Supervised Machine Learning

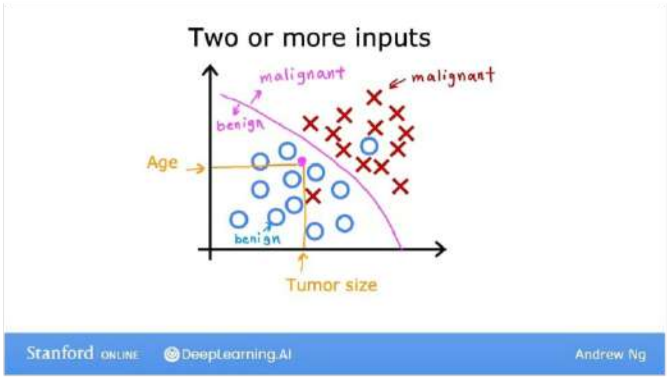
Regression

predict a number from many possible outputs

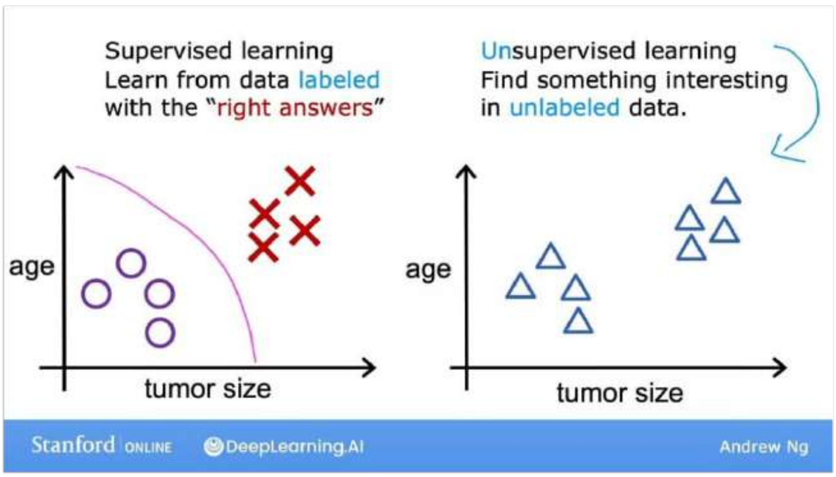
Classification

small number of outputs

tumor → benign
tumor → malignant



Supervised vs Unsupervised



Unsupervised Machine Learning

Clustering

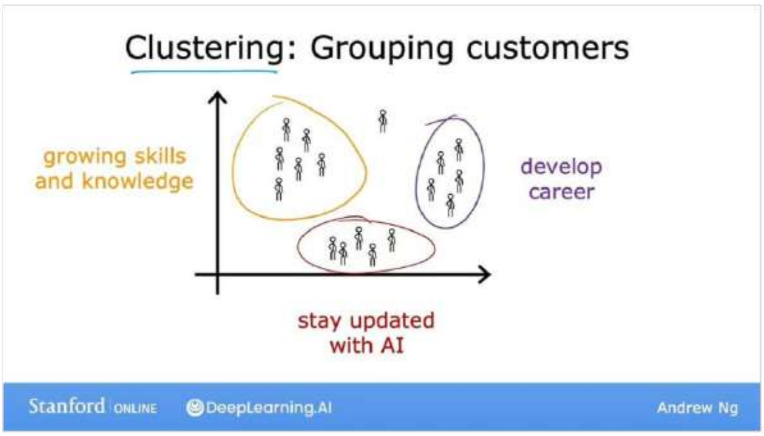
take data without labels and automatically group them to clusters

