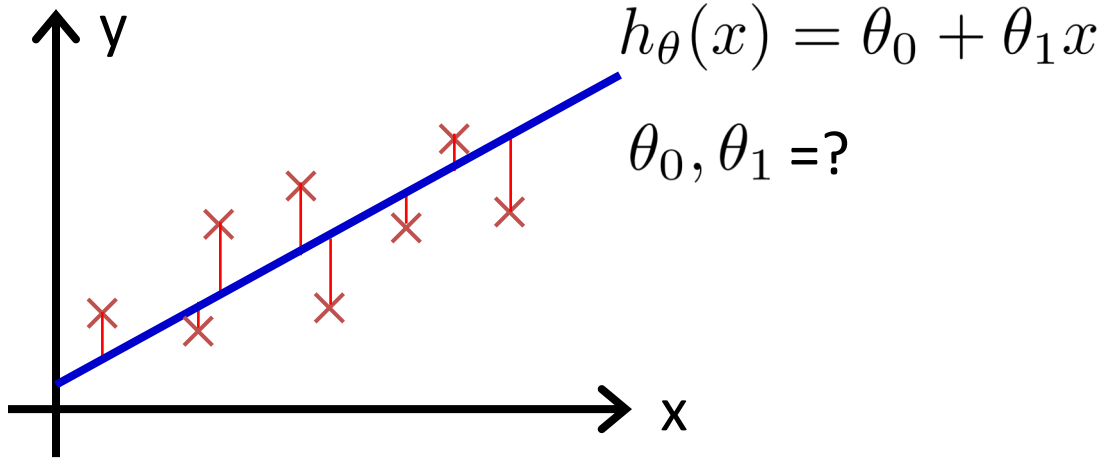


Review: How to Select the Parameters

- How to find the best Parameters θ_i's ?

Answer: We should find the **Best Fit** to the training samples. To do that, we have to minimize the differences between "prediction" and "actual value" in **training set** to find the best "fit" to our training data.

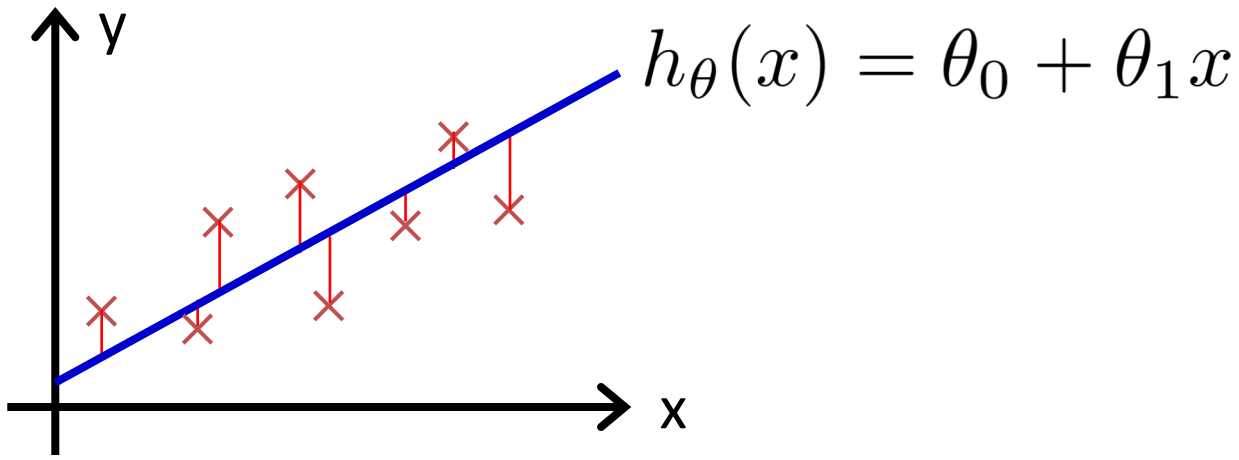
Training Set:



Let's define a **Cost Function $J(\theta_0, \theta_1)$** as the summation (or average) of all differences between "**training samples**" and the **"regression line"**.

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

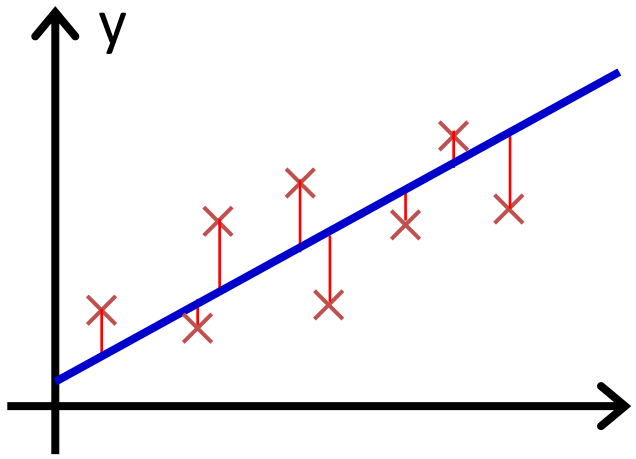
Thus, The Best Fit Line is the line with minimum Cost Function **$J(\theta_0, \theta_1)$.**



Training Set: $\{(x^{(1)},y^{(1)}), (x^{(2)},y^{(2)}), \ldots, (x^{(m)},y^{(m)})\}$

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$



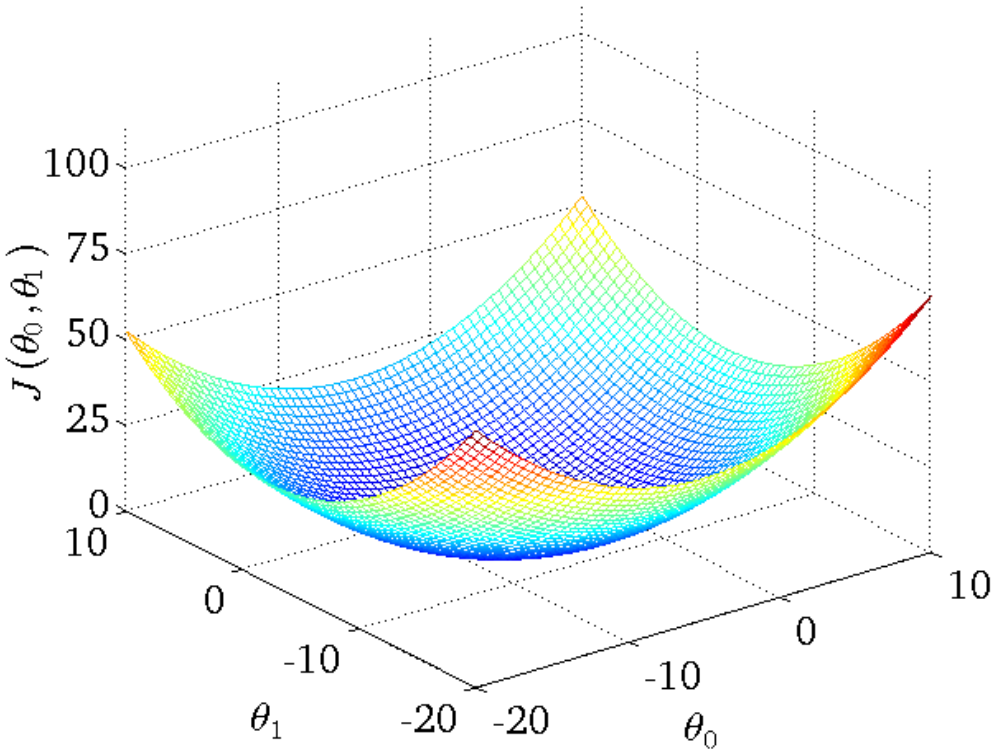
$h_\theta(x) = \theta_0 + \theta_1 x$

Regression Model: $h_\theta(x) = \theta_0 + \theta_1 x$

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Example: a convex cost function J:
(Convex means bowl-shaped!)



Regression Model: $h_\theta(x) = \theta_0 + \theta_1 x$

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Question:
How to minimize the cost function?

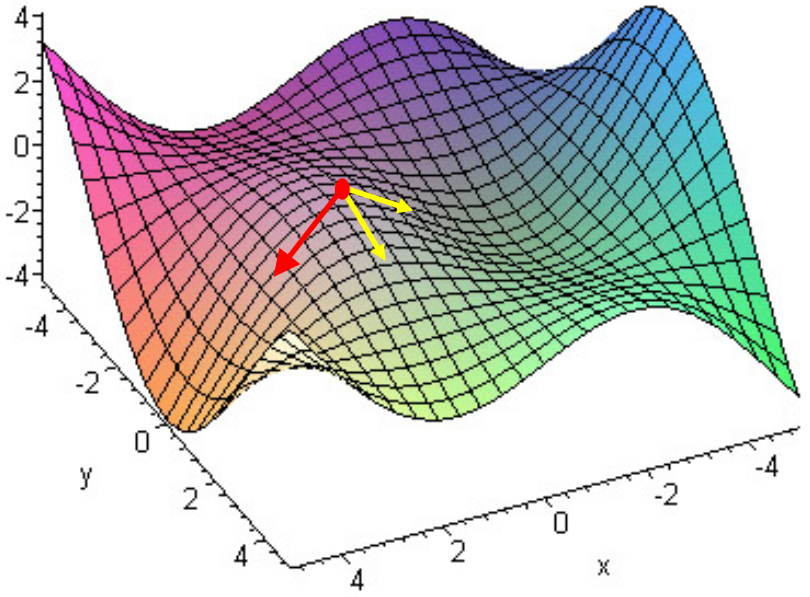
Answer:
We can use "**Gradient Descent**" algorithm to minimize the cost function and find the parameters.

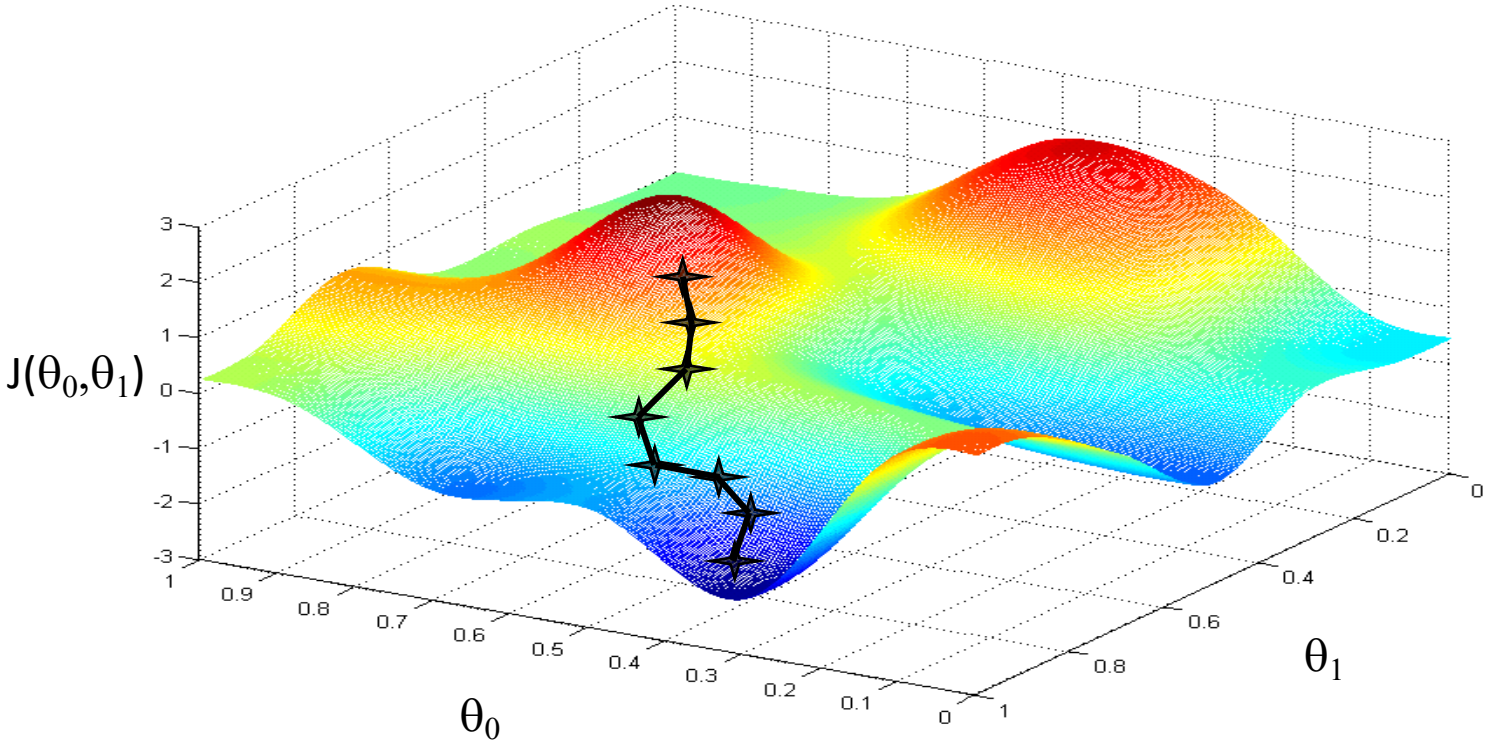
Review: Gradient Descent Algorithm*

- Gradient Descent is an iterative optimization approach that tries to minimize a function.

- The **gradient** is a generalization of the usual concept of **derivative** to functions of several variables.

- The **gradient represents the slope of the tangent of the graph of the function**.
- The **negative gradient** points in the **direction of the greatest rate of reduction** of the function, and its **magnitude** is the **slope** of the graph in that direction.

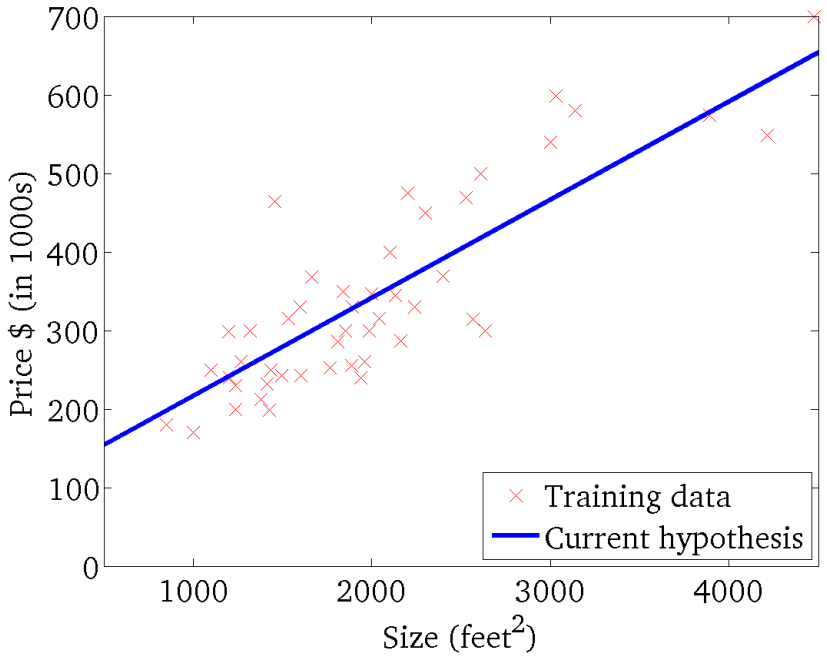




* Reference: Andrew Ng, Machine Learning, Stanford University.

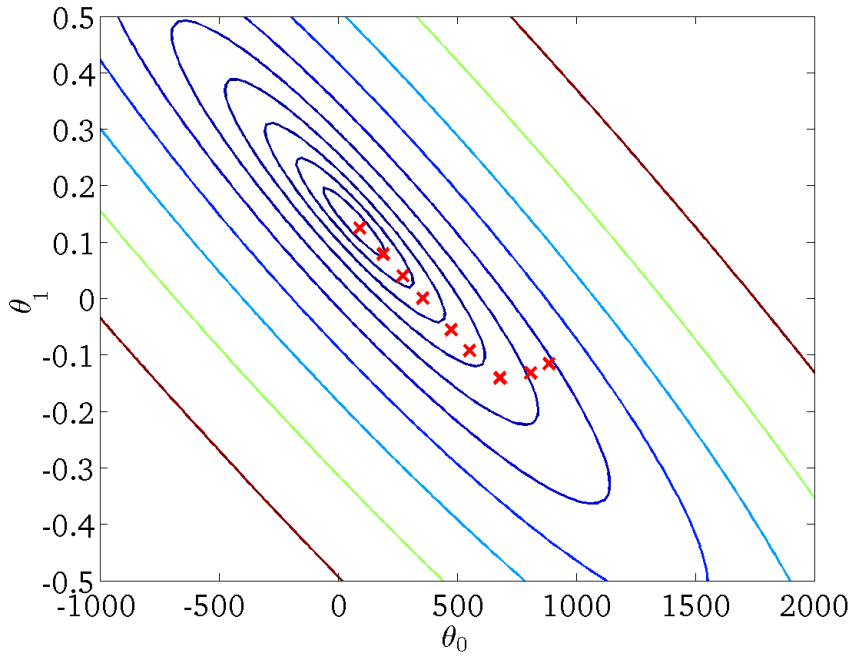
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Review: Gradient Descent Algorithm*

Linear regression with regular gradient descent

Also called Batch Gradient Descent

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

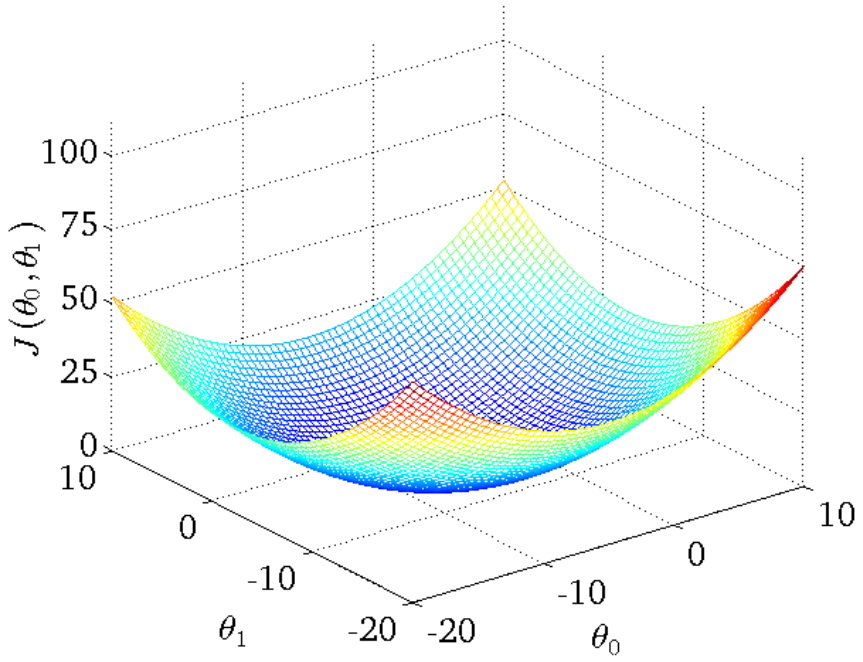
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}



* Reference: Andrew Ng, Machine Learning, Stanford University.

Review: Gradient Descent Algorithm*

Linear regression with regular gradient descent

Also called Batch Gradient Descent

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

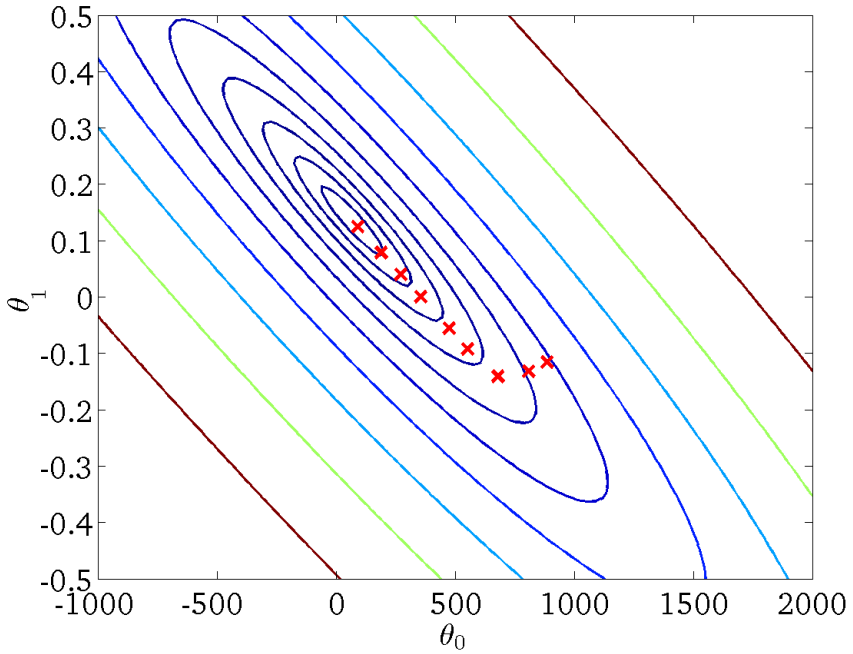
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}



* Reference: Andrew Ng, Machine Learning, Stanford University.

Regular (Batch) Gradient Descent

- In Regular Gradient Descent, we should wait to calculate all costs **over all m samples** and use the average of them to modify theta.
- Suppose that we want to train a **linear (or logistic) regression** using gradient descent algorithm on ***m = 100 Million* samples**. We need to calculate a summation over 100 Million samples in EACH iteration of gradient descent!
-

Repeat {

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

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

}

Stochastic Gradient Descent

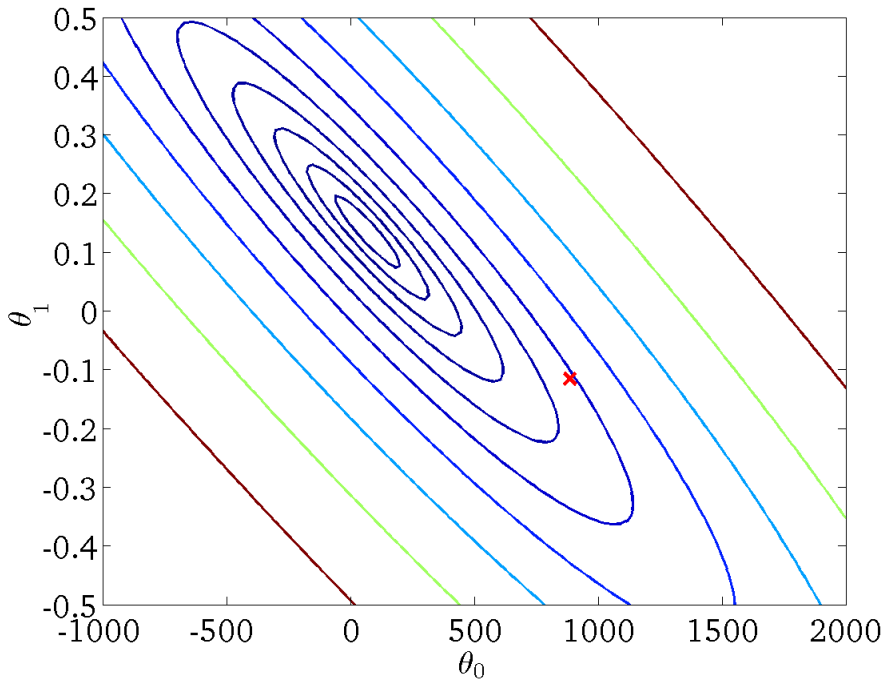
- In the Stochastic Gradient Descent, Rather than waiting to calculate all costs **over all m samples** and use the average of them to modify theta, we can start making some small progress after calculating each cost for each training sample!

Stochastic Gradient Descent

New Approach: Stochastic Gradient Descent:

1. Randomly shuffle (reorder) training examples

2. Repeat {
   for $i := 1, \ldots, m$ {
   $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$
   (for every $j = 0, \ldots, n$)
   }
   }



* Reference: Andrew Ng, Machine Learning, Stanford University.

Notes

- **Stochastic Gradient Descent** tries to take a small step after processing each individual data sample. In other word, rather than waiting to go over all data samples to form the summation and make a final step, it makes a small progress using each data sample individually.

- **Stochastic Gradient Descent** uses much higher number of steps to move toward the minimum (in some cases even in wrong directions), but overall it turns out to be much faster than Batch Gradient Descent.

- In some cases, it may never reach the global minimum. But, as long as it is close enough to the global minimum, that should be good enough for most applications.

Mini-Batch Gradient Descent for Big Data

- **Batch (Regular) Gradient Descent**: Use all data samples in each iteration:

Repeat {

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

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

}

- **Stochastic Gradient Descent**: Use 1 example in each iteration:

Repeat {

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

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

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

}

- **Mini-batch Gradient Descent**: Use b examples in each iteration!

Mini-Batch Gradient Descent

- **Mini-batch Gradient Descent**: Use b examples in each iteration:

Example: $b = 10, m = 1000.$

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 $j = 0, \ldots, n$)

}

}

Mini Batch Gradient Descent

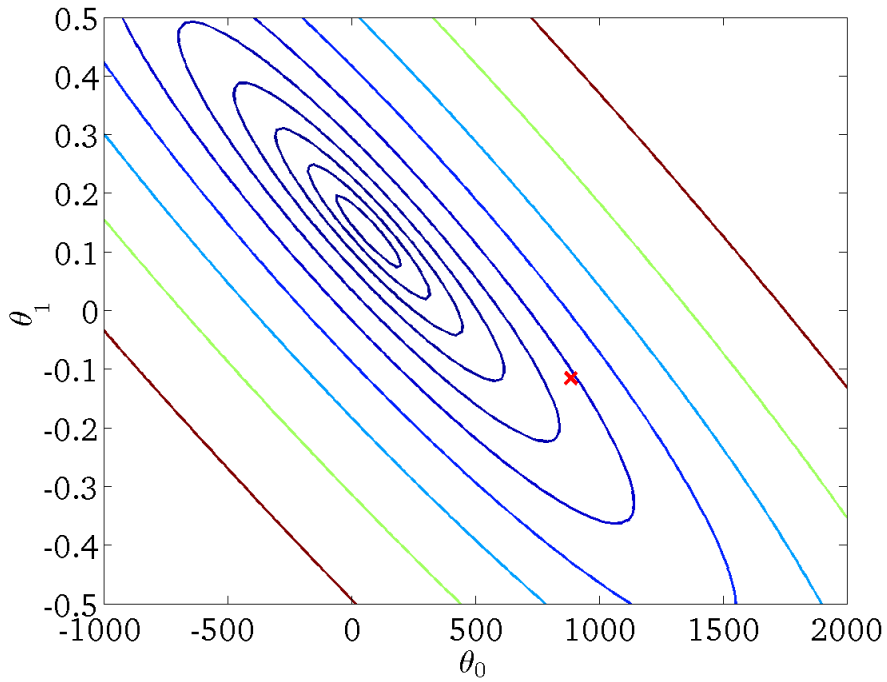
New Approach: Mini-Batch Gradient Descent:

1. Randomly shuffle (reorder) training examples

2. 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 $j = 0, \ldots, n$)
   }
   }



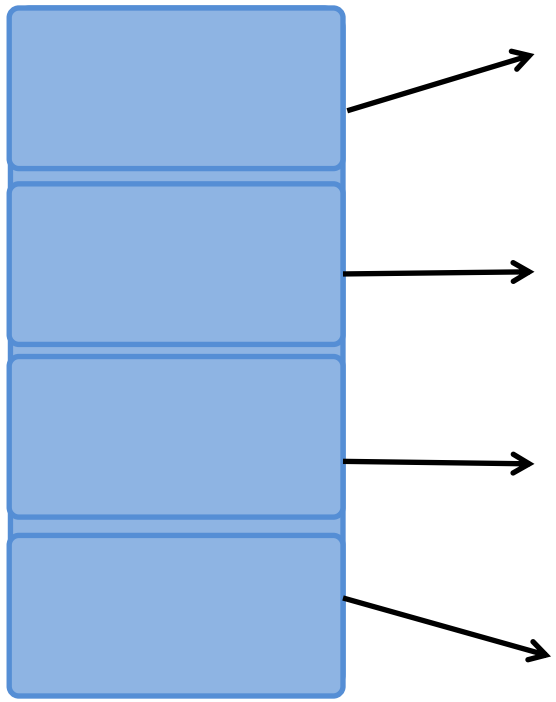
* Reference: Andrew Ng, Machine Learning, Stanford University.

Map-Reduce for Machine Learning

MapReduce for Batch Gradient Descent

- **Map Function**: Each mapper takes a chunk of data, and calculate the summation over that chunk.

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



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

$$temp_j^{(1)} = \sum_{i=101}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

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

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

MapReduce for Batch Gradient Descent

- **Reduce Function**: Combine the results and update theta.

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

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

$$temp_j^{(1)} = \sum_{i=101}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

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

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

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

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

Thank You!

Questions?