Machine Learning

Linear Regression

Some slides taken from course materials of Andrew Ng

Dataset of living area and price of houses in a city

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

This is a training set.

How can we learn to predict the prices of houses of other sizes in the city, as a function of their living area?

Dataset of living area and price of houses in a city

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

Example of supervised learning problem.

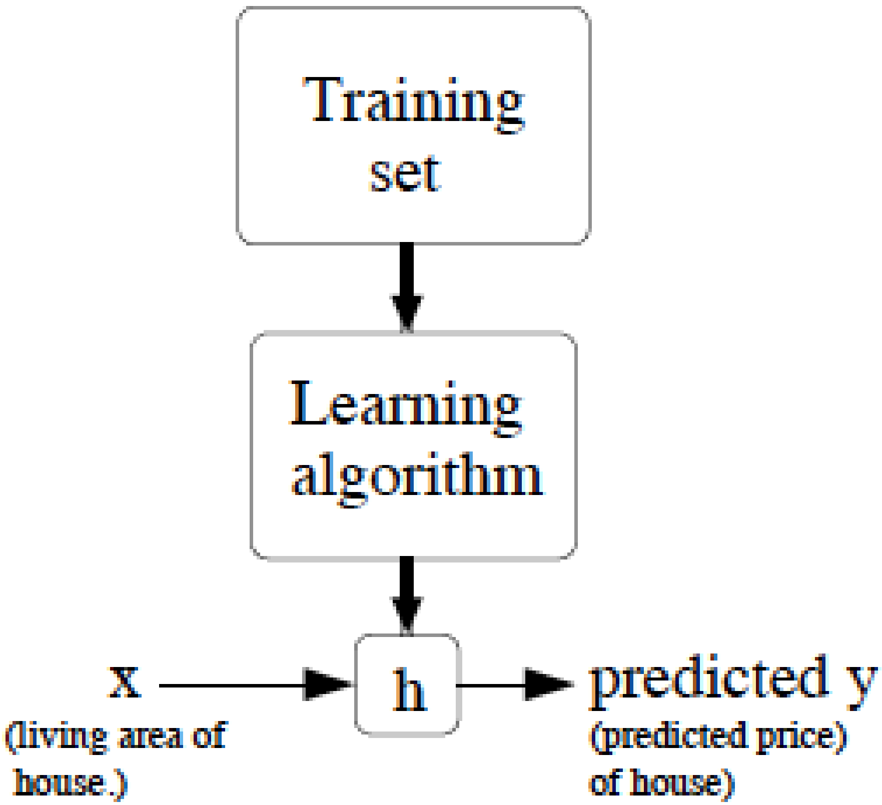
When the target variable we are trying to predict is continuous, regression problem.

Dataset of living area and price of houses in a city

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
$\vdots$	$\vdots$

m = number of **training examples**
x's = input variables / features
y's = output variables / "target" variables
(x,y) - single training example
(x^i, y^i) - specific example (i^{th} training example)
i is an index to training set

How to use the training set?



Learn a function h(x), so that h(x) is a good predictor for the corresponding value of y

h: hypothesis function

What is a Hypothesis

- Machine learning, specifically supervised learning, hypothesis can be described as the desire to use available data to learn a function that best maps inputs to outputs.
- Technically, this is a problem called function approximation, where we are approximating an unknown target function (that we assume exists) that can best map inputs to outputs on all possible observations from the problem domain.
- An example of a model that approximates the target function and performs mappings of inputs to outputs is called a hypothesis in machine learning.
- Learning for a machine learning algorithm involves navigating the chosen space of hypothesis toward the best or a good enough hypothesis that best approximates the target function

Linear Regression

- The power of regression coefficient should be maximum 1

  y=a+bx--is linear regression

  y=a+b^2x--is not linear regression

- The first order derivative w.r.t parameters should not have parameters as coefficient

  y=a+b1x1+b2x2--is linear regression

  y=a+log(b)x--is not linear regression

Simple (univariate) Linear Regression

Linear and only one independent variable

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

Multiple (multivariate) Linear Regression

Linear and more than one independent variable

No_of_friend_request= **a+b*(no_of-pics_uploaded)+c*(education)+d*(age)**

Multiple features (variables).

Size ($feet^2$)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Hypothesis:

For univariate linear regression:

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

For multi-variate linear regression:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

How to represent hypothesis h?

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

θ_i are **parameters**
- θ_0 is zero condition
- θ_1 is gradient
θ: vector of all the parameters

We assume y is a linear function of x
Univariate linear regression
How to learn the values of the parameters?

Digression: Multivariate linear regression

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
$\vdots$	$\vdots$	$\vdots$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

How to represent hypothesis h?