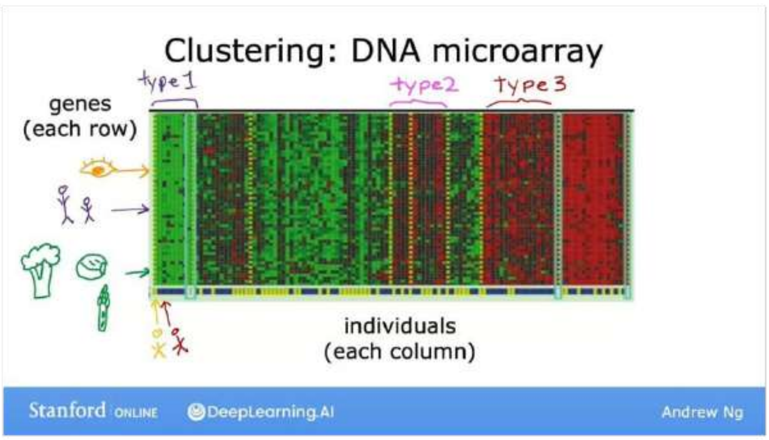
Anomaly Detection

Find unusual data points

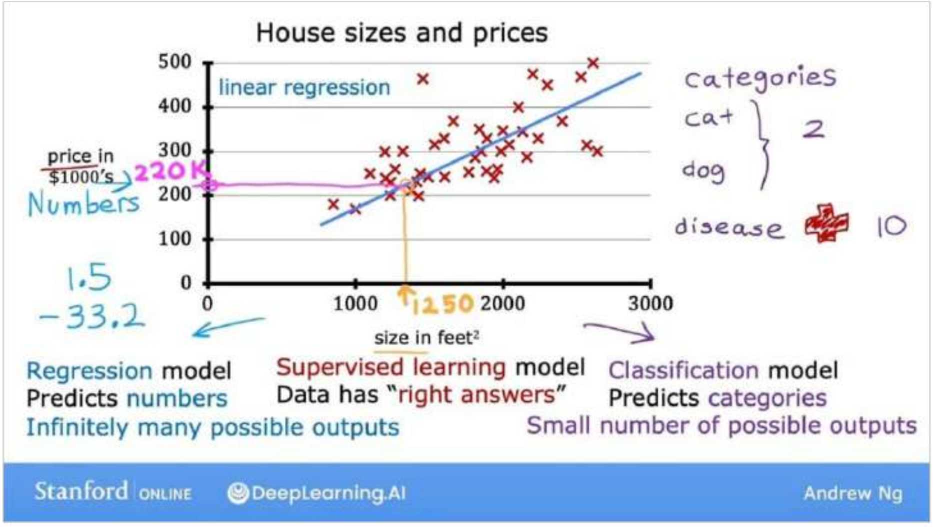
Dimensionality Reduction

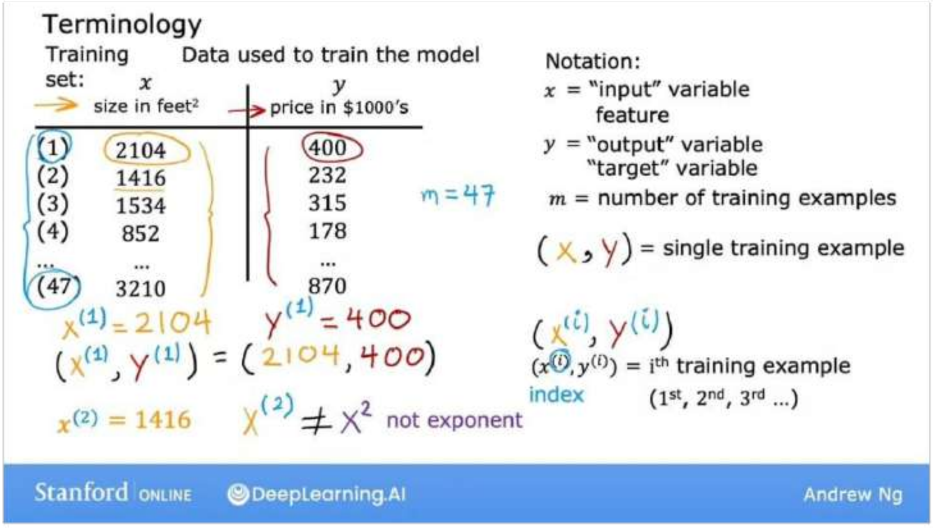
Compress data using fewer numbers

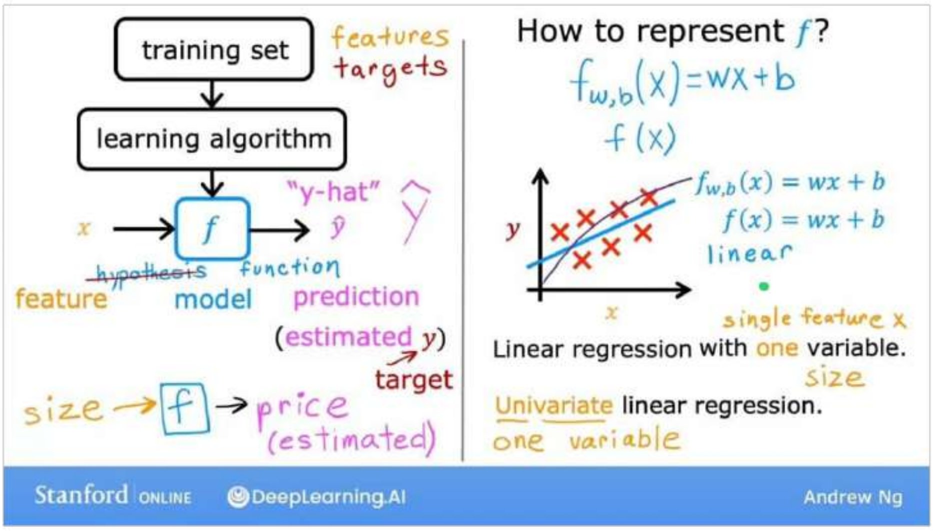




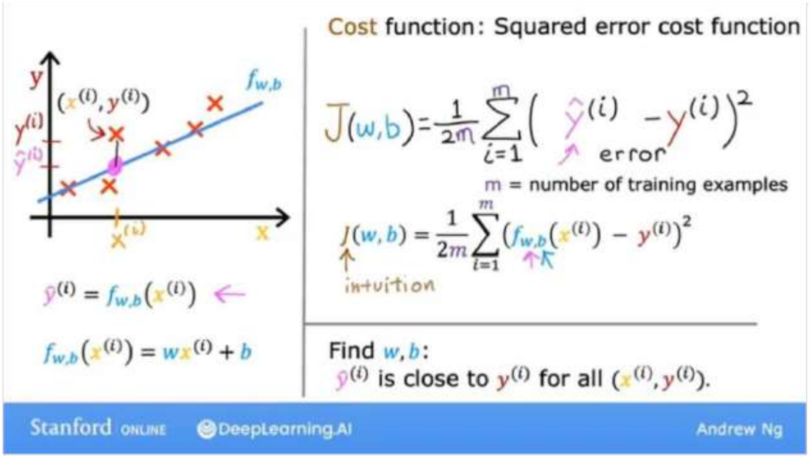
Linear Regression

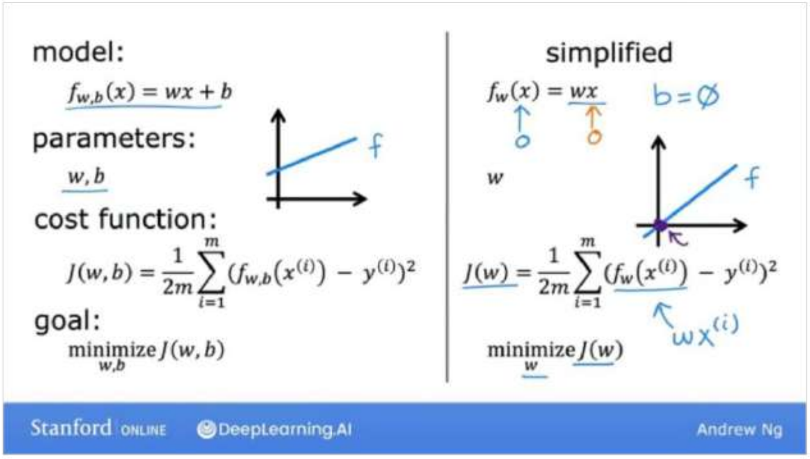


