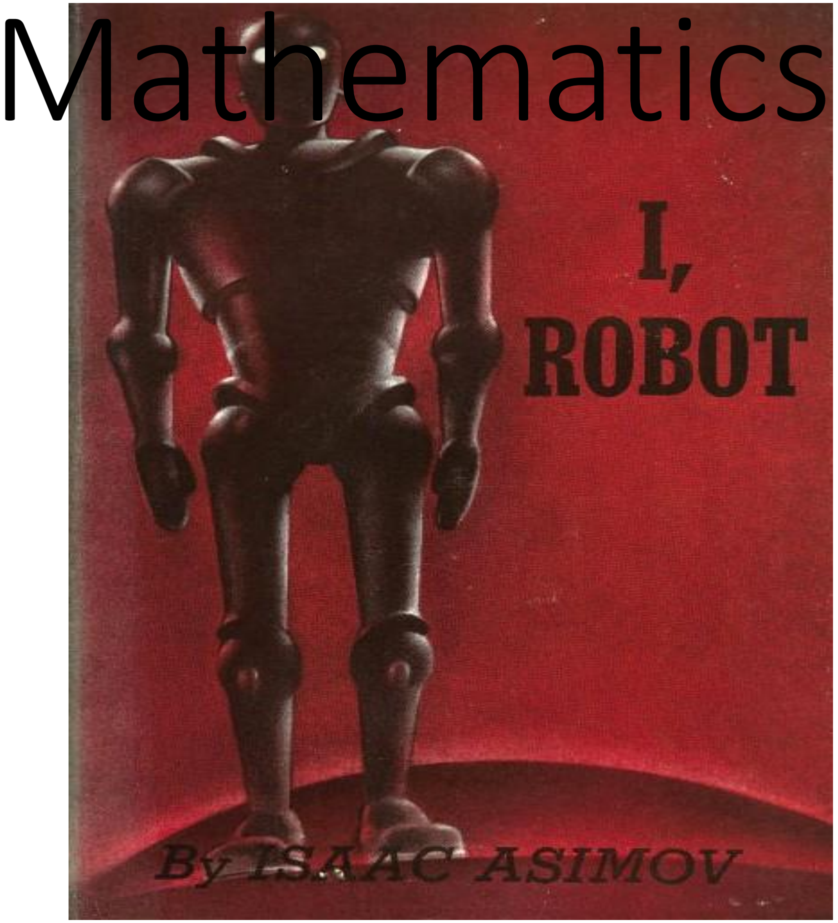
DL in Applied Mathematics



Machine Learning

Linear regression with one variable

Lecture 3: Introducing Optimization.

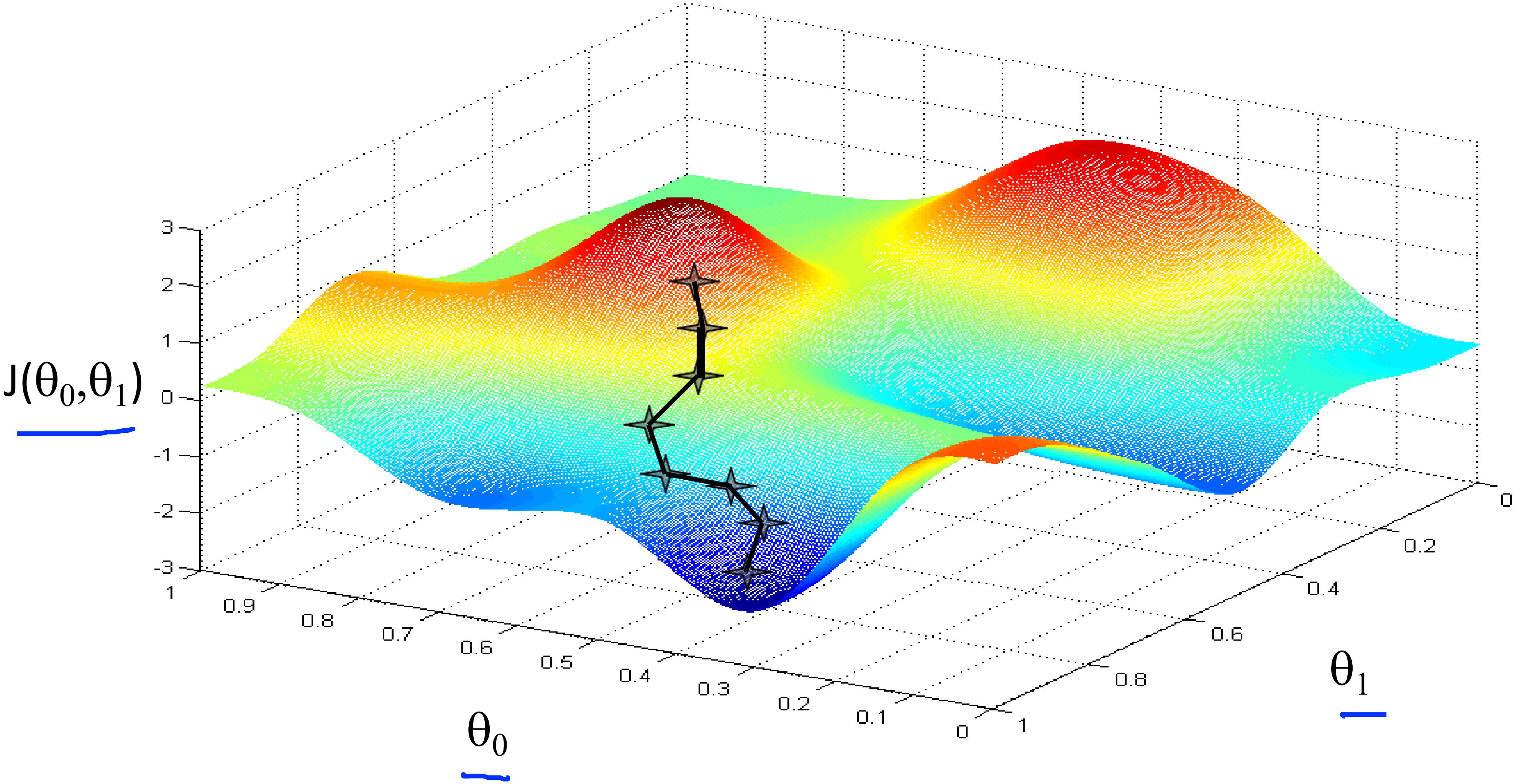
Derivatives in Neural Networks.

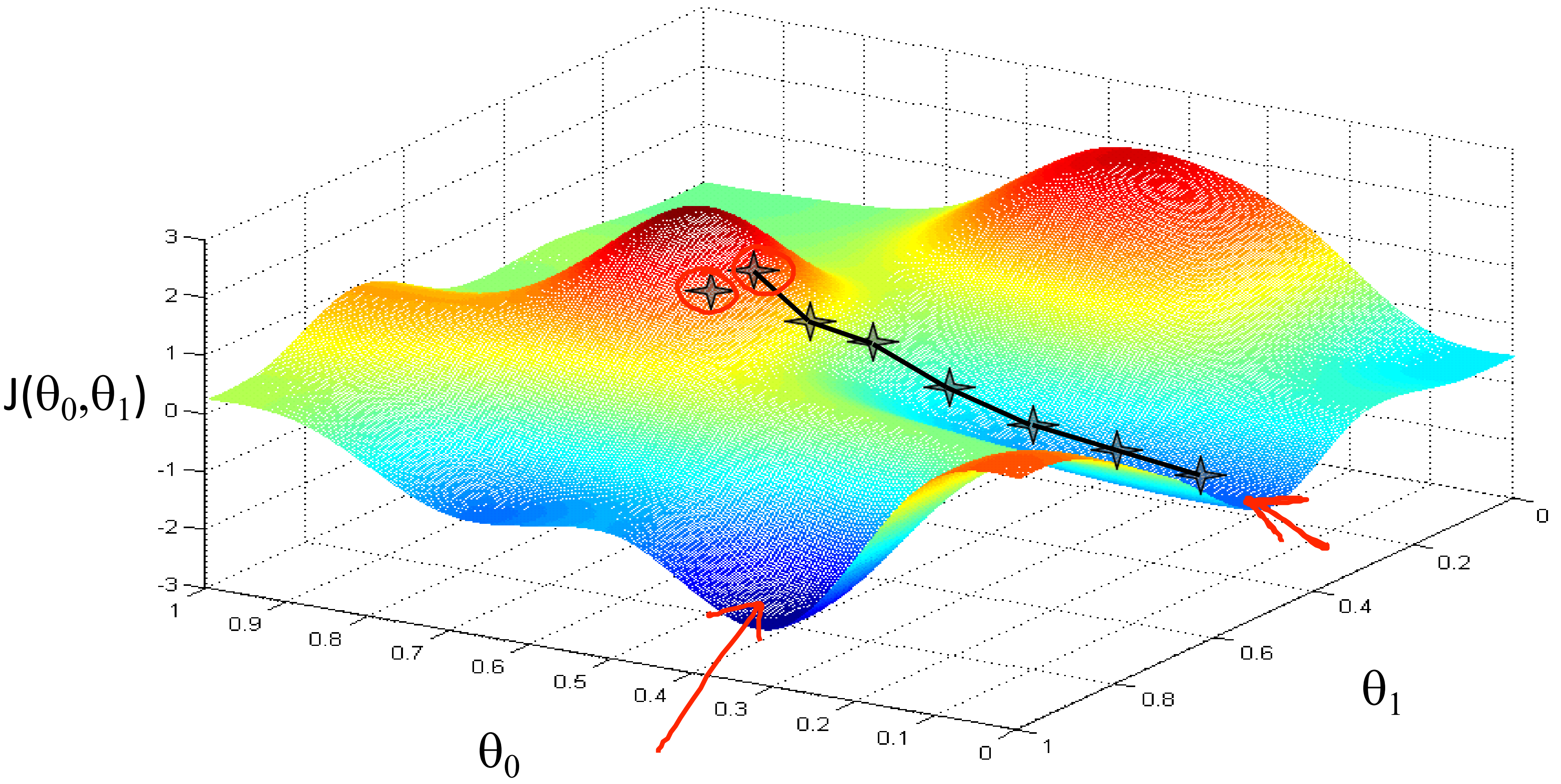
Have some function $J(\theta_0, \theta_1)$ $J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$

Want $\min\limits_{\theta_0, \theta_1} J(\theta_0, \theta_1)$ $\min\limits_{\theta_0 \ldots \theta_n} J(\theta_0, \ldots, \theta_n)$

Outline:

- Start with some θ_0, θ_1 (say $\theta_0 = 0, \theta_1 = 0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum


$J(\theta_0,\theta_1)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1



J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1

Gradient descent algorithm

Assignment

$a := b$

$a := a + 1$

Truth assertion

$a = b$

$a = a + 1$ ✗

θ_0, θ_1

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(for } j = 0 \text{ and } j = 1\text{)}$$

}

learning rate

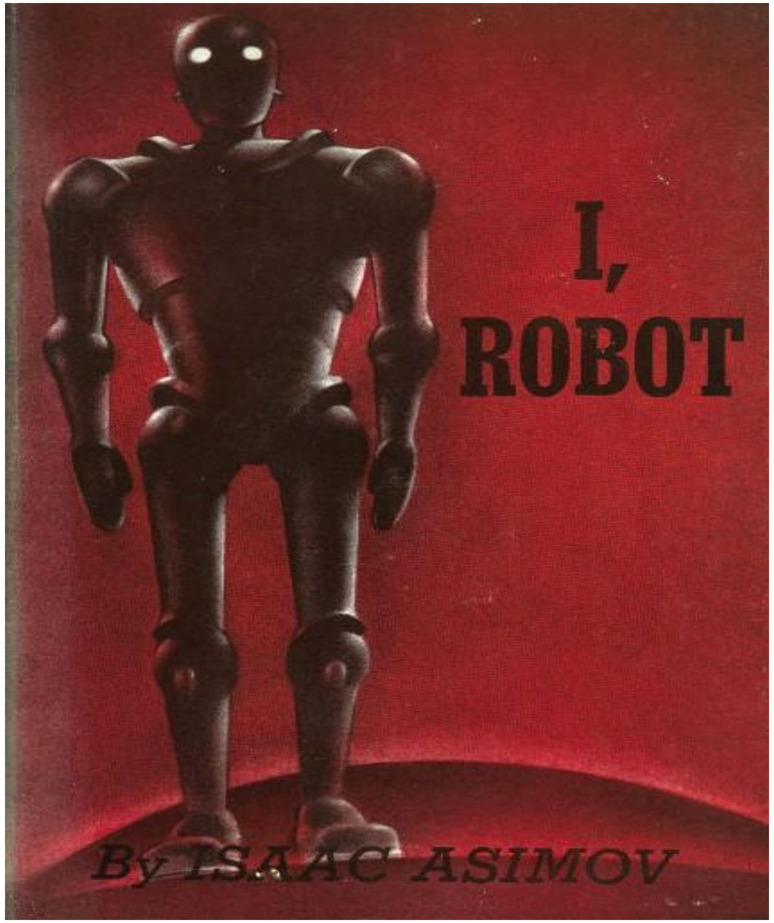
Simultaneously update θ_0 and θ_1

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$



Machine Learning

Linear regression with one variable

Gradient descent intuition

What is Gradient Descent?