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

θ_i are **parameters**

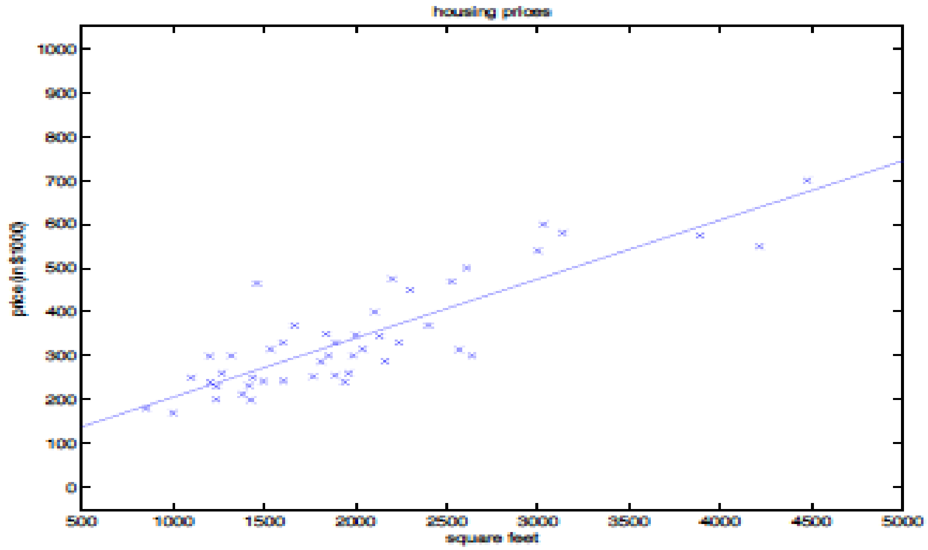
- θ_0 is zero condition
- θ_1 is gradient

We assume y is a linear function of x

Univariate linear regression

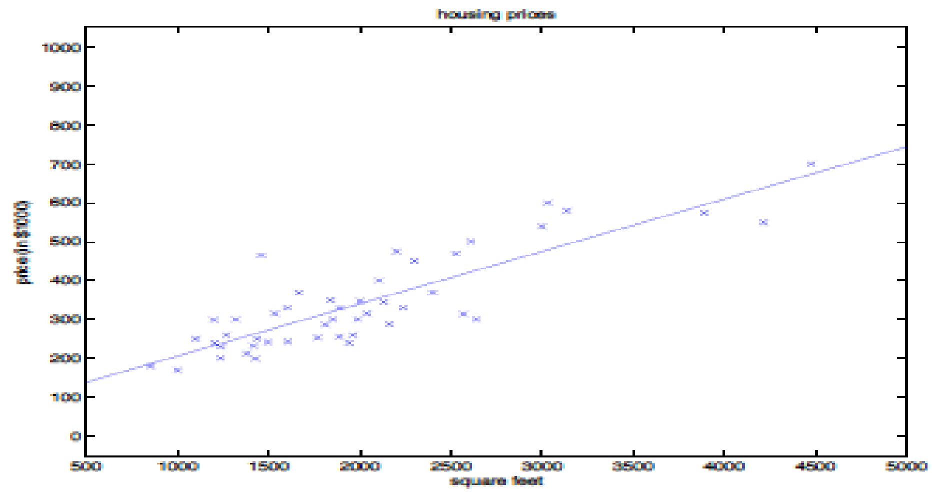
How to learn the values of the parameters θ_i?

Intuition of hypothesis function



- We are attempting to fit a straight line to the data in the training set
- Values of the parameters decide the equation of the straight line
- Which is the best straight line to fit the data?