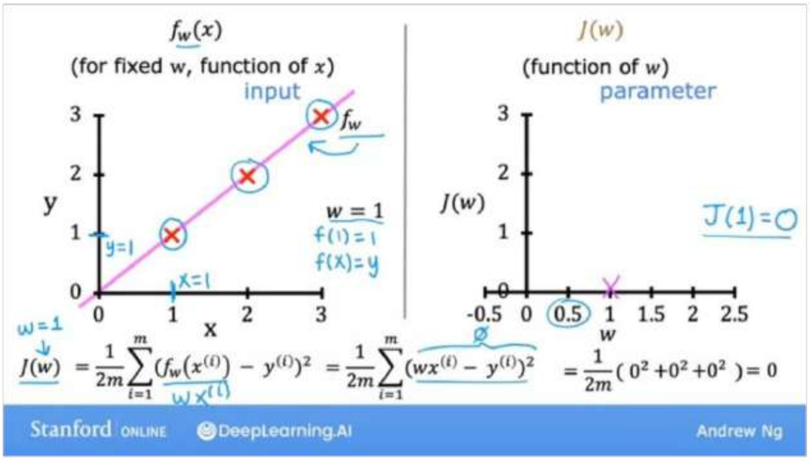


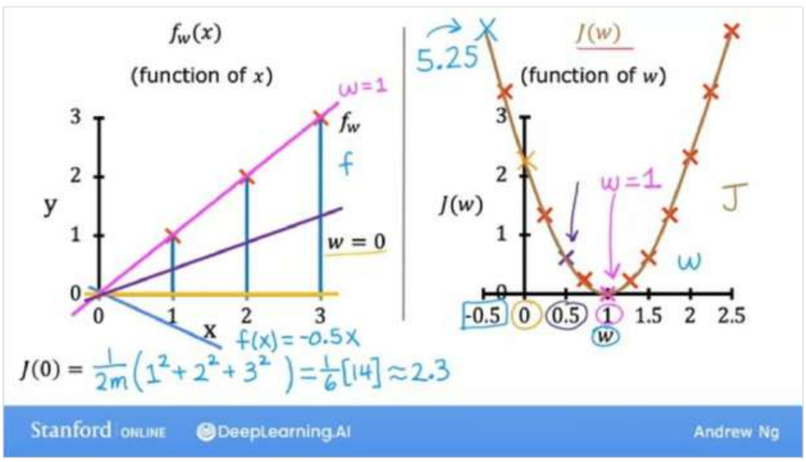


Cost function



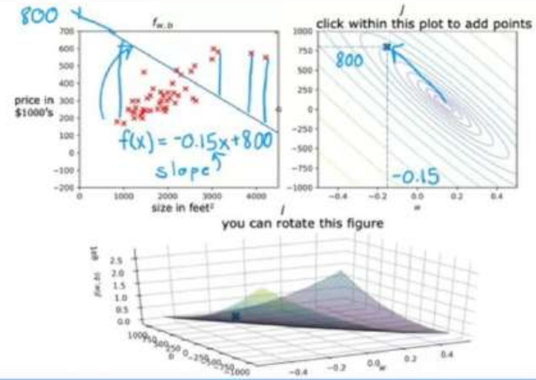


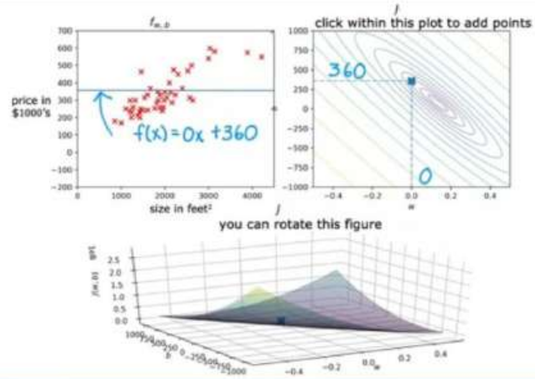


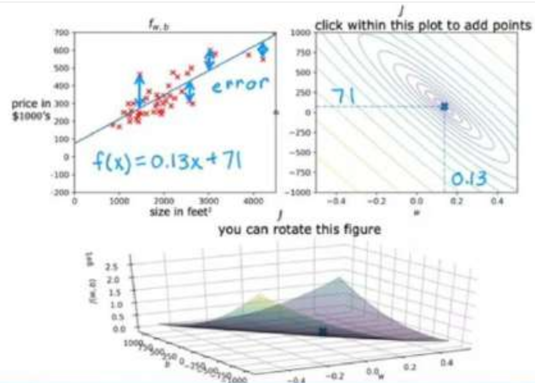


Visualize cost function

Model	$f_{w,b}(x) = wx + b$
Parameters	w, b
Cost Function	$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$