Gradient Descent is an iterative optimization algorithm for finding the minimum of a function.

In the context of machine learning, this function is usually the cost or loss function, which measures the difference between the model's predictions and the actual data.

The goal is to adjust the model's parameters (e.g., weights in a neural network) to minimize this cost function.

Gradient descent algorithm

repeat until convergence {

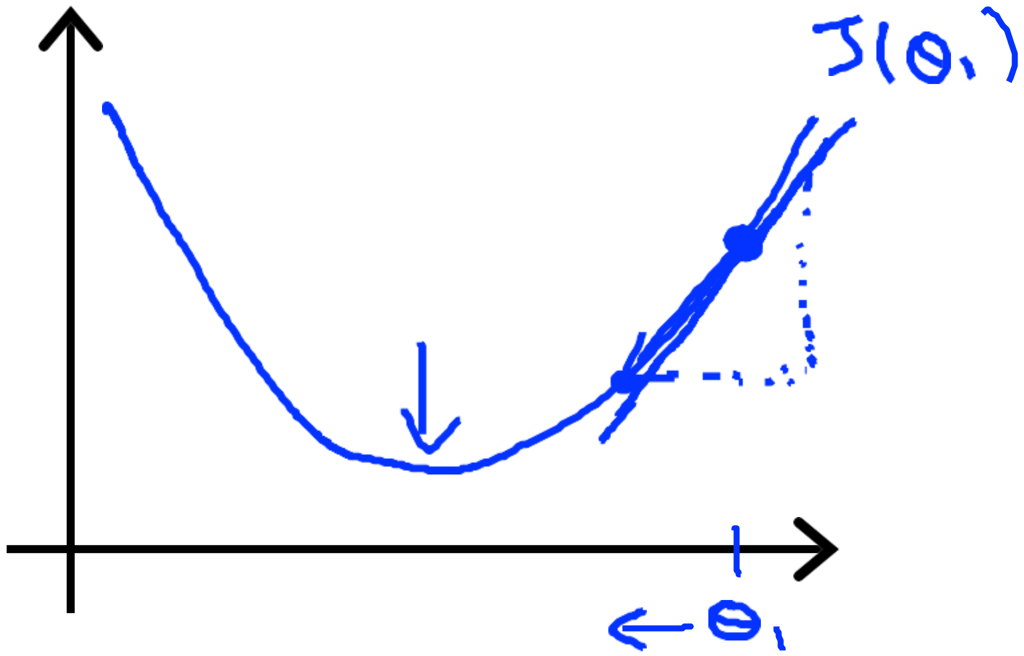
$$\rightarrow \underline{\theta_j} := \underline{\theta_j} - \underbrace{\alpha}_{\text{learning rate}} \underbrace{\boxed{\frac{\partial}{\partial\theta_j} J(\theta_0,\theta_1)}}_{\text{derivative}}$$

(simultaneously update $j = 0$ and $j = 1$)

}

$$\min_{\theta_1} J(\theta_{\underline{1}})$$

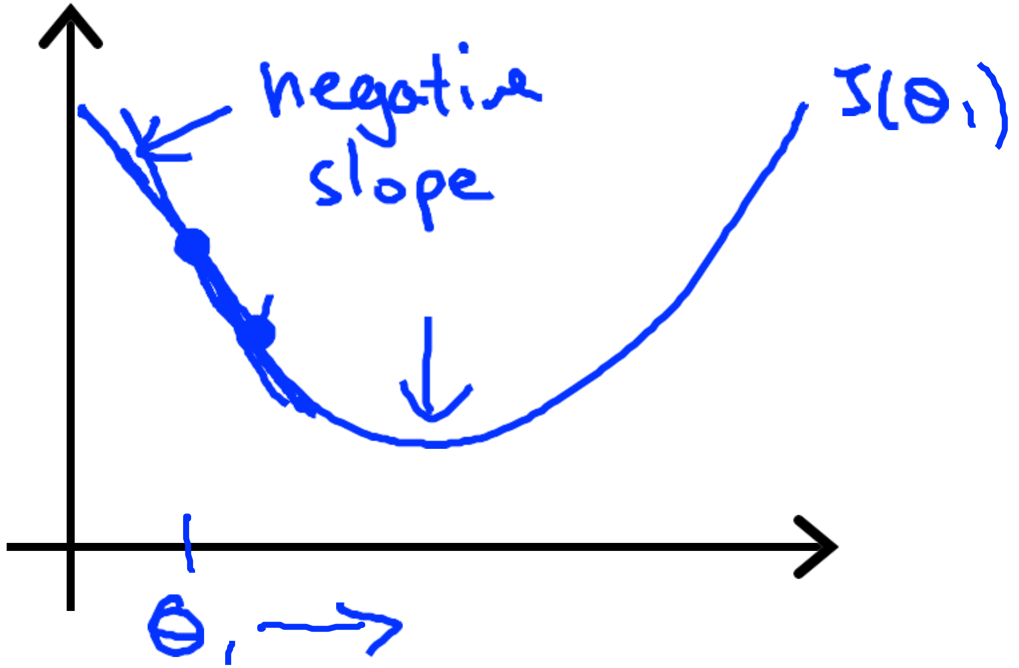
$\theta_1 \in \mathbb{R}.$



$(\theta_1 \in \mathbb{R})$

$$\theta_1 := \theta_1 - \alpha \underbrace{\boxed{\overset{\frac{\partial}{\partial \theta_1}}{\frac{\partial}{\partial \theta_1} J(\theta_1)}}}_{\geq 0}$$

$$\theta_1 := \theta_1 - \underline{\alpha} \cdot (\text{positive number})$$

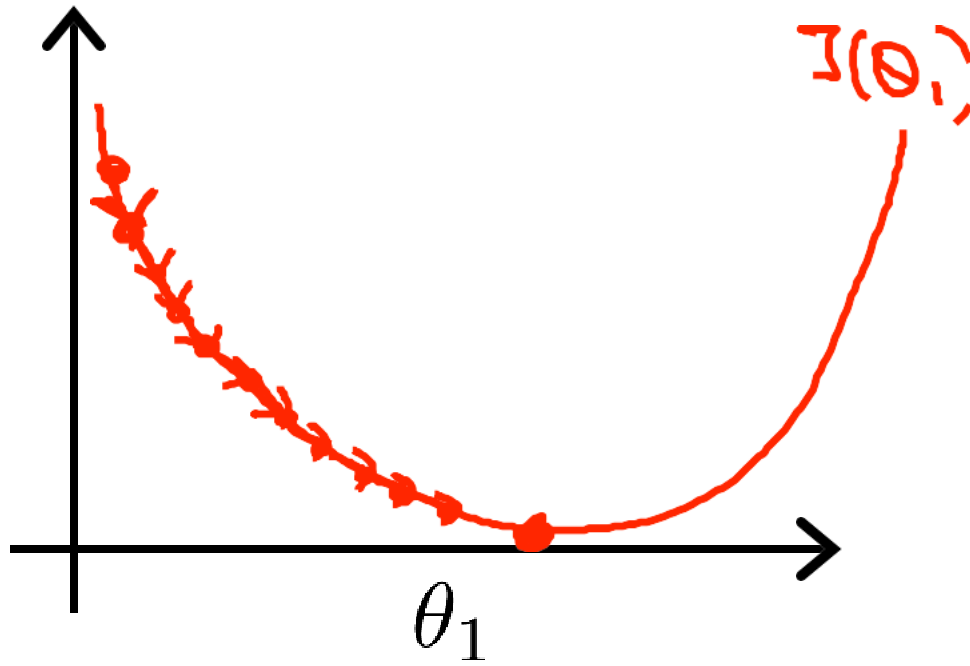


$$\underbrace{\frac{\partial}{\partial \theta_1} J(\theta_1)}_{\leq 0}$$

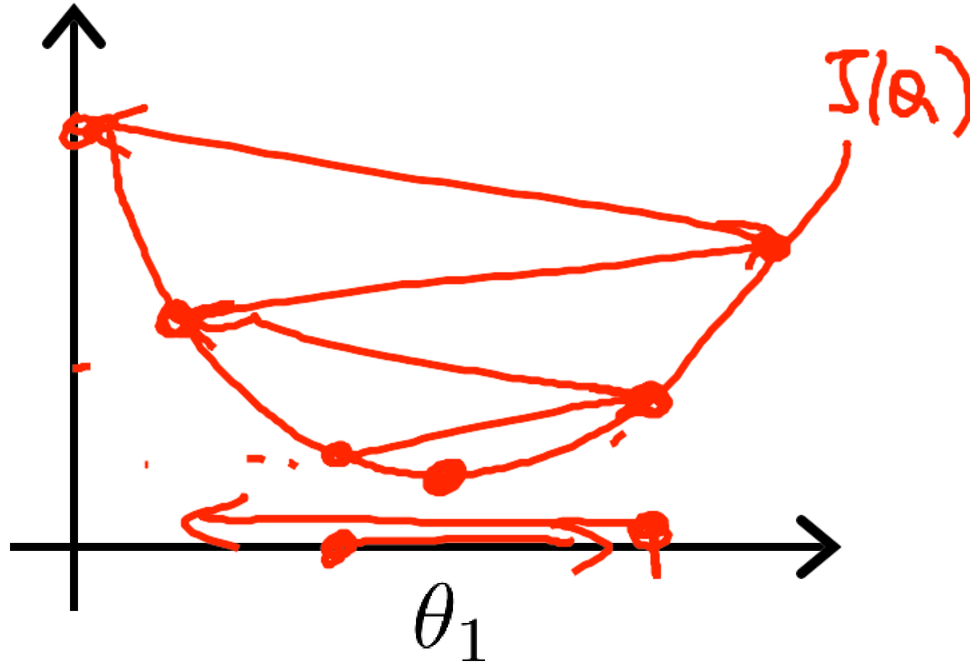
$$\theta_1 := \underline{\theta_1} \underset{\uparrow}{-} \alpha \, (\underset{\uparrow}{\text{negative}} \text{ number})$$

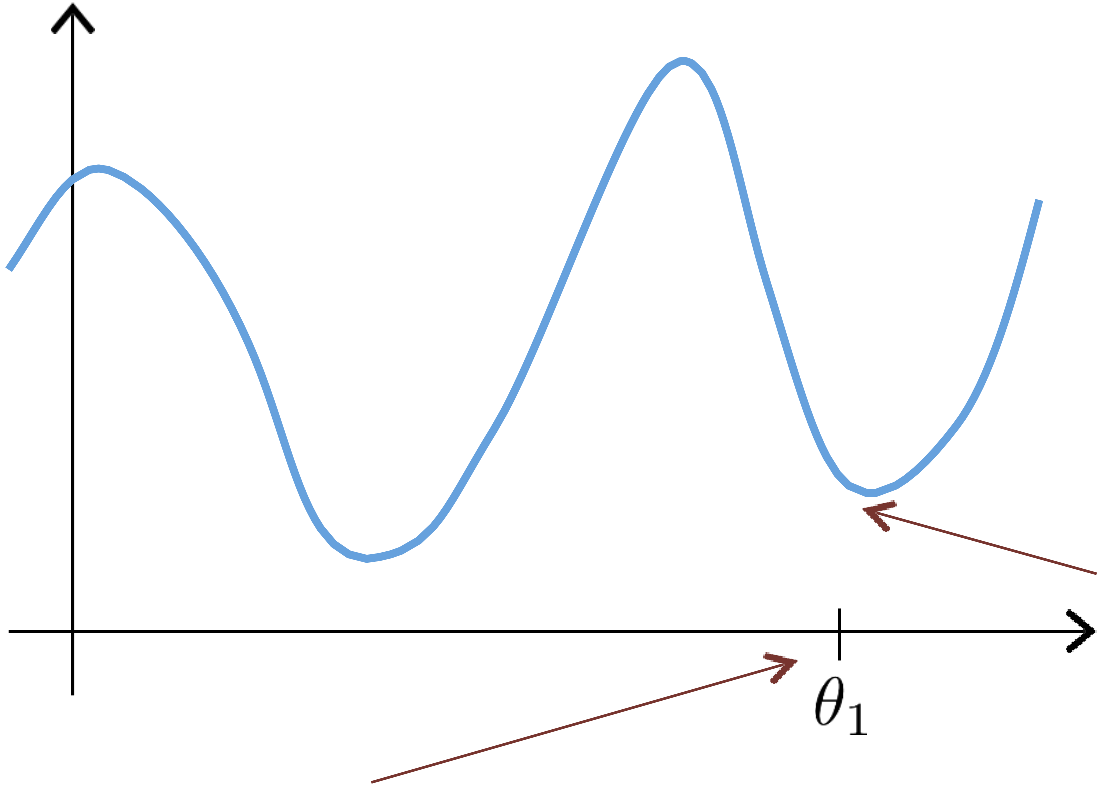
$$\boxed{\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)}$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.