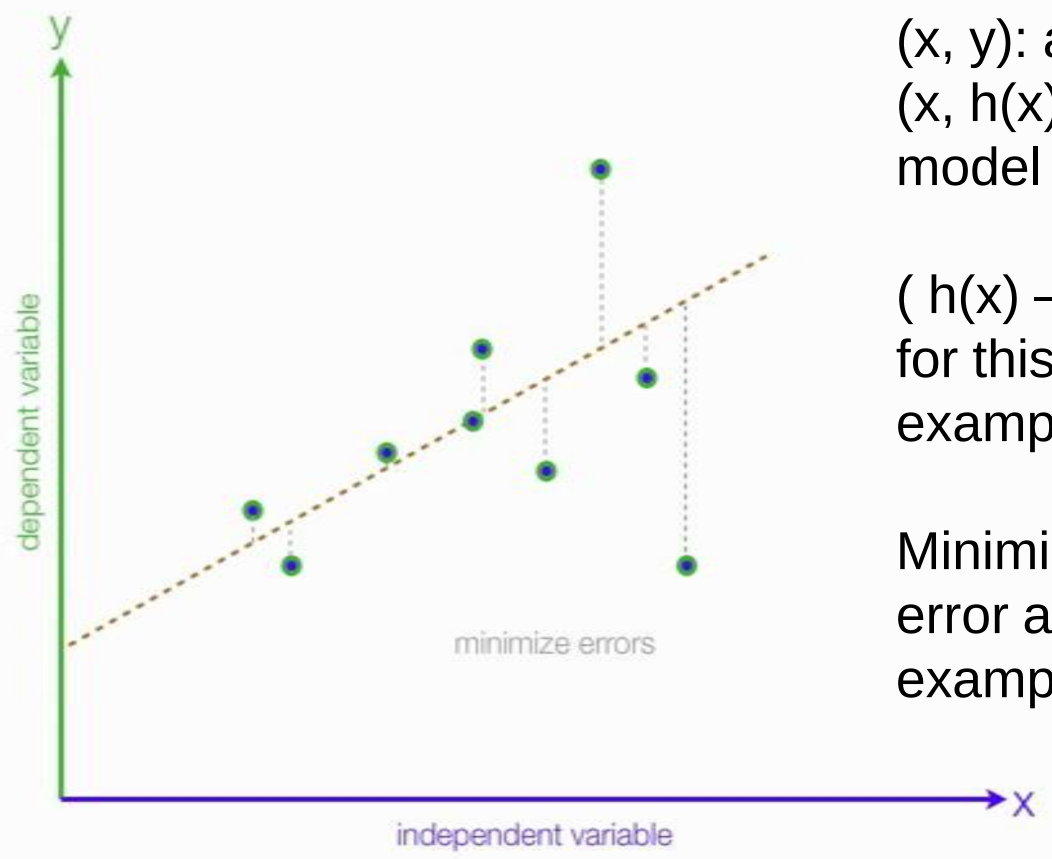
Intuition of hypothesis function



- Which is the best straight line to fit the data?
- How to learn the values of the parameters θ_i?

- Choose the parameters such that the prediction is close to the actual y-value for the training examples

How good is the prediction given by the straight line?



(x, y): a training example
(x, h(x)): prediction of the model

(h(x) – y): prediction error for this particular training example

Minimize the prediction error across all training examples

Cost function

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

- Measure of how close the predictions are to the actual y-values
- Average over all the m training instances

- Squared error cost function $J(\theta)$
- Choose parameters θ so that $J(\theta)$ is minimized

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

Parameters: θ_0, θ_1

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

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

Minimize $J(\theta_0, \theta_1)$

θ_0, θ_1

$$\min_{\theta_0, \theta_1} \sum_{i=1}^{n} \left([\theta_1 x_i + \theta_0] - y_i \right)^2$$

$$\frac{\partial J}{\partial \theta_1} = \sum_{i=1}^{n} 2(\theta_1 x_i + \theta_0 - y_i) x_i$$

$$\frac{\partial J}{\partial \theta_0} = \sum_{i=1}^{n} 2(\theta_1 x_i + \theta_0 - y_i)$$

$$\frac{\partial J}{\partial \theta_1} = 0$$

$$\frac{\partial J}{\partial \theta_0} = 0$$

$$\theta_1 = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$\theta_0 = \frac{\sum_{i=1}^{n} y_i - \theta_1 \sum_{i=1}^{n} x_i}{n}$$

Learning parameters in Method of least square regression

- For the 2-*d* problem

- To find the values for the coefficients which minimize the objective function we take the partial derivates of the objective function (SSE) with respect to the coefficients. Set these to 0, and solve.

$$\beta_1 = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}$$

$$\beta_0 = \frac{\sum y - \beta_1 \sum x}{n}$$

Learning parameters in Method of least square regression using Calculus

Height	Weight
151	63
174	81
138	56
186	91
128	47
136	57
179	76
163	72
152	62
131	48

Computing the Relationship Model

X	Y	X^2	XY
151	63	22801	9153
174	81	30276	14094
138	56	19044	7728
186	91	34596	16926
128	47	16384	6016
136	57	18496	7752
179	76	32041	13608
163	72	26569	11736
152	62	23104	9424
131	48	17161	6288
$\sum x=1538$	$\sum y=653$	$\sum X^2=240472$	$\sum xy=103081$

Computing the Relationship Model

$\beta 1 = (10X103081-1538X653)/(10X240472-2365444)$

$= 26{,}496 /(39{,}276)$

$=0.6746$

$\beta 0 = -38.4551$

Thus if we want to predict the weight of a student whose height is=170

$Y = 0.6746\ X170 - 38.4551$

$=76.23$

Gradient-Based Optimization

Most Machine/deep learning algorithms involve optimization of some sort.

Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x.

The function we want to minimize or maximize is called the objective function or criterion.

When we are minimizing it, we may also call it the cost function, loss function, or error function

Calculus behind Gradient-Based Optimization

We often denote the value that minimizes or maximizes a function with a superscript $*$.
For example, we might say $x^* = \text{argmin}\, f(x)$.

Suppose we have a function $y = f(x)$, where both x and y are real numbers.