Objective	$\underset{w,b}{\text{minimize}}\, J(w,b)$







Gradient Descent

Have some function $J(w,b)$ — for linear regression or any function

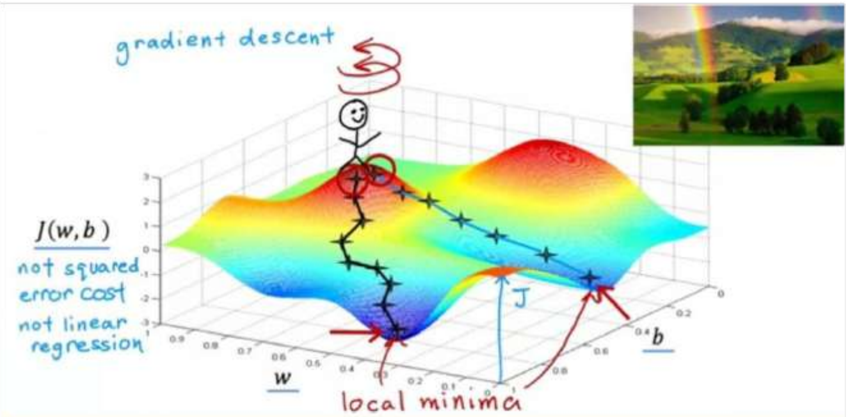
Want $\min\limits_{w,b} J(w,b)$ — $\min\limits_{w_1,\dots,w_n,b} J(w_1,w_2,\dots,w_n,b)$

Outline:

- Start with some w,b (set $w=0$, $b=0$)
- Keep changing w,b to reduce $J(w,b)$
- Until we settle at or near a minimum

may have >1 minimum

J not always



Gradient descent algorithm

repeat until convergence {

learning rate — derivative

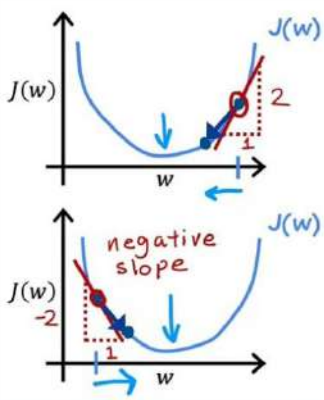
$$w = w - \alpha \boxed{\frac{\partial}{\partial w} J(w,b)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