θ_1 at local optima

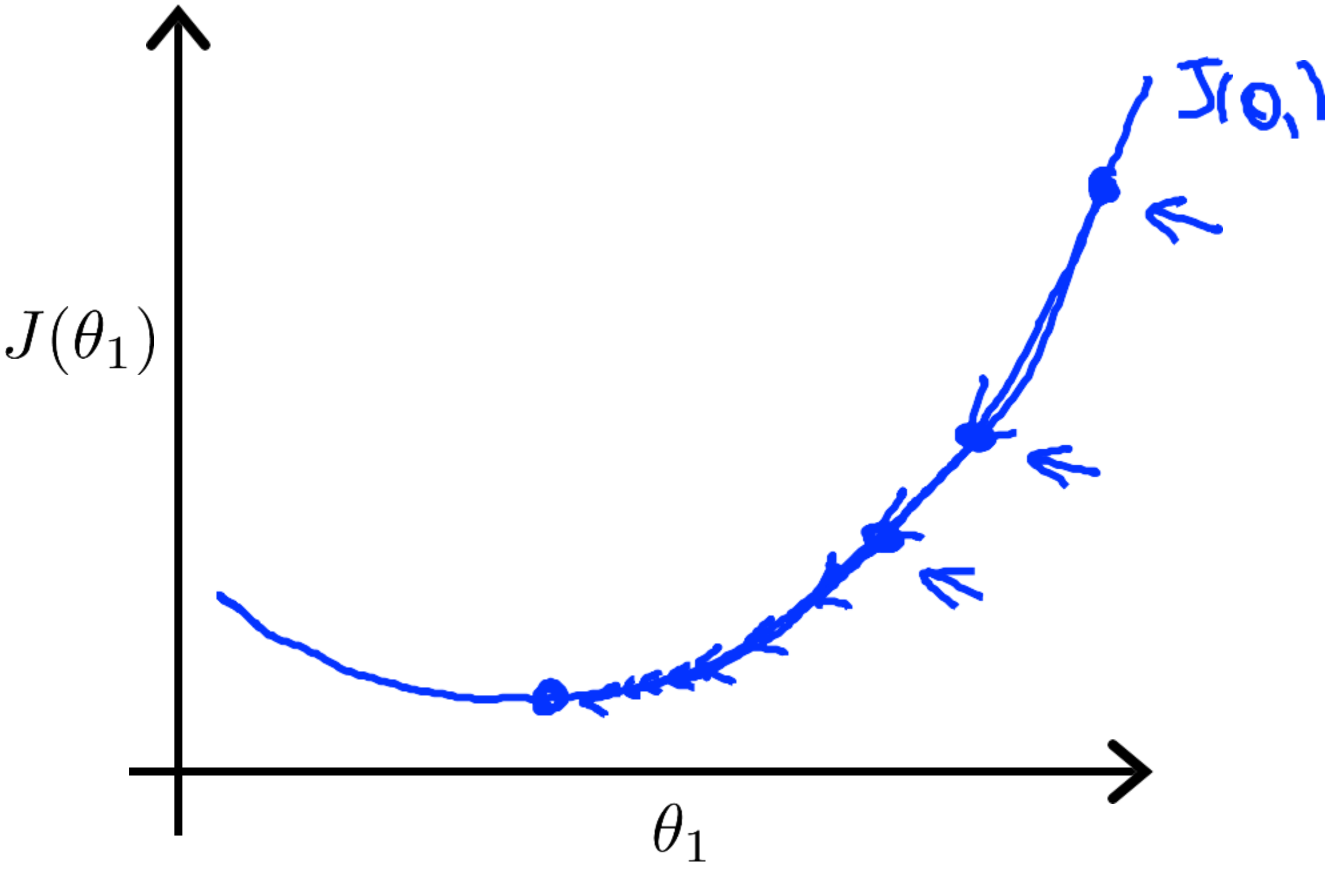
Current value of θ_1

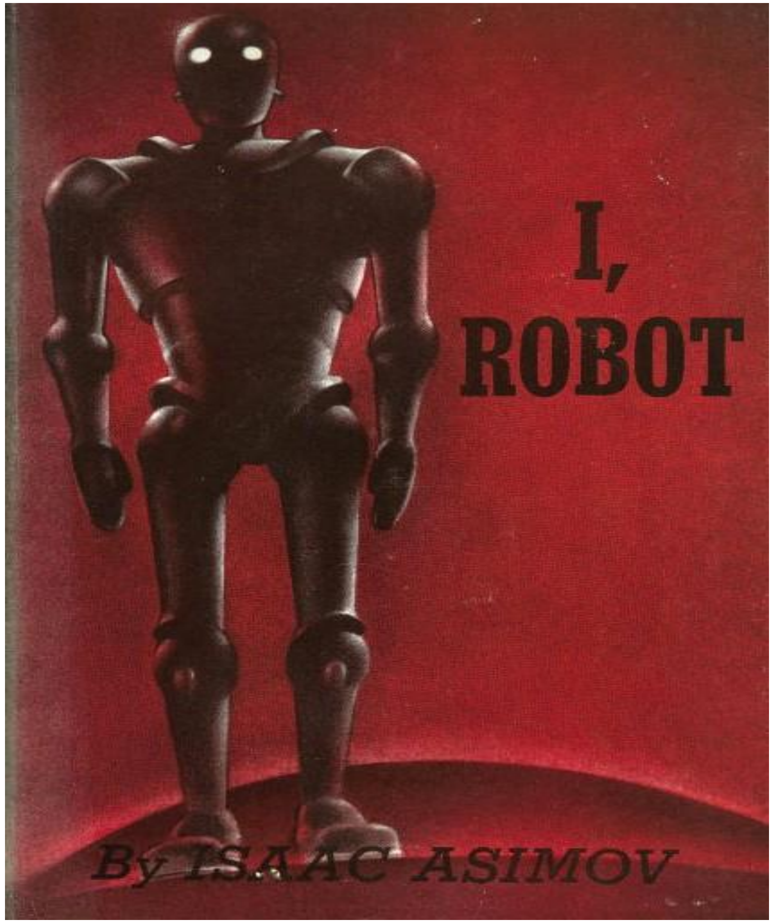
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{d}{d\theta_1} J(\theta_1)}$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.





Machine Learning

Linear regression with one va

Gradient descent for linear regression

Gradient descent algorithm

```
repeat until convergence {
    $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
        (for $j = 1$ and $j = 0$)
}
```

Linear Regression Model

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

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

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

$$= \frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=1}^{m}\left(\underline{\theta_0 + \theta_1 x^{(i)}} - y^{(i)}\right)^2$$

$$j = 0 : \underline{\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$j = 1 : \underline{\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x^{(i)}$$

Gradient descent algorithm

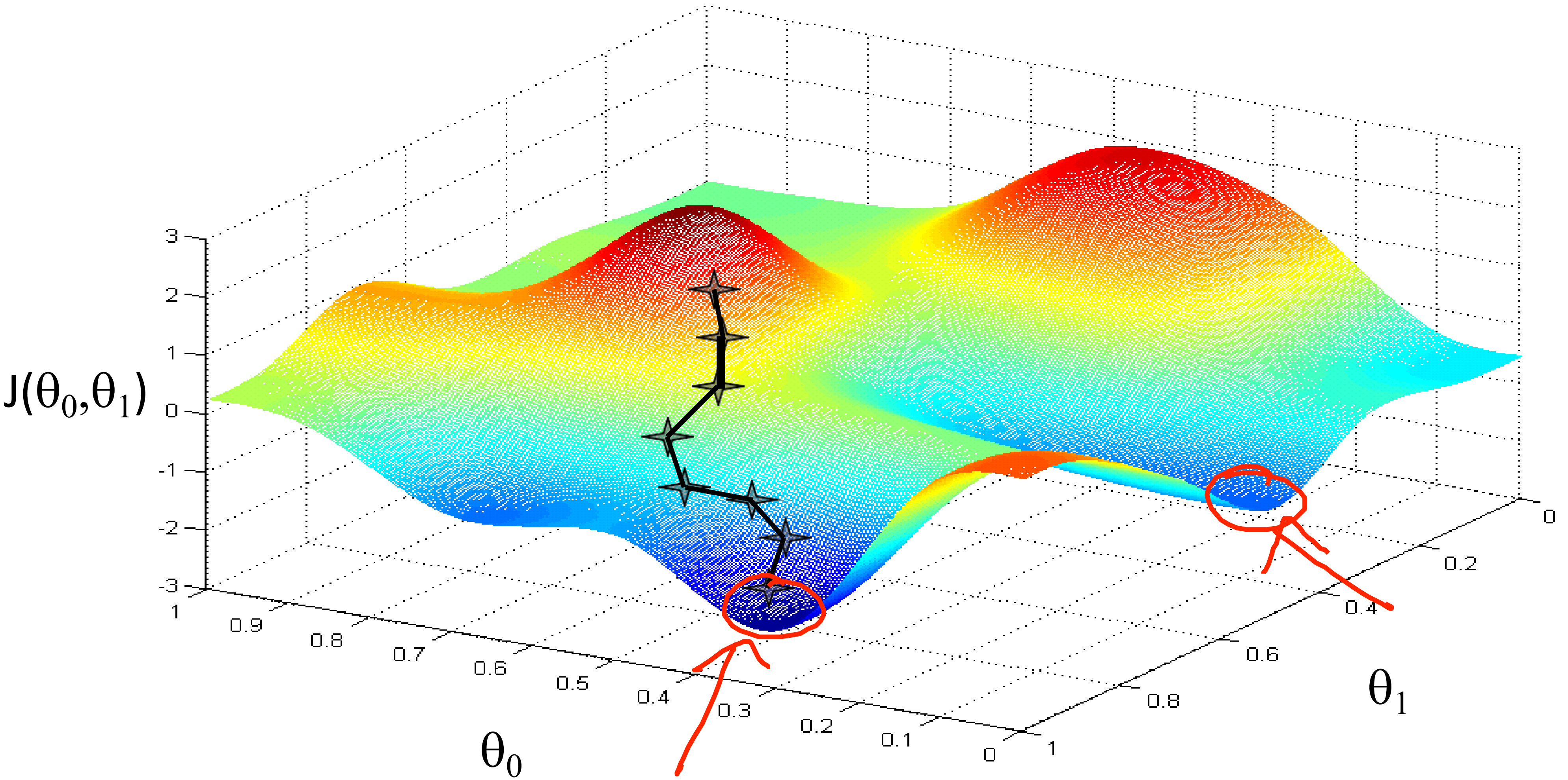
repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \overbrace{\boxed{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)}}^{\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)}$$

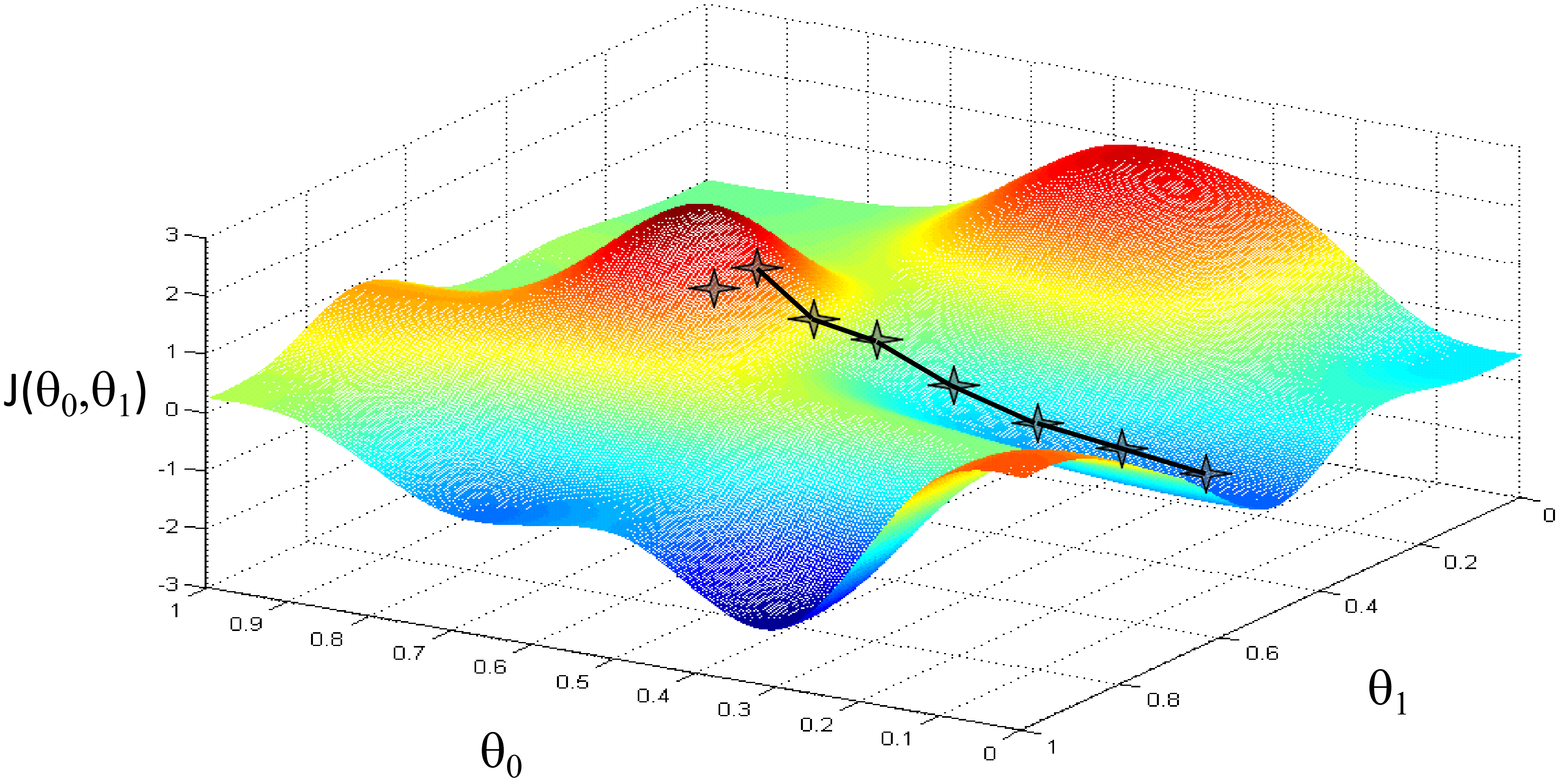
$$\theta_1 := \theta_1 - \alpha \underbrace{\boxed{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x^{(i)}}}_{\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)}$$