The derivative of this function is denoted as $f'(x)$ or as dy/dx
The derivative $f'(x)$ gives the slope of f (x) at the point x

It specifies how to scale a small change in the input in order to obtain the corresponding change in the output:
$f(x + \varepsilon) = f(x) + \varepsilon f'(x)$

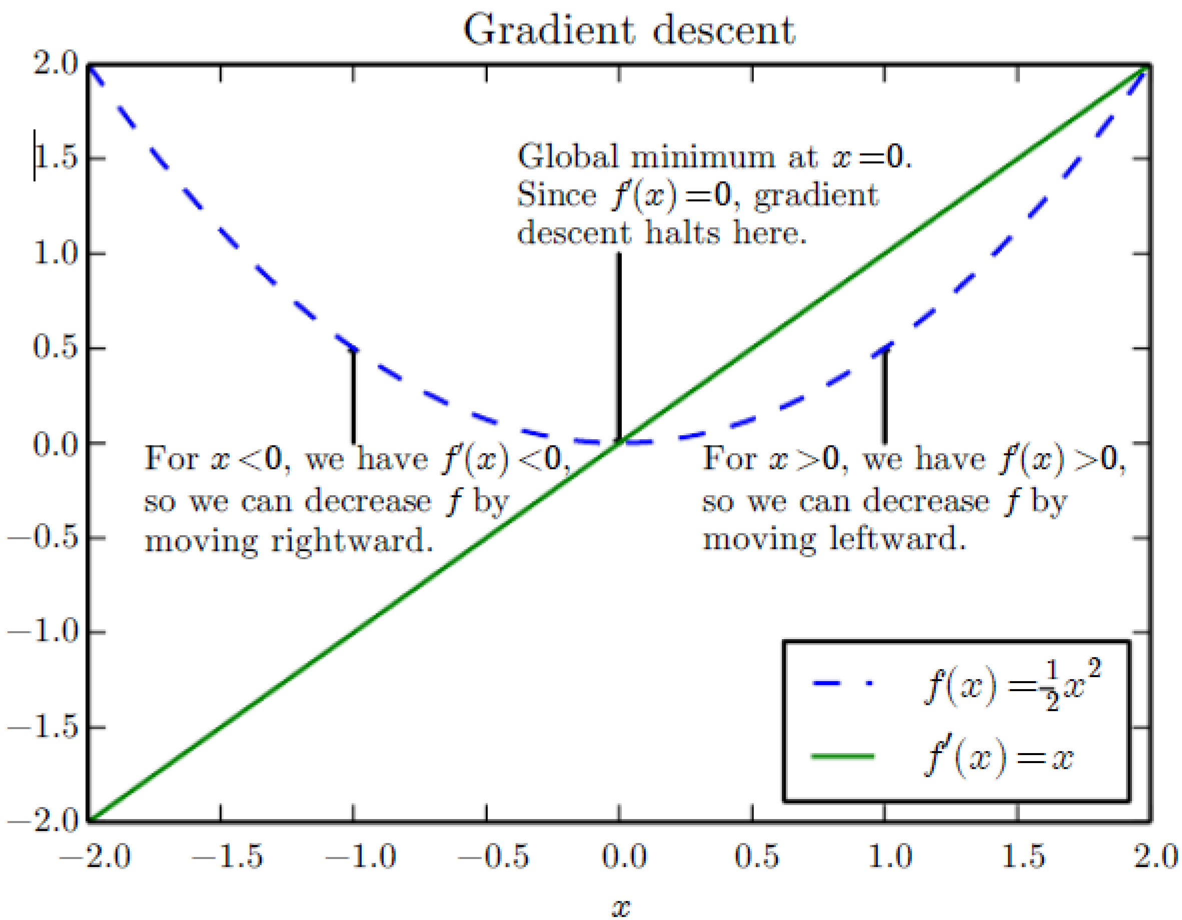


Figure 4.1: An illustration of how the derivatives of a function can be used to follow the function downhill to a minimum. This technique is called *gradient descent*.

Gradient-Based Optimization

For example, we know that $f(x - \mathcal{E}\, sign(\, f'(x)))$ is less than f(x) for small enough Ɛ. We can thus reduce f(x) by moving x in small steps with opposite sign of the derivative (dy/dx). This technique is called gradient descent (Cauchy, 1847).

If $f'(x)$>0 , x should be decreased and if $f'(x)$<0

x should be increased

Gradient descent algorithm checks the gradient and always move opposite direction of the gradient to reach to local or global minimum