$J(w)$

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

$\min\limits_{w} J(w)$

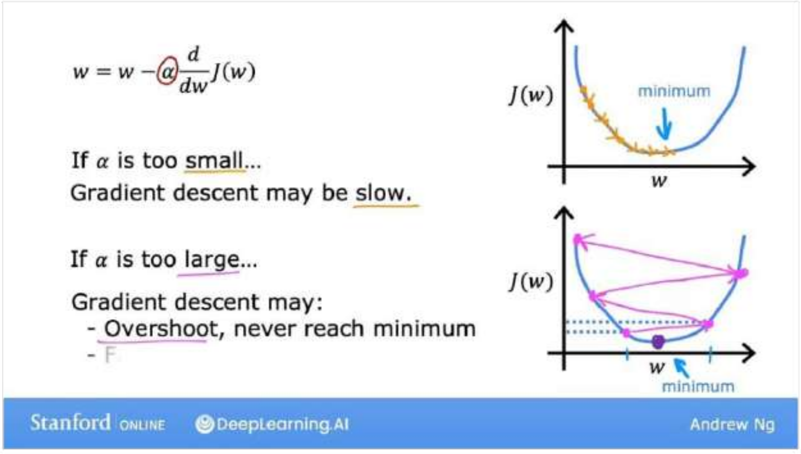


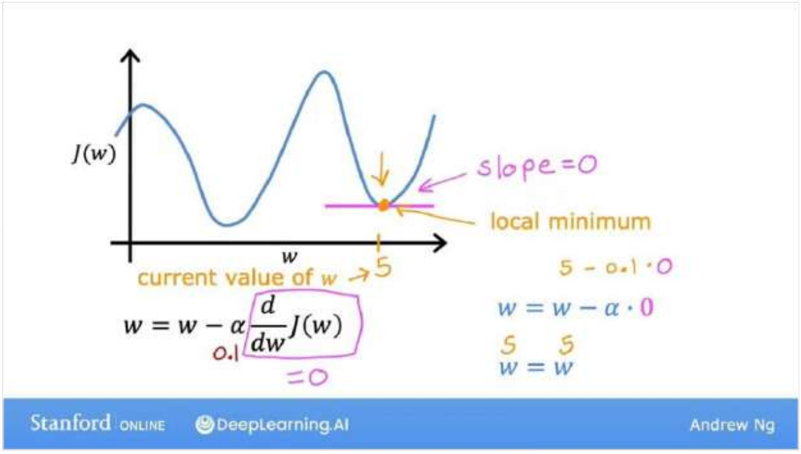
$$w = w - \alpha \underbrace{\boxed{\frac{d}{dw} J(w)}}_{>0}$$

$w = w - \alpha \cdot (\text{positive number})$

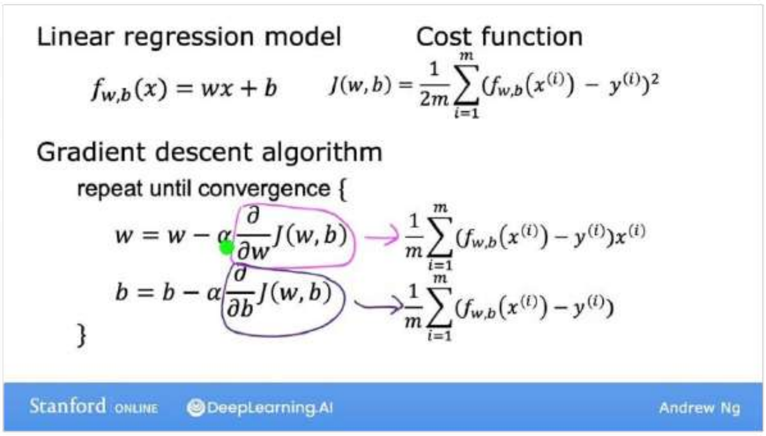
$\frac{d}{dw} J(w) < 0$

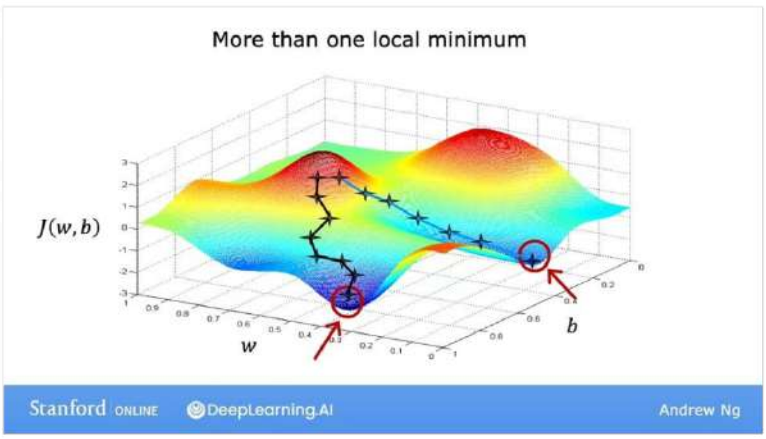
$w = w - \alpha \cdot (\text{negative number})$