}

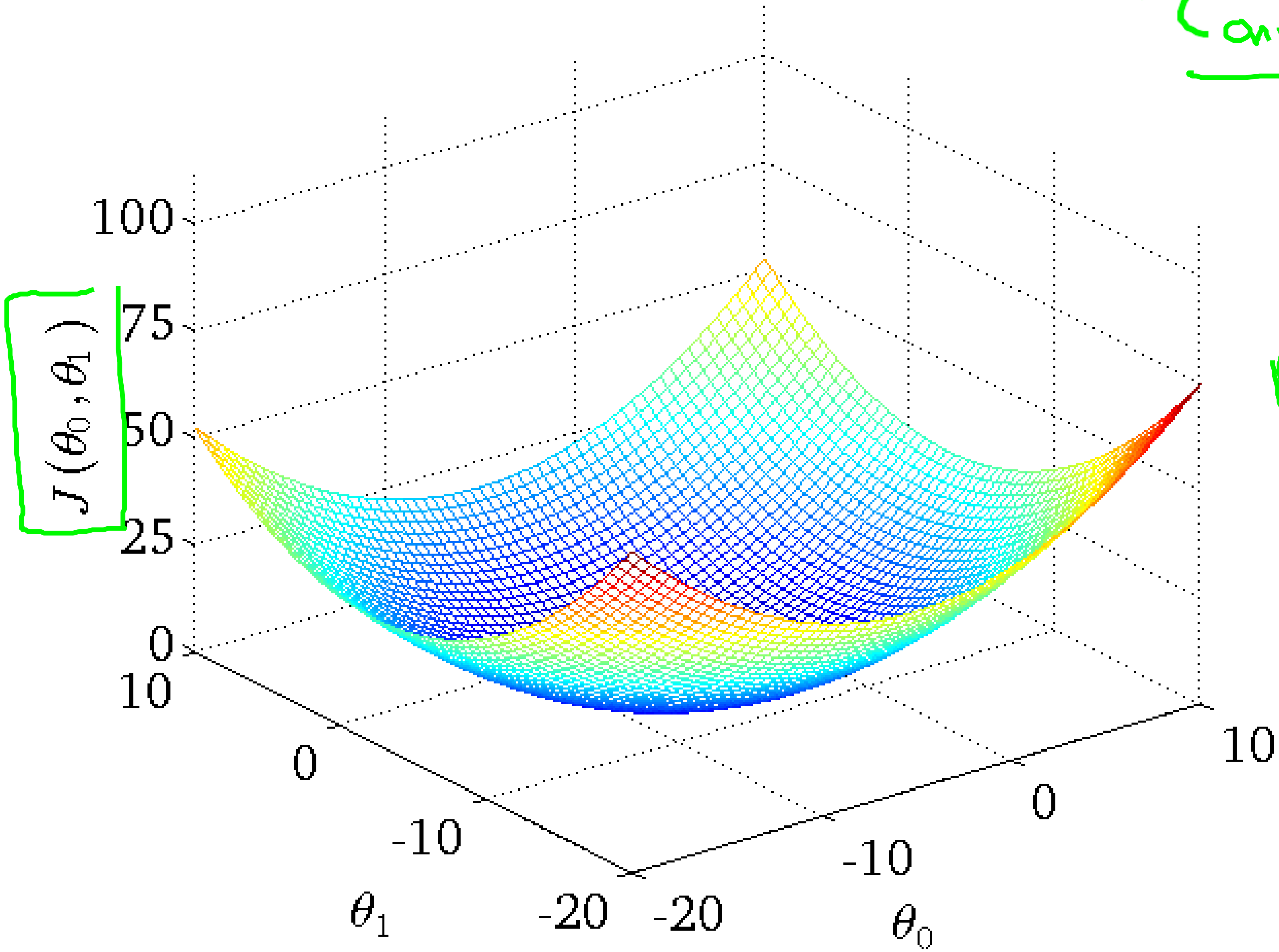
update θ_0 and θ_1 simultaneously



J(θ_0,θ_1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1


J(θ_0,θ_1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1

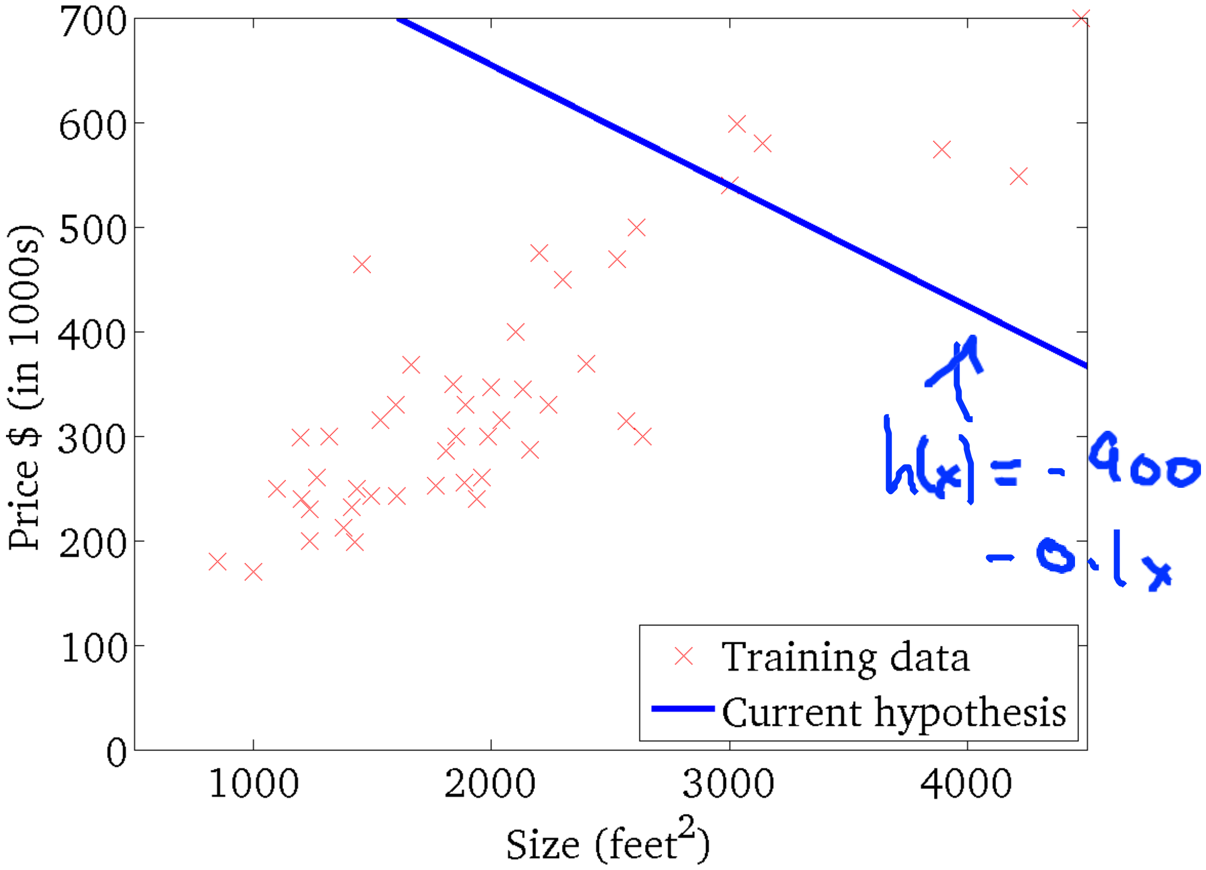
"Convex function"



Bowl-shaped

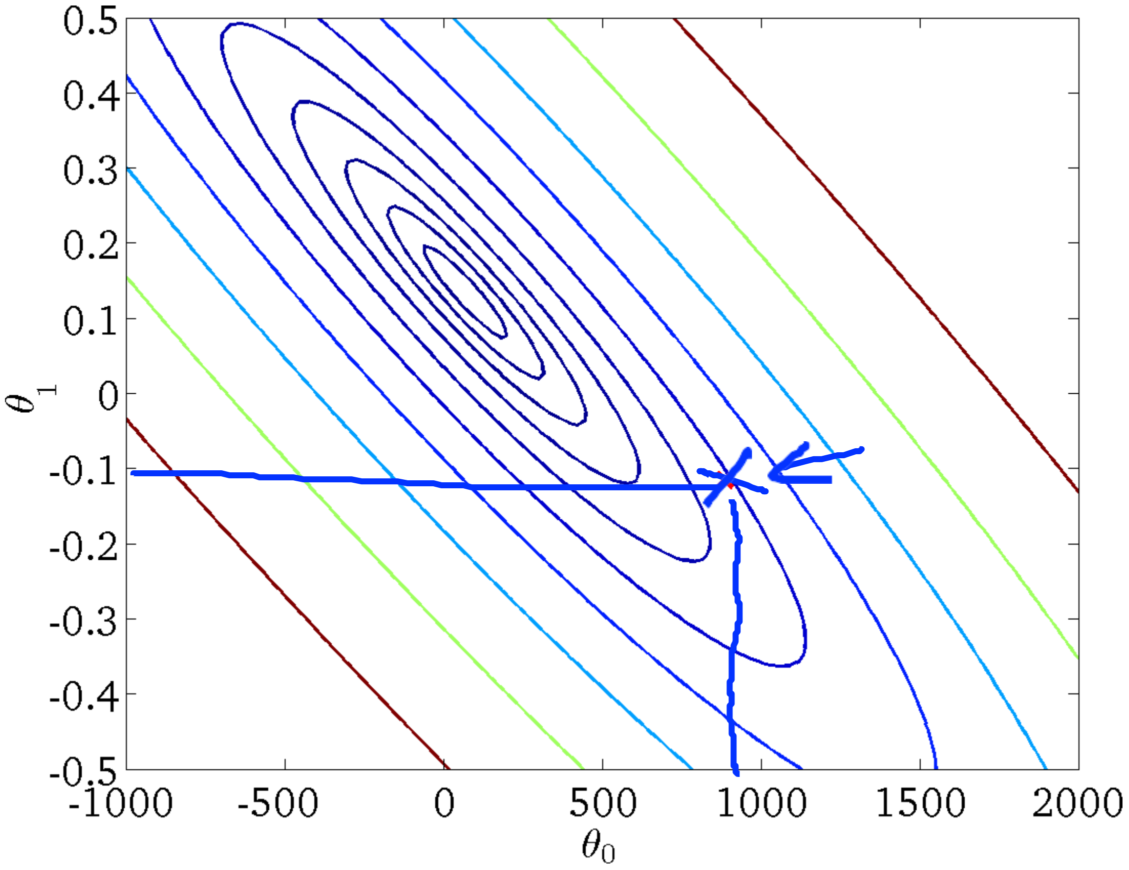
$h_\theta(x)$

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



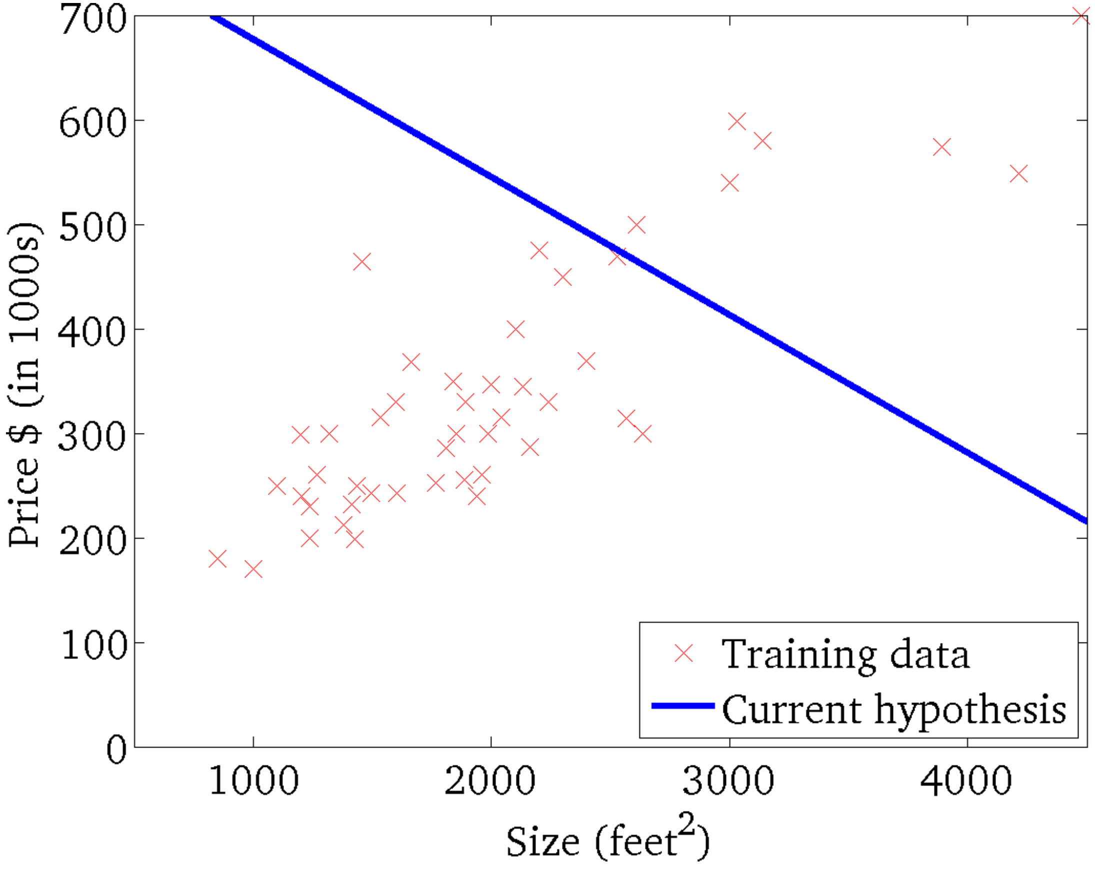
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



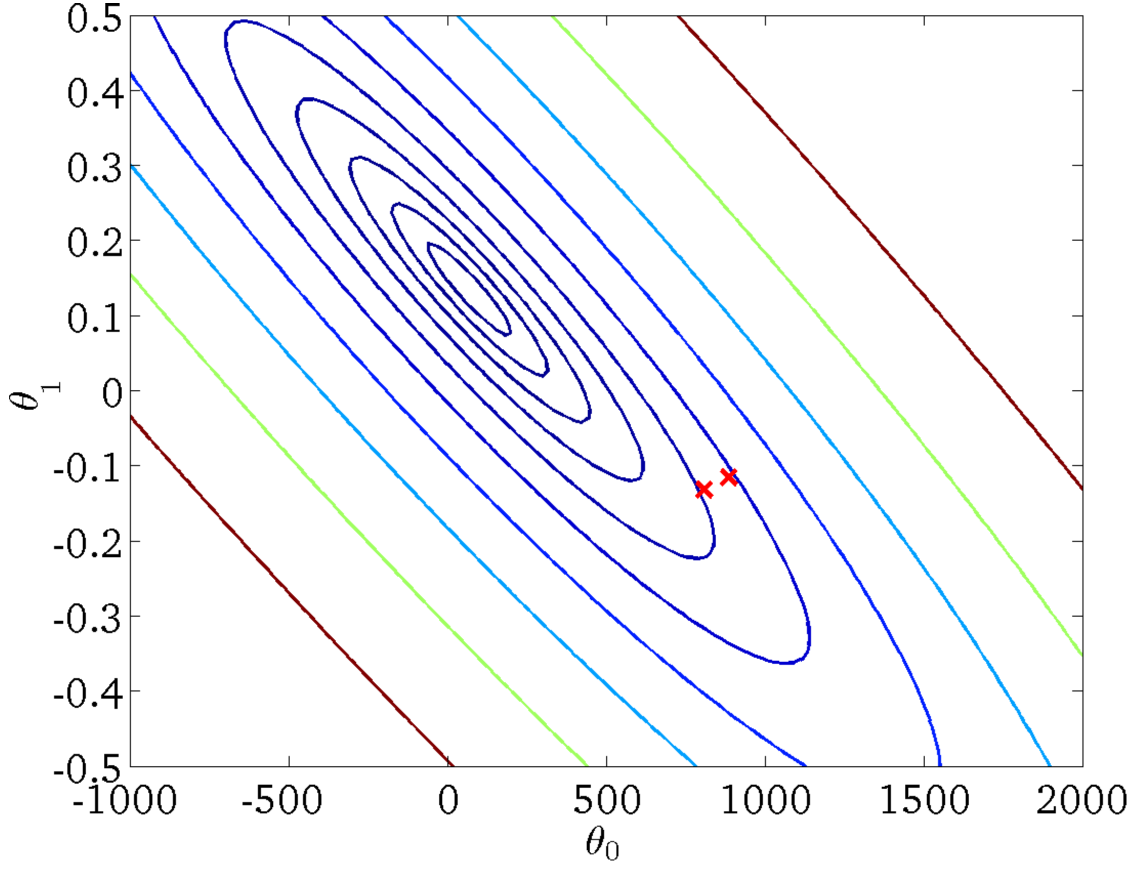
$h_\theta(x)$

(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)

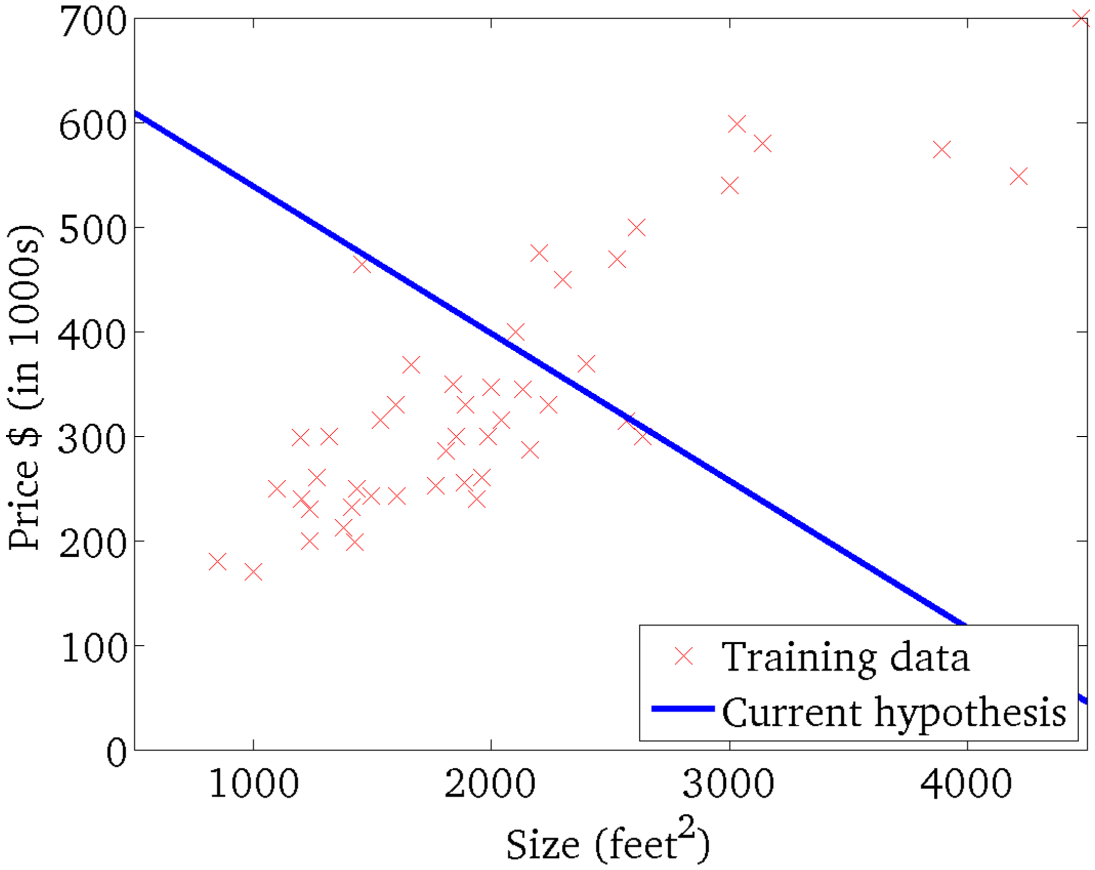


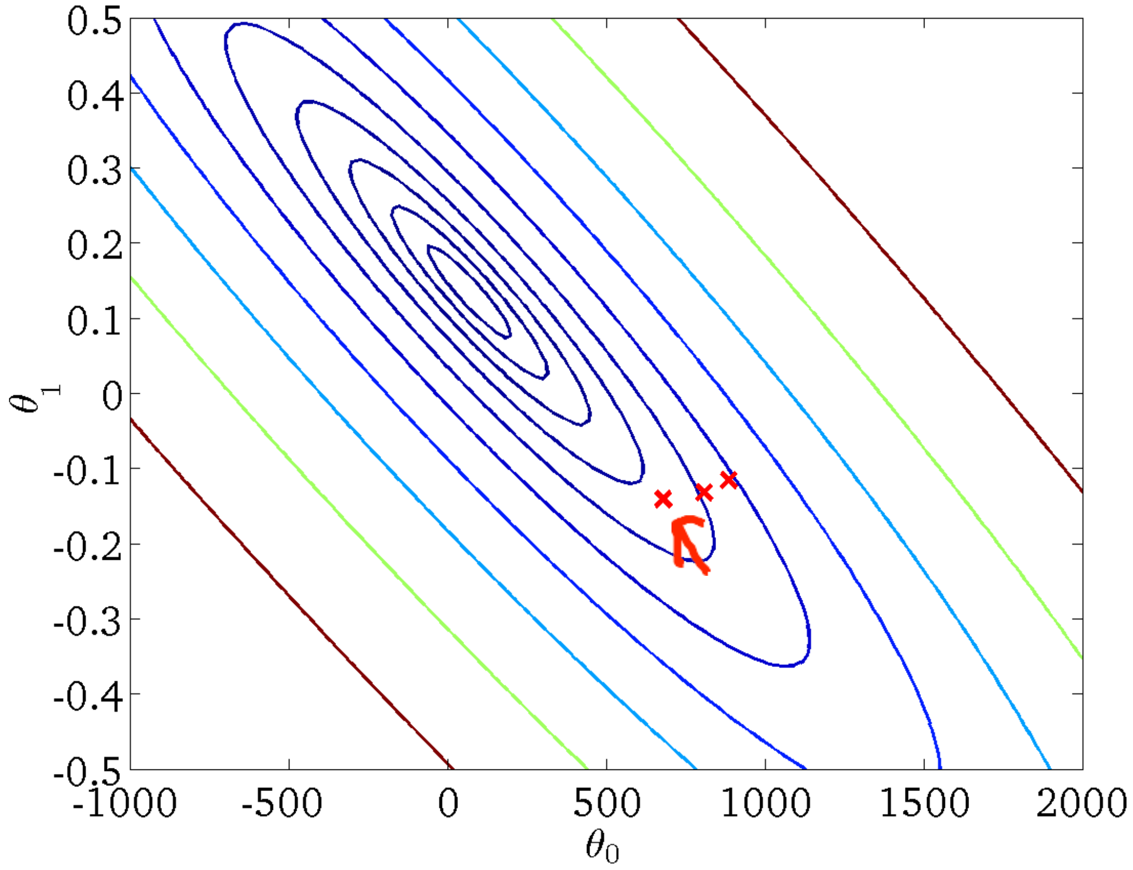
$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

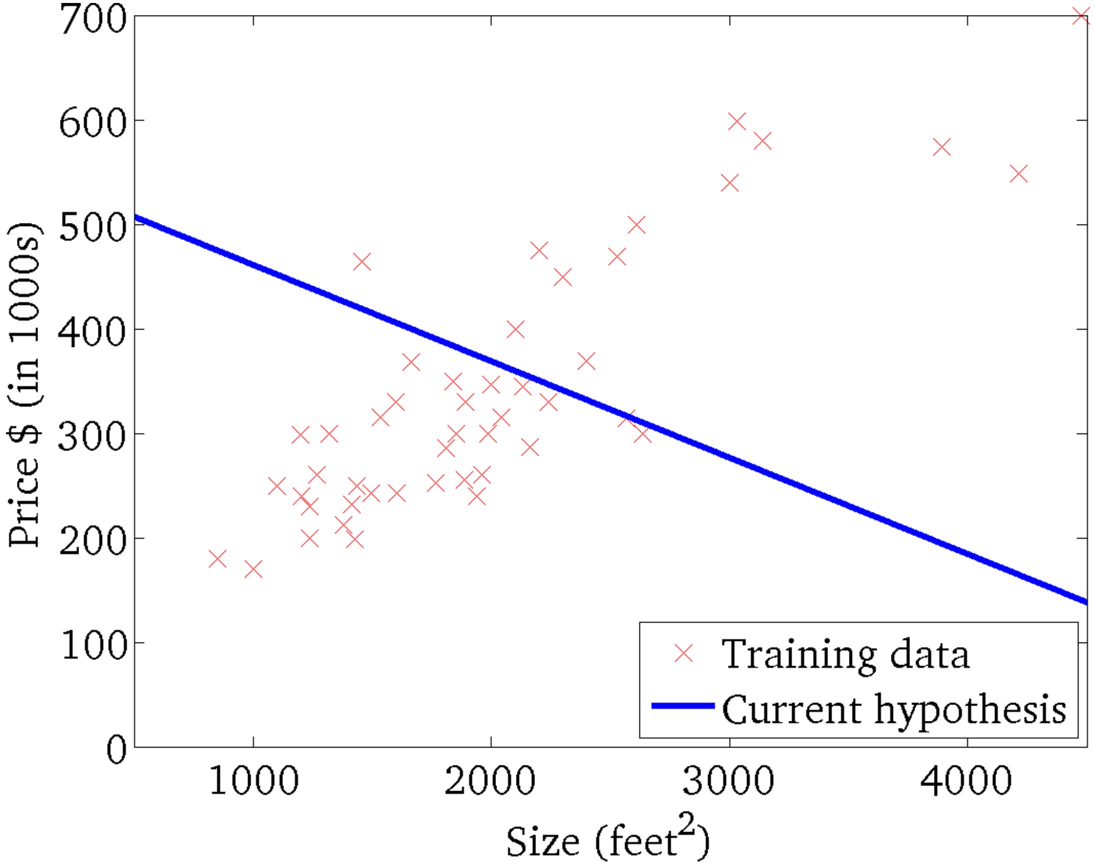
(function of the parameters θ_0, θ_1)





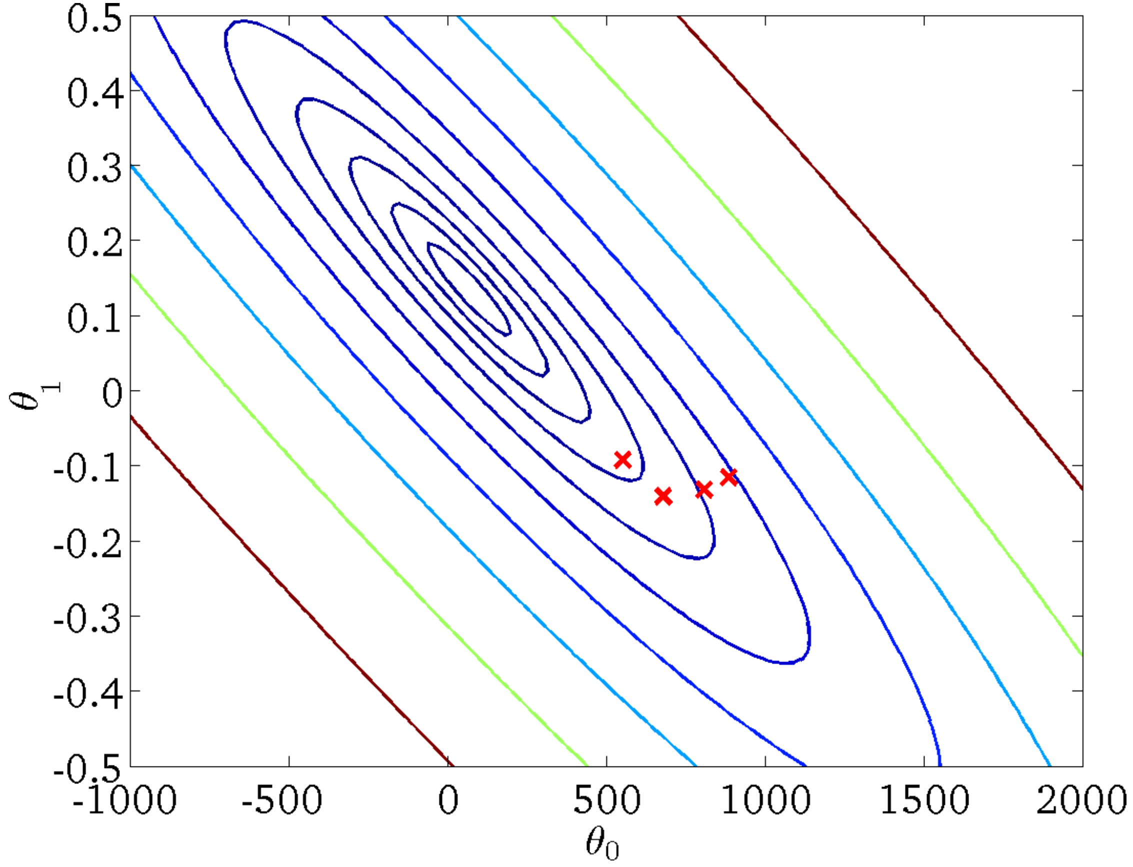
$h_\theta(x)$

(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)

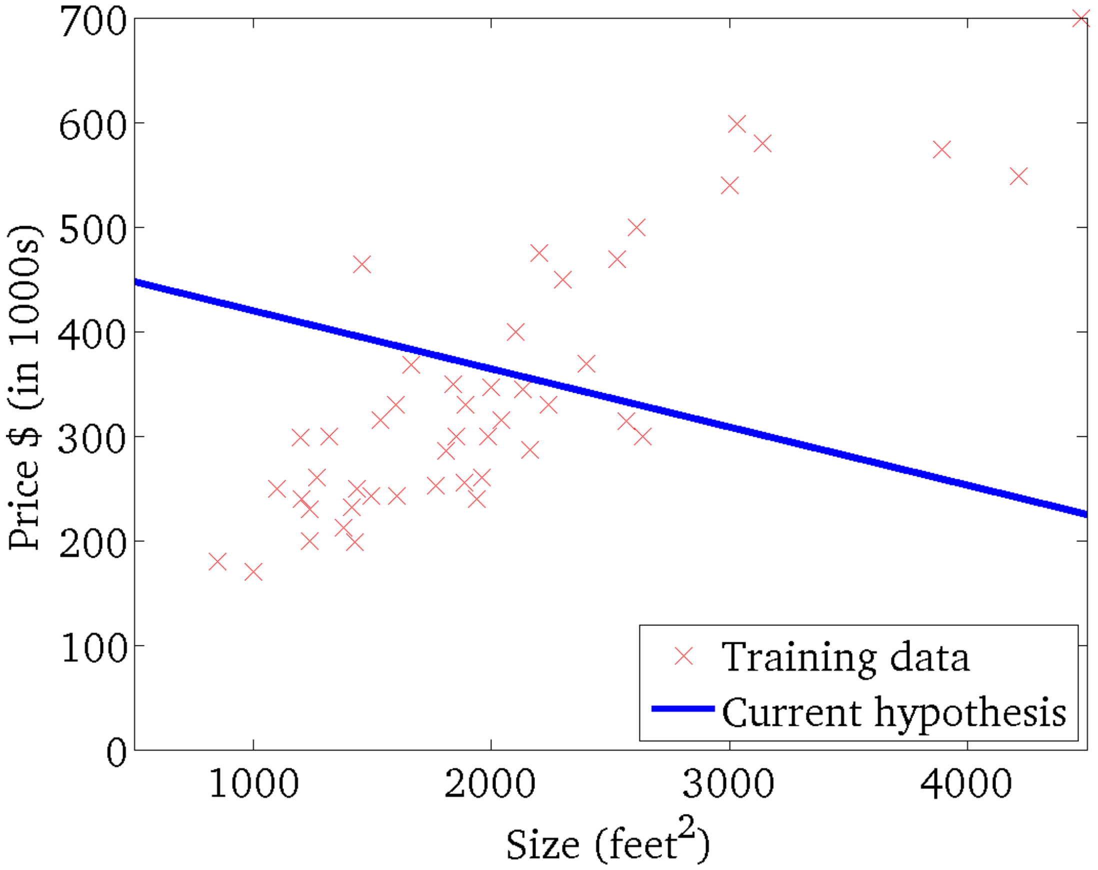


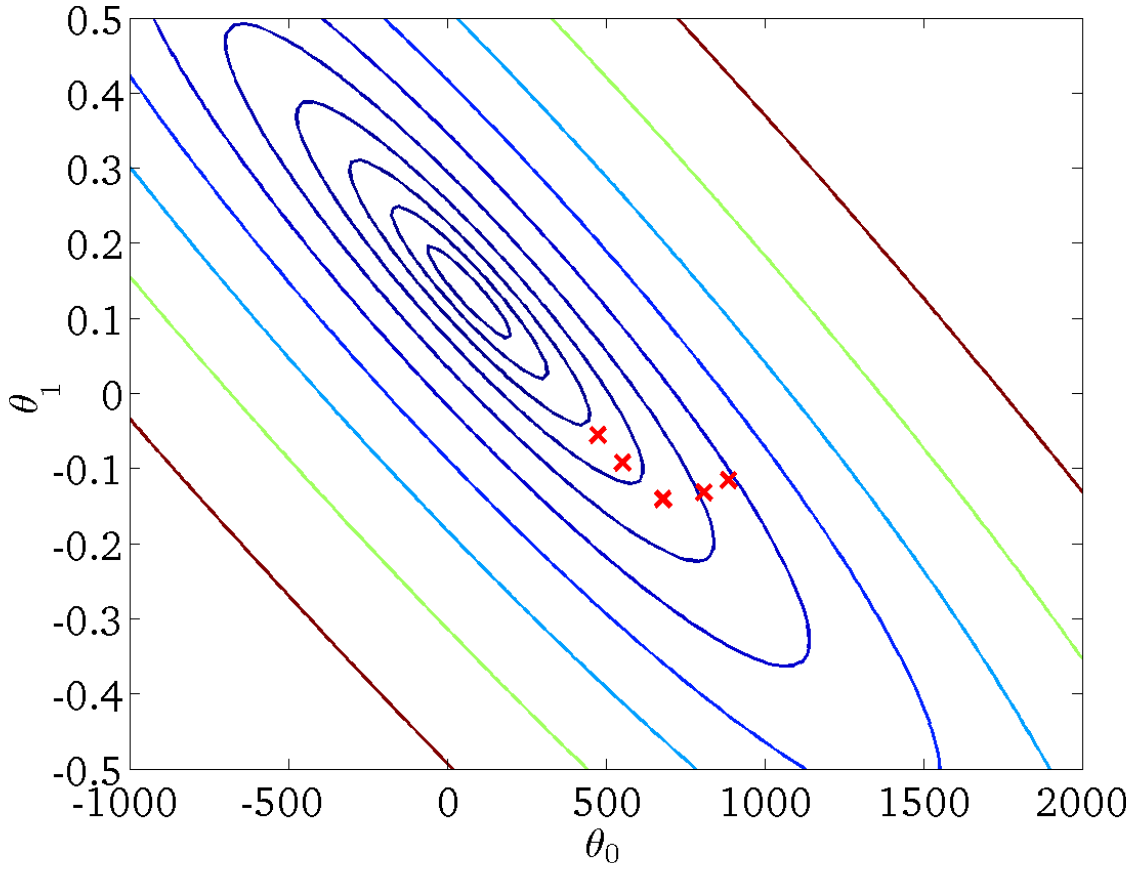
$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

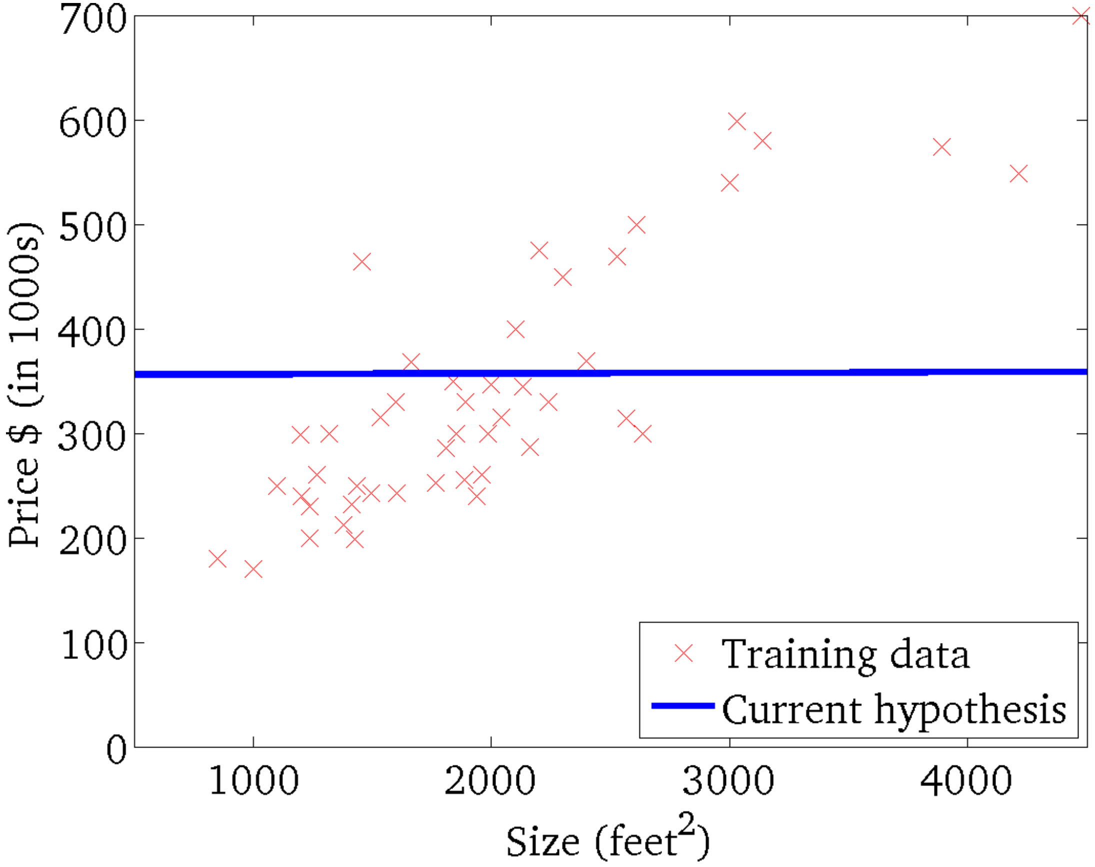
(function of the parameters θ_0, θ_1)





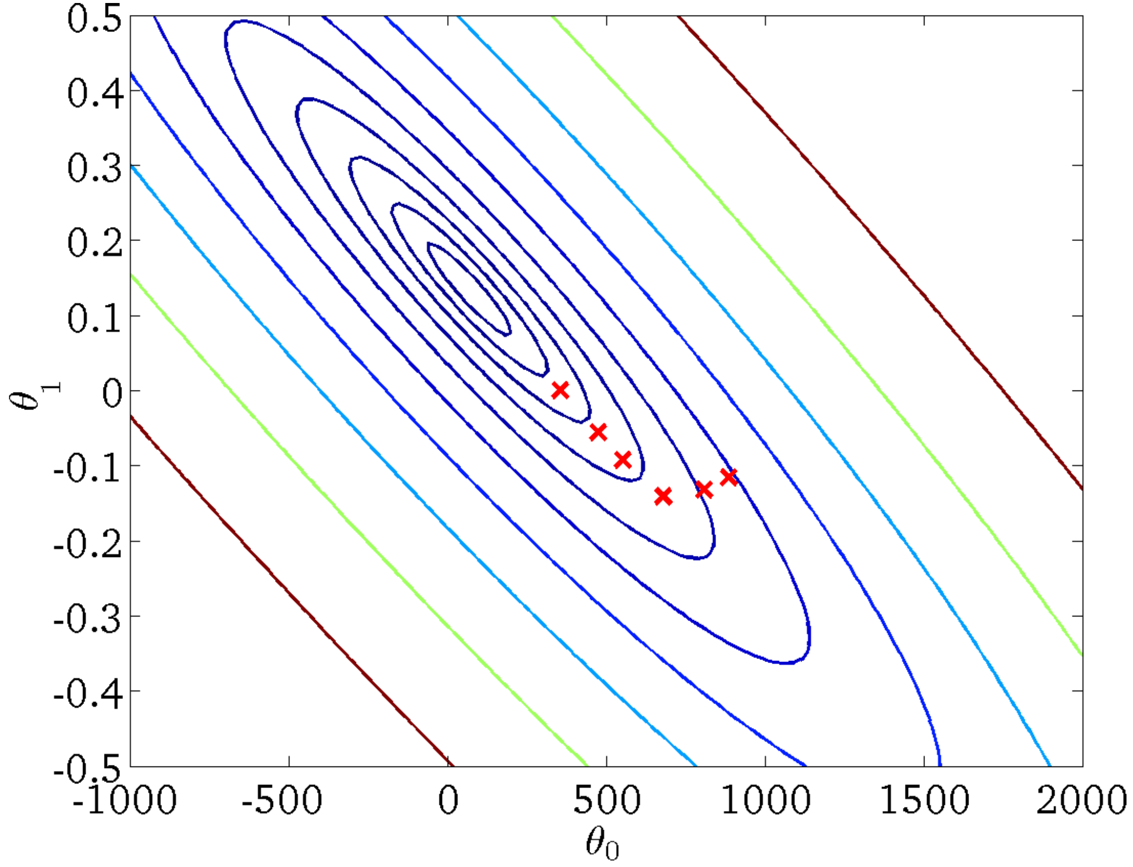
$h_\theta(x)$

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



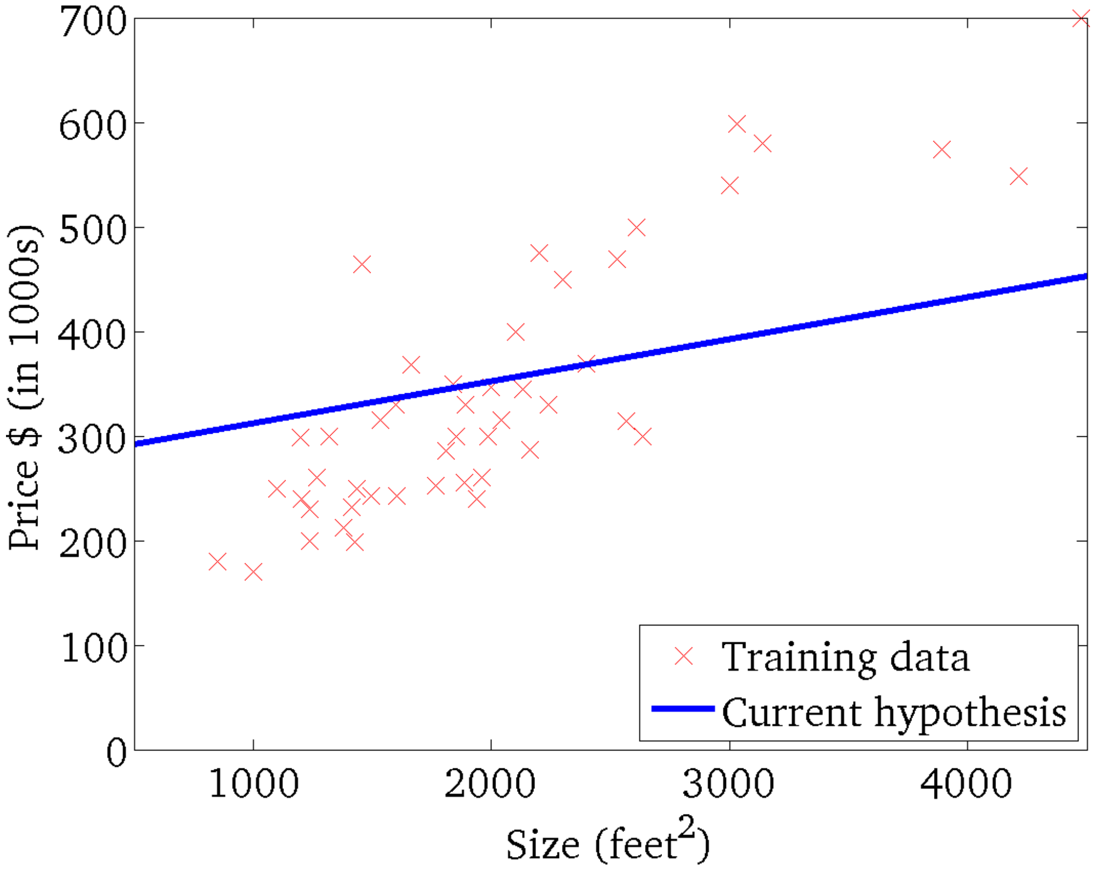
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



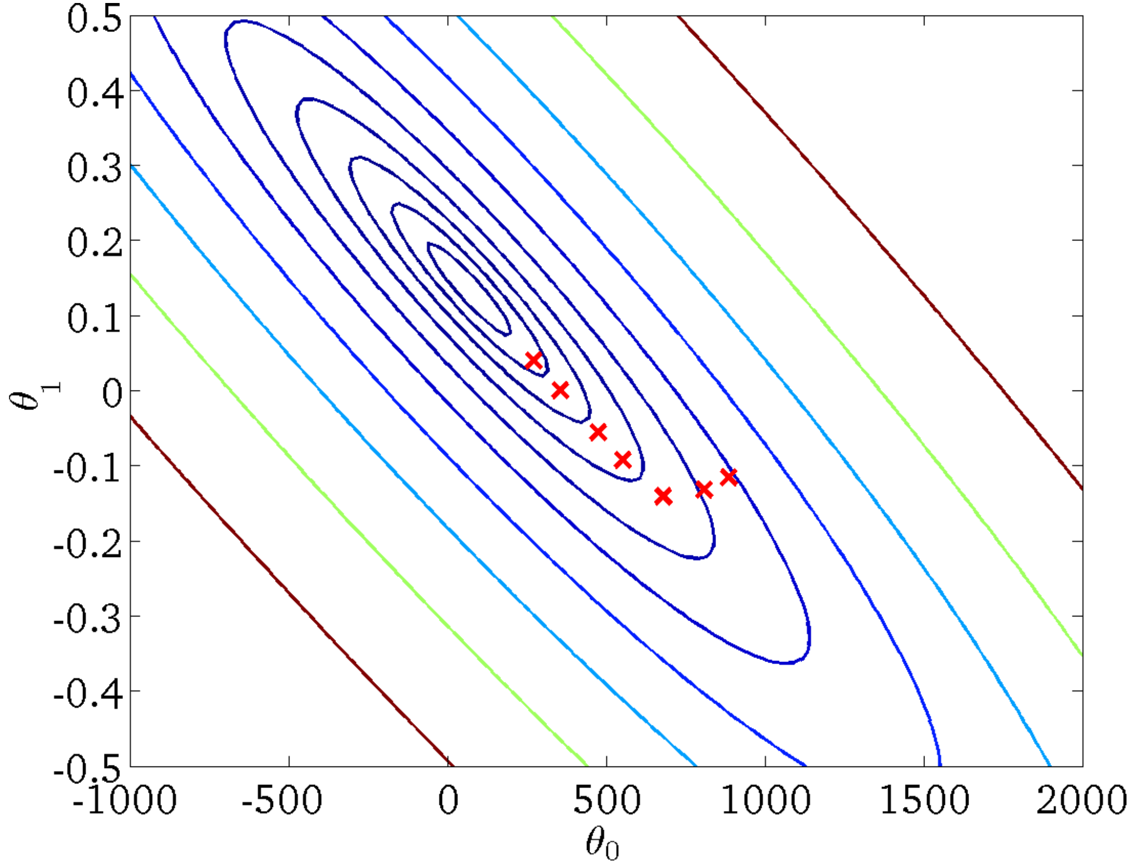
$h_\theta(x)$

(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)

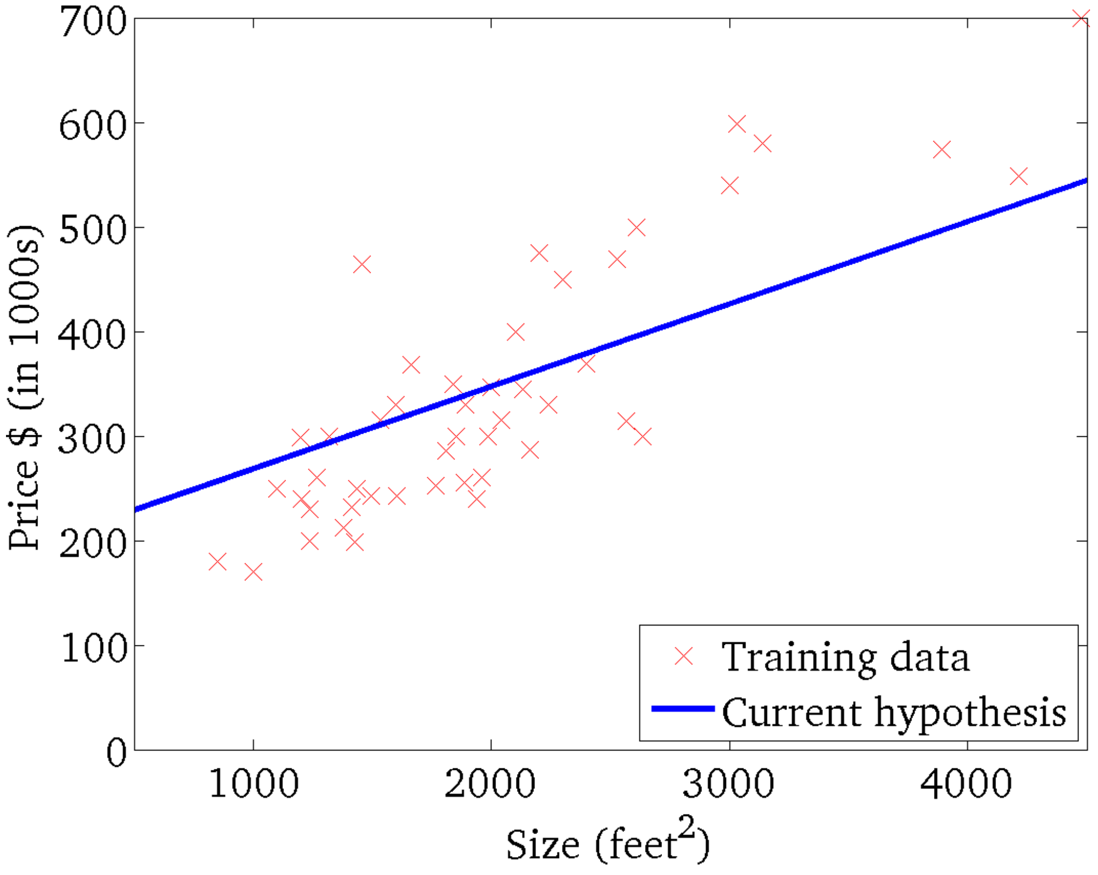


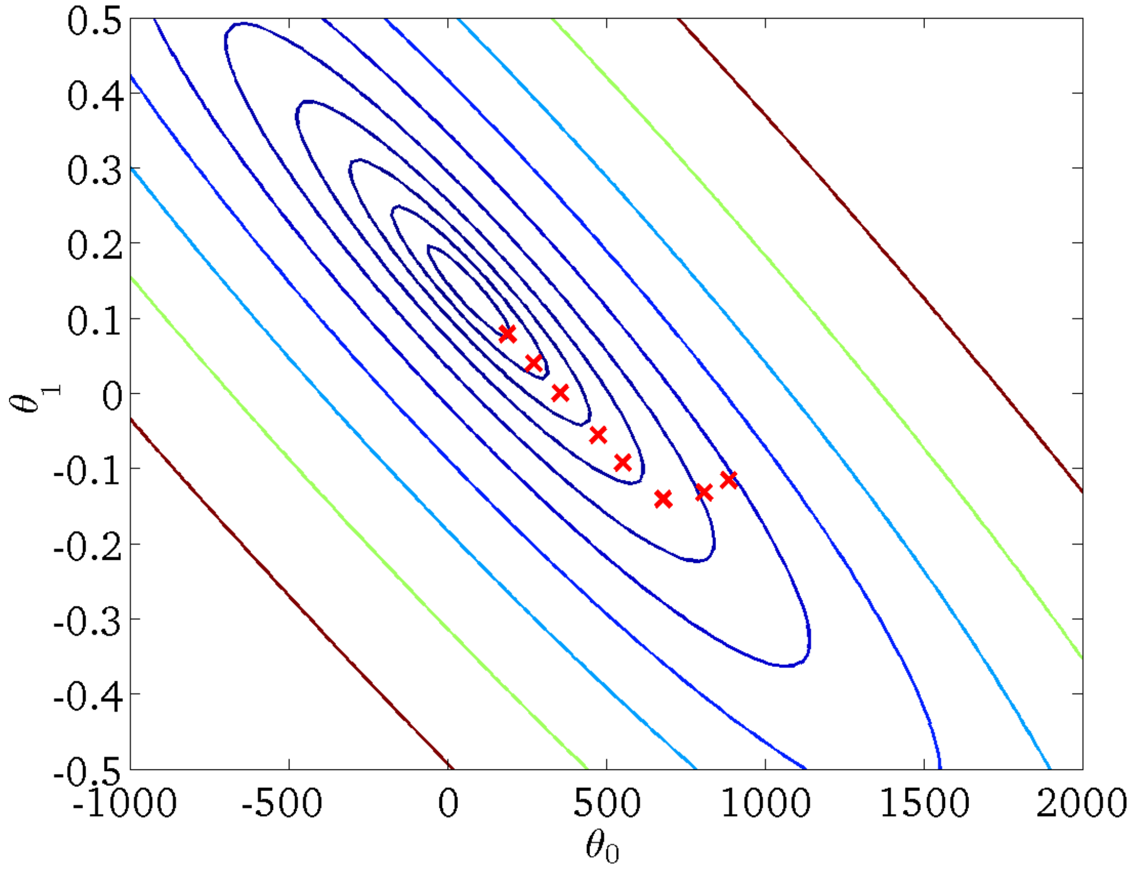
$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

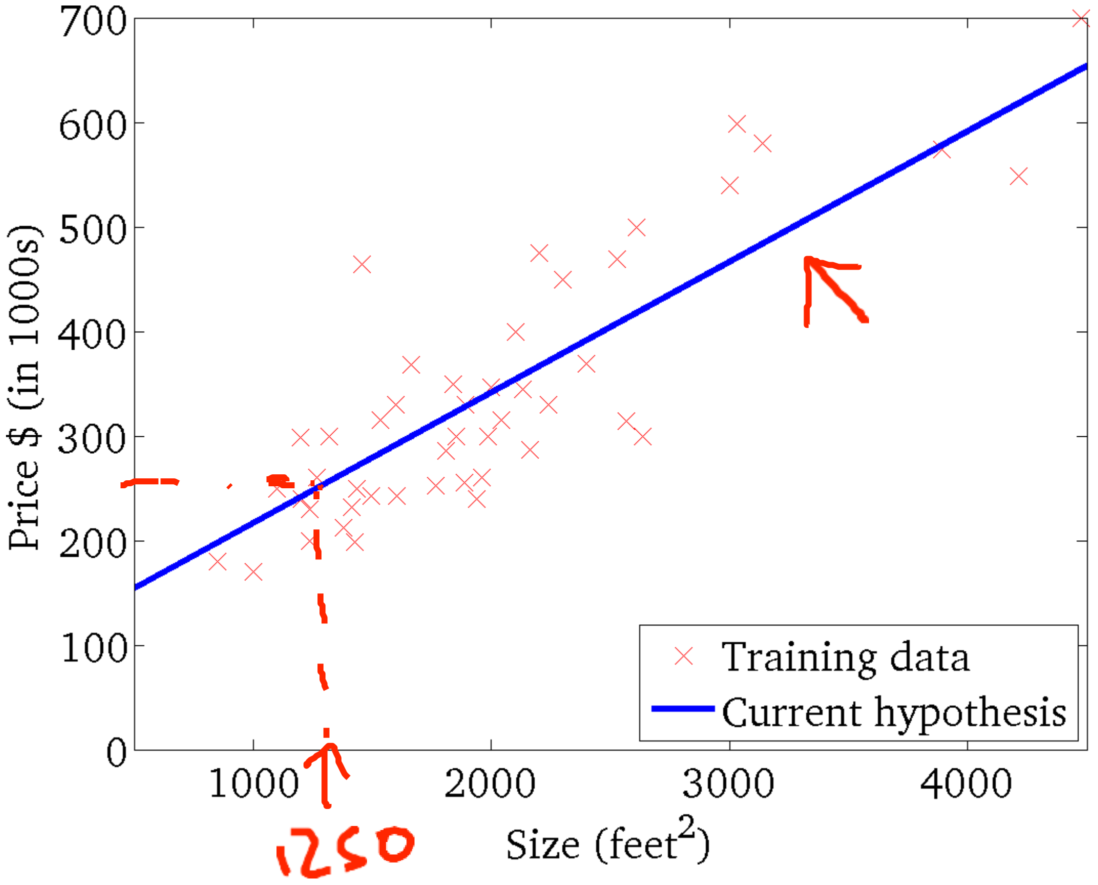
(function of the parameters θ_0, θ_1)





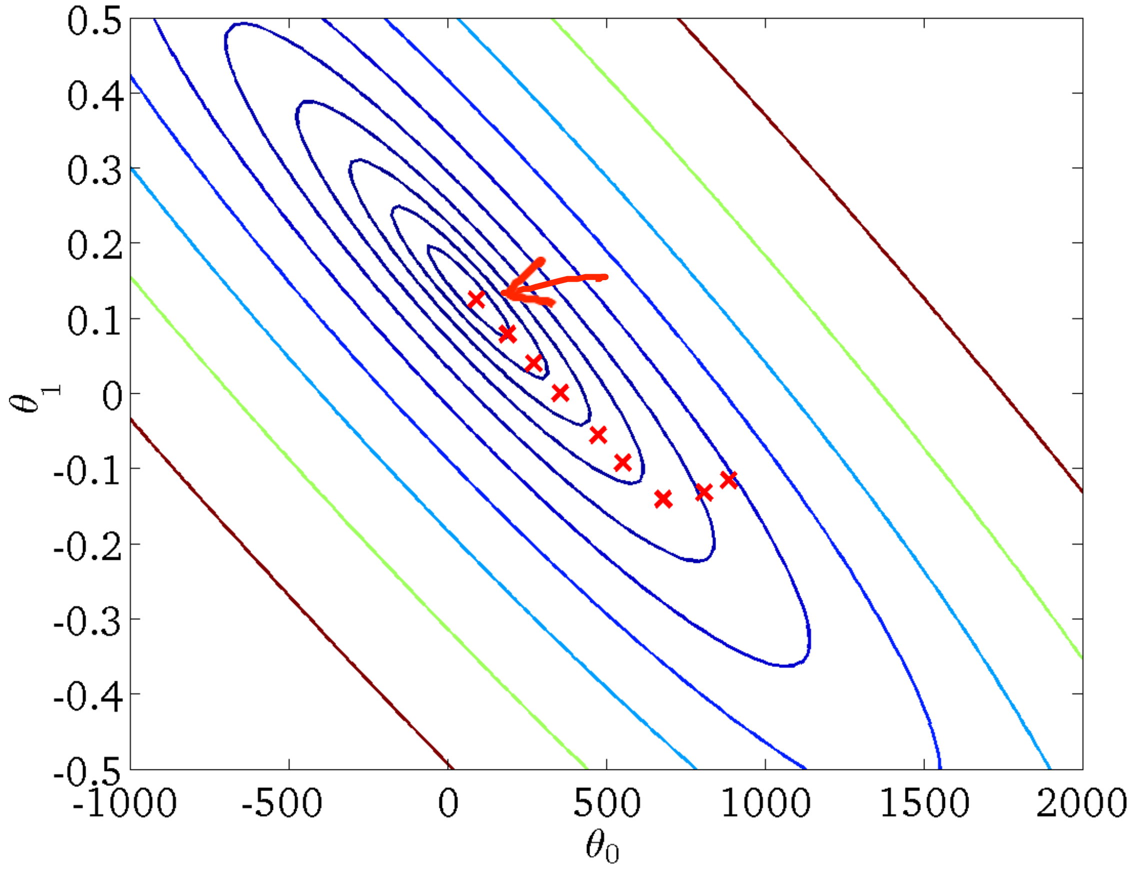
$h_\theta(x)$

(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)



"Batch" Gradient Descent

"Batch": Each step of gradient descent uses all the training examples.

$$\rightarrow \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

"Batch" Gradient Descent is a variation of the Gradient Descent optimization algorithm, which is used to minimize a cost function. It's a fundamental algorithm in machine learning for training a wide range of models, including linear regression, logistic regression, and neural networks.

How Does "Batch" Gradient Descent Work?

- **Batch Processing:** In Batch Gradient Descent, the term "batch" refers to the use of the entire training dataset to calculate the gradient of the cost function with respect to the model parameters. This means that for each iteration of the algorithm, the model uses all available data points to perform an update.
- **Gradient Calculation:** The gradient—a vector of partial derivatives—indicates the direction in which the cost function is increasing most rapidly. By moving in the opposite direction (i.e., descending), we aim to find the function's minimum.
- **Parameter Update:** The model parameters are updated in the direction of the negative gradient scaled by the learning rate, a hyperparameter that controls the size of the step. The equation for updating each parameter ϑ is given by: $\theta=\theta-\eta\cdot\nabla_{\theta}j(\theta)$ where $\boldsymbol{\eta}$ is the learning rate and $\nabla_{\theta}j(\theta)$ is the gradient of the cost function ***J*** with respect to θ.

Advantages of Batch Gradient Descent

- **Simplicity:** It's straightforward to implement and understand.
- **Stable Convergence:** Using the entire dataset provides a clear direction toward the minimum, leading to stable convergence in convex or quadratic problems.
- **Efficiency on Small Datasets:** When the dataset is small enough to fit into memory, batch processing can be computationally efficient due to optimized matrix operations.

Disadvantages of Batch Gradient Descent

- **Scalability:** For very large datasets, loading the entire dataset into memory can be impractical, leading to computational inefficiencies.

- **Speed:** Each iteration can be slow because it must process the entire dataset before making a single update to the parameters.

- **Convergence to Global Minimum:** In non-convex functions, such as those common in deep learning, Batch Gradient Descent can get stuck in local minima.

Alternatives to Batch Gradient Descent

- **Stochastic Gradient Descent (SGD):** Updates parameters for each training example, leading to faster iterations but more variance in the convergence path.
- **Mini-batch Gradient Descent:** Strikes a balance between Batch and SGD by updating parameters after computing the gradient on small subsets of the data. This approach is often used in practice due to its balance between efficiency and stability.
- In summary, Batch Gradient Descent is a powerful but somewhat simplistic tool in the machine learning optimization arsenal. Its use is best suited to situations where datasets are manageable in size and the cost function is relatively simple. For larger datasets or more complex cost functions, its alternatives, such as Mini-batch Gradient Descent, are generally preferred.