Why Gradient Descent is popular than Calculus

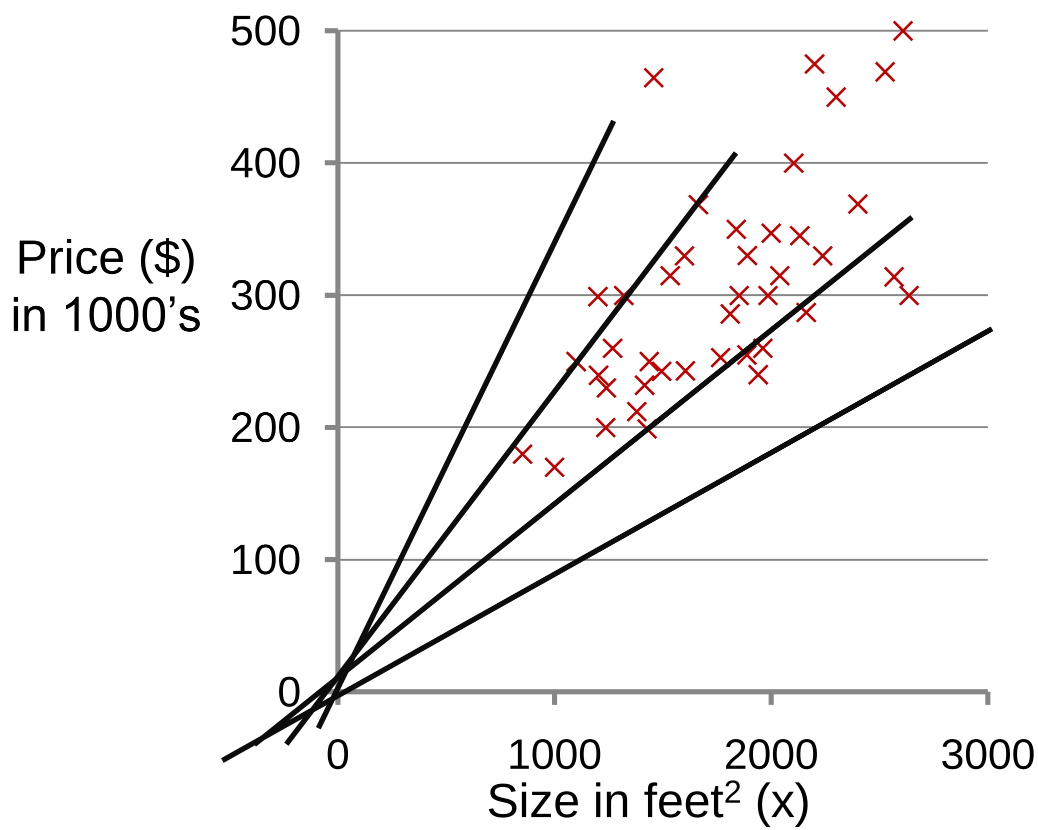
For some critical mathematical function it is difficult to find x by solving the equation $f'(x) = 0$

Gradient Descent is an first order iterative optimization algorithm to findthe minimum of a function

However, Gradient Descent is susceptible to local minimum in case of non-convex function

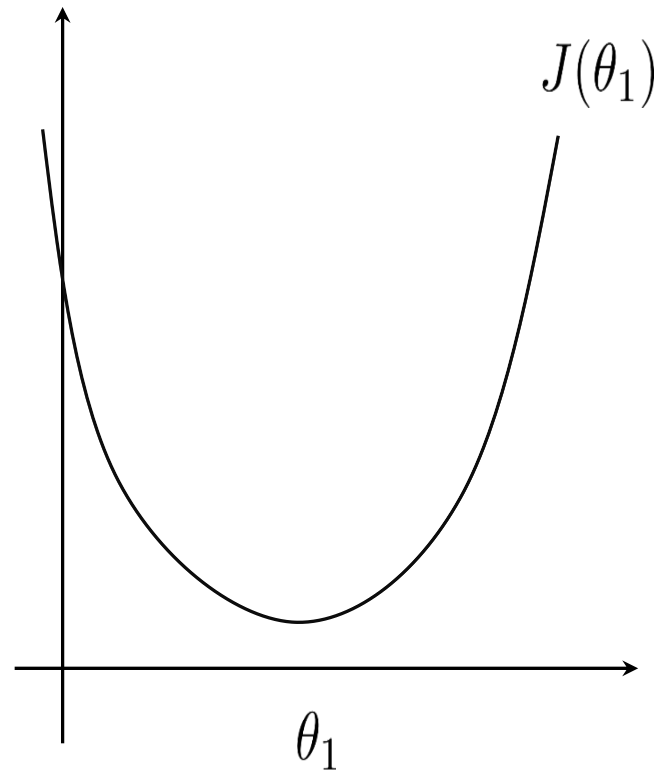
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

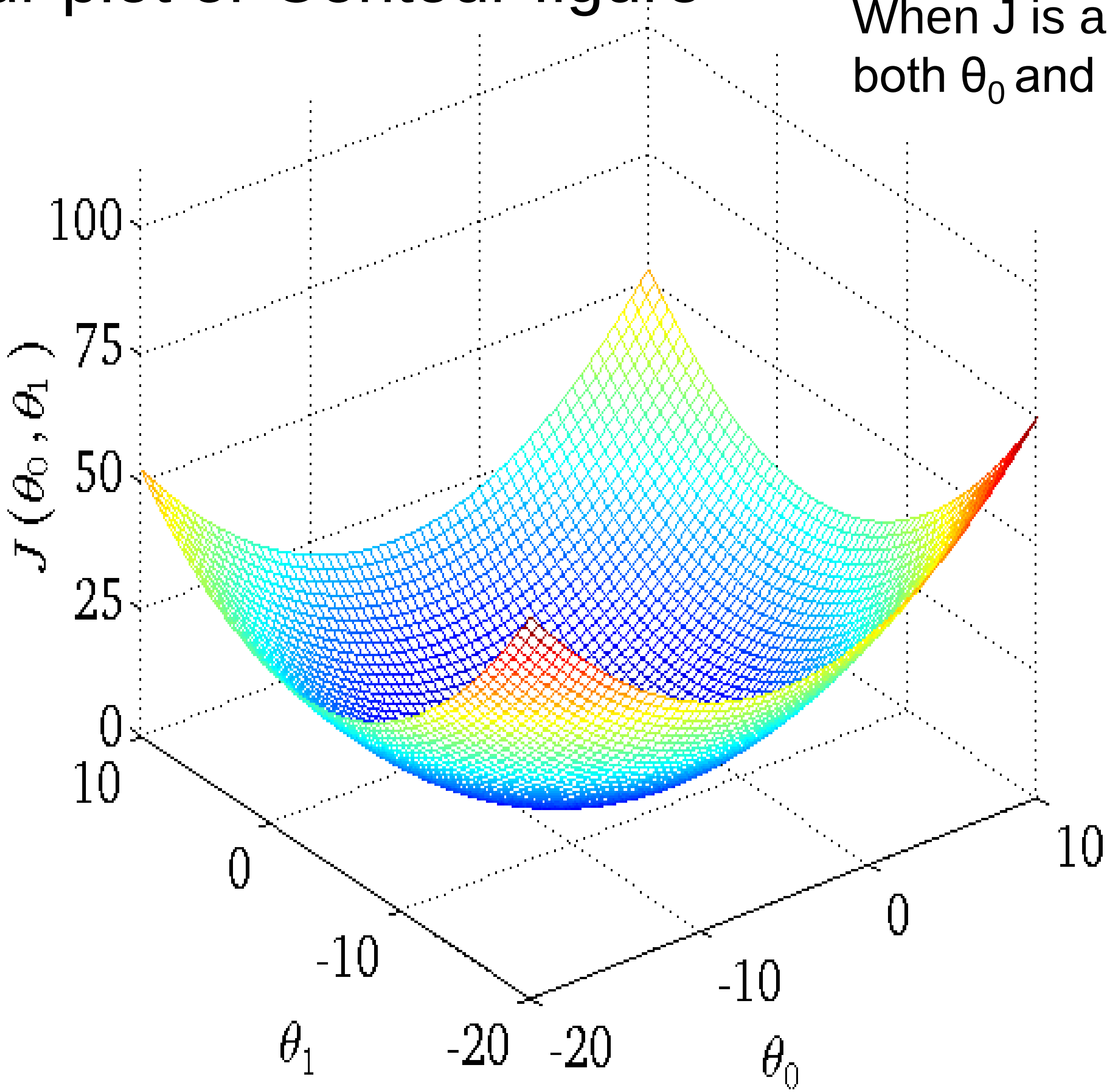
(function of the parameters θ_0, θ_1)



For simplicity, assume Θ_0 is a constant

Contour plot or Contour figure

When J is a function of both θ_0 and θ_1



Minimizing a function

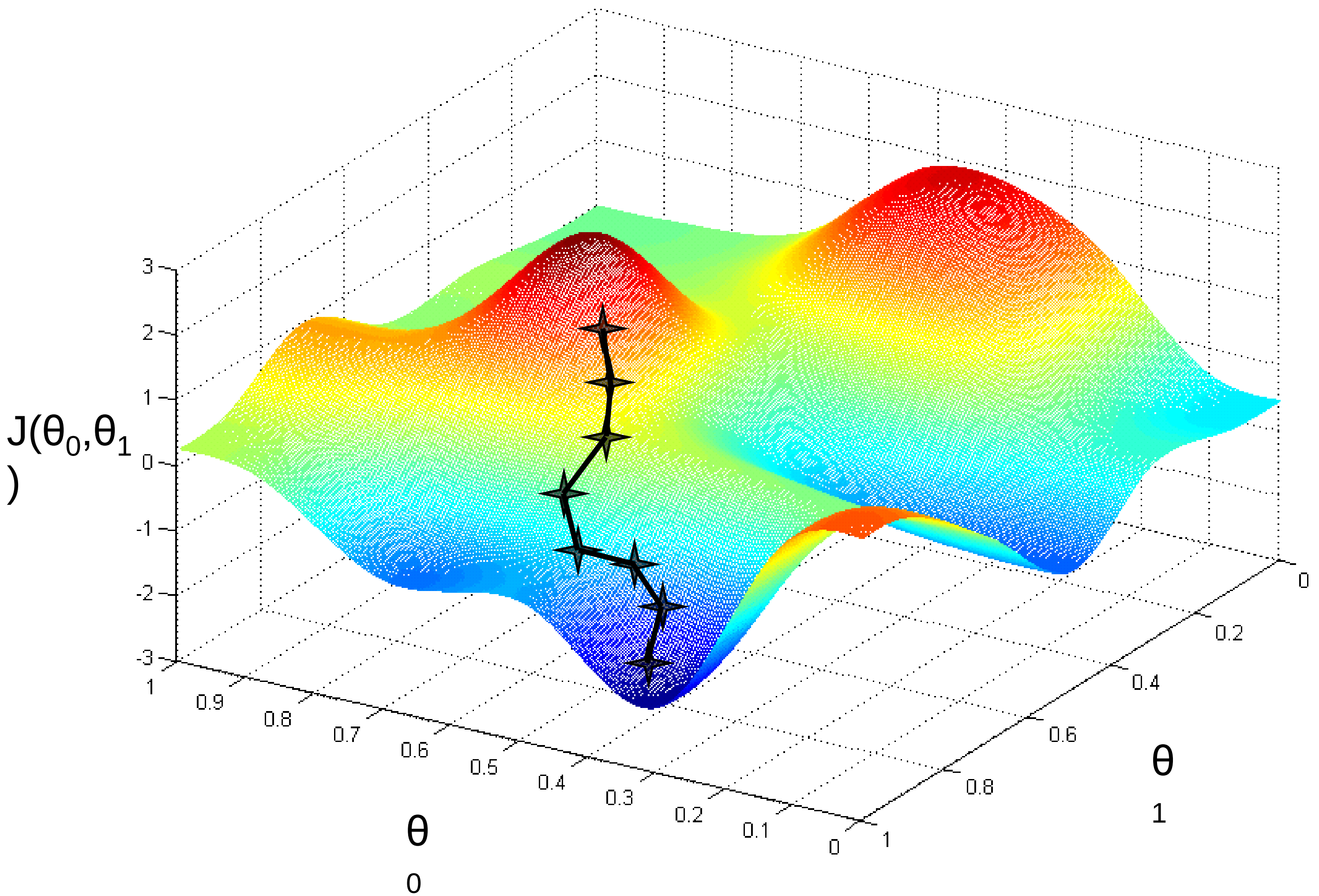
- For now, let us consider some arbitrary function (not necessarily a cost function)
- Analytical minimization not scalable to complex functions of hundreds of parameters
- Algorithm called gradient descent
  - Efficient and scalable to thousands of parameters
  - Used in many applications of minimizing functions

Have some function $J(\theta_0, \theta_1)$

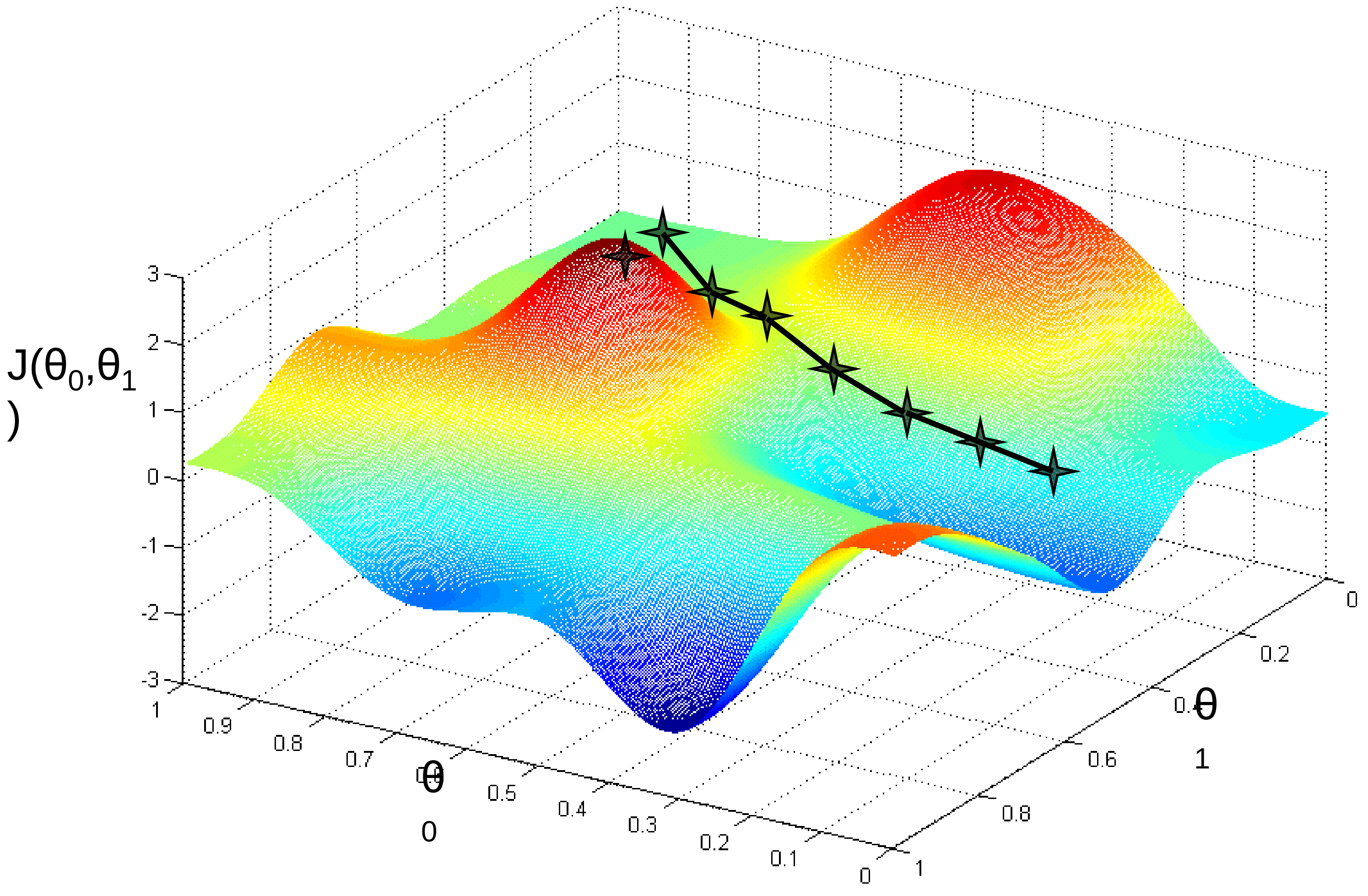
Wan
t $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

- **Outline:**
  - Start with some θ_0, θ_1
  - Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

    until we hopefully end up at a minimum


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
θ
0
0
0.2
0.4
0.6
0.8
1
θ
1

If the function has multiple local minima, where one starts can decide which minimum is reached



Gradient descent algorithm

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

α is the learning rate – more on this later

Gradient descent algorithm

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1) \\
&\}
\end{aligned}
$$

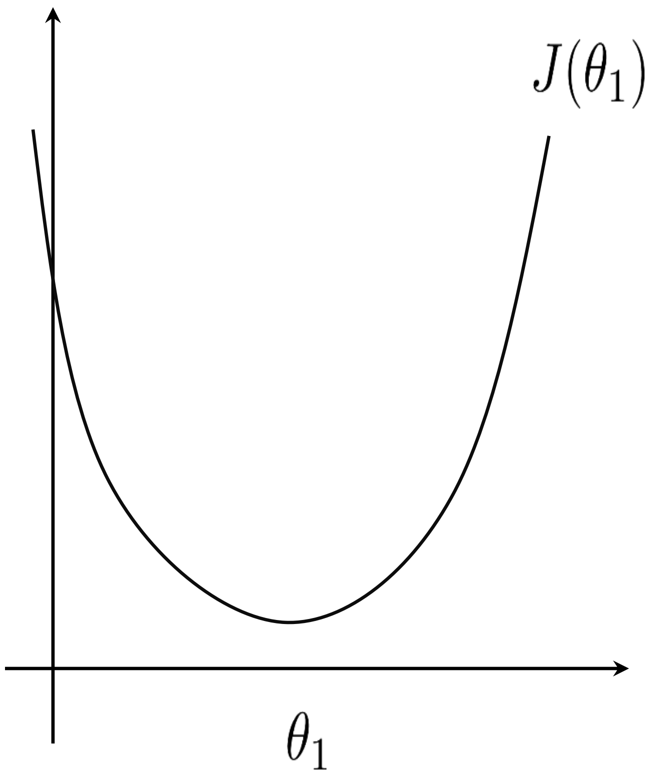
Correct: Simultaneous update

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

Incorrect:

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

For simplicity, let us first consider a function of a single variable



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

If the derivative is positive, reduce value of θ_1

If the derivative is negative, increase value of θ_1

The learning rate

- Do we need to change learning rate over time?
  - No, Gradient descent can converge to a local minimum, even with the learning rate α fixed
  - Step size adjusted automatically
- But, value needs to be chosen judiciously
  - If α is too small, gradient descent can be slow to converge
  - If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

The Step size and learning rate

- Step size adjusted automatically
- Big steps when far from optimal solution
- Small steps when close to optimal solution
- Step size= slope* learning rate
- ✓ Gradient descent converges when step size is close to zero (In practice step size less or equal to .001)
- ✓ Step size is close to zero when slope is close to zero

Gradient descent for univariate linear regression

Gradient descent algorithm

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\qquad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\
&\qquad\qquad (\text{for } j = 1 \text{ and } j = 0) \\
&\}
\end{aligned}
$$

Linear Regression Model

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

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

Gradient descent for univariate linear regression

$$\text{repeat until convergence } \{$$

$$\left.\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \end{aligned}\right\}$$

$$\}$$

update θ_0 and θ_1 simultaneously

“Batch” Gradient Descent

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

There are other variations like “stochastic gradient descent” (used in learning over huge datasets)