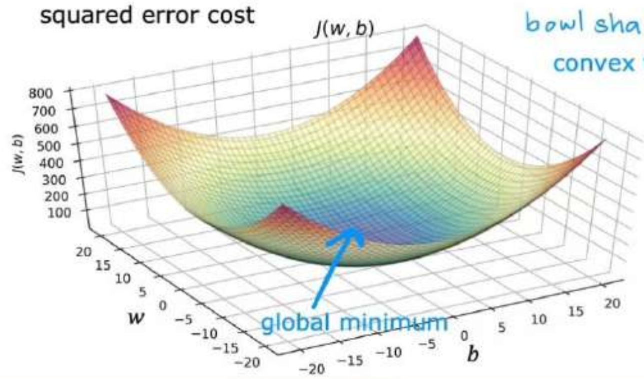


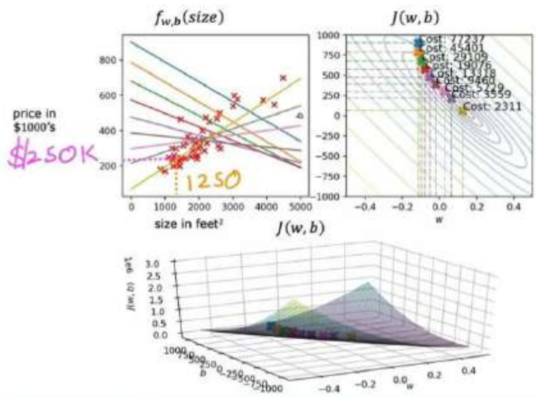
Gradient Disecent for Linear Regression





squared error cost





"Batch" gradient descent

DeepLearning.AI THE BATCH

other gradient descent: subsets

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

	x size in feet²	y price in $1000's
(1)	2104	400
(2)	1416	232
(3)	1534	315
(4)	852	178
...	...	...
(47)	3210	870

$m = 47$

$$\sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Week 2

Multiple Linear Regression

Multiple features (variables)

	Size in feet²	Number of bedrooms	Number of floors	Age of home in years	Price ($) in $1000's
	x_1	x_2	x_3	x_4	
	2104	5	1	45	460
$i=2$	1416	3	2	40	232
	1534	3	2	30	315
	852	2	1	36	178
	...	...	...	...	...

$j = 1 \ldots 4$

$n = 4$

x_j = j^{th} feature

n = number of features

$\vec{x}^{(i)}$ = features of i^{th} training example

$x_j^{(i)}$ = value of feature j in i^{th} training example

$\vec{x}^{(2)} = [1416 \ \ 3 \ \ 2 \ \ 40]$

$x_3^{(2)} = 2$

Stanford ONLINE DeepLearning.AI Andrew Ng

Model:

Previously: $f_{w,b}(x) = wx + b$

$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b$

example

$f_{w,b}(x) = 0.1 x_1 + 4 x_2 + 10 x_3 + -2 x_4 + 80$

size #bedrooms #floors years base price

$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$

Stanford ONLINE DeepLearning.AI Andrew Ng

$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$

$\vec{w} = [w_1 \ \ w_2 \ \ w_3 \ \ldots \ w_n]$ parameters of the model

b is a number

vector $\vec{x} = [x_1 \ \ x_2 \ \ x_3 \ \ldots \ x_n]$

$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_n x_n + b$

dot product

multiple linear regression

(not multivariate regression)

Stanford ONLINE DeepLearning.AI Andrew Ng

Multiple features

Multiple features (variables)

	Size in feet²	Number of bedrooms	Number of floors	Age of home in years	Price ($) in $1000's
	x_1	x_2	x_3	x_4	
	2104	5	1	45	460
$i=2$	1416	3	2	40	232
	1534	3	2	30	315
	852	2	1	36	178
	...	...	...	...	...

$j = 1 \ldots 4$

$n = 4$

x_j = j^{th} feature
n = number of features
$\vec{x}^{(i)}$ = features of i^{th} training example
$x_j^{(i)}$ = value of feature j in i^{th} training example

$\vec{x}^{(2)} = [1416 \;\; 3 \;\; 2 \;\; 40]$

$x_3^{(2)} = 2$

Model:

Previously: $f_{w,b}(x) = wx + b$

$f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$

example

$f_{w,b}(x) = 0.1x_1 + 4x_2 + 10x_3 + -2x_4 + 80$

size #bedrooms #floors years base price

$f_{w,b}(\mathrm{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$

Stanford ONLINE DeepLearning.AI Andrew Ng

$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$

$\vec{w} = [w_1 \;\; w_2 \;\; w_3 \ldots w_n]$ parameters of the model

b is a number

vector $\vec{x} = [x_1 \;\; x_2 \;\; x_3 \ldots x_n]$

$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b$

dot product

multiple linear regression

(not multivariate regression)

Stanford ONLINE DeepLearning.AI Andrew Ng

Vectorization

Parameters and features

$\vec{w} = [w_1 \quad w_2 \quad w_3]$ $\quad n=3$

b is a number

$\vec{x} = [x_1 \quad x_2 \quad x_3]$

linear algebra: count from 1

NumPy

w[0] w[1] w[2]

```
w = np.array([1.0,2.5,-3.3])
b = 4                        x[0] x[1] x[2]
x = np.array([10,20,30])
```

code: count from 0

Without vectorization $n = 100{,}000$

$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$

```
f = w[0] * x[0] +
    w[1] * x[1] +
    w[2] * x[2] + b
```

Without vectorization

$f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} w_jx_j\right) + b$ $\quad \sum_{j=1}^{n} \rightarrow j=1\ldots n$ $\quad 1,2,3$

range(0,n) → j = 0…n-1

```
f = 0        range(n)
for j in range(0,n):
    f = f + w[j] * x[j]
f = f + b
```

Vectorization

$f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$

```
f = np.dot(w,x) + b
```


Without vectorization

```
for j in range(0,16):
    f = f + w[j] * x[j]
```

t_0 `f + w[0] * x[0]`

t_1 `f + w[1] * x[1]`

…

t_{15} `f + w[15] * x[15]`

Vectorization

```
np.dot(w,x)
```

t_0

w[0]	w[1]	…	w[15]
*	*	…	*
x[0]	x[1]	…	x[15]

in parallel

t_1

w[0]*x[0] + w[1]*x[1] +…+ w[15]*x[15]

efficient → scale to large datasets

Gradient descent

$\vec{w} = (w_1 \quad w_2 \quad \cdots \quad w_{16})$ $\cancel{b}$ parameters

derivatives $\vec{d} = (d_1 \quad d_2 \quad \cdots \quad d_{16})$

```
w = np.array([0.5, 1.3, … 3.4])
d = np.array([0.3, 0.2, … 0.4])
```

compute $w_j = w_j - \underbrace{0.1}_{\text{learning rate } \alpha} d_j$ for $j = 1 \ldots 16$

Without vectorization

$w_1 = w_1 - 0.1d_1$

$w_2 = w_2 - 0.1d_2$

$\vdots$

$w_{16} = w_{16} - 0.1d_{16}$

```
for j in range(0,16):
    w[j] = w[j] - 0.1 * d[j]
```

With vectorization

$\vec{w} = \vec{w} - 0.1\vec{d}$

w

−

0.1*d

```
w = w - 0.1 * d
```

Gradient Descent for multiple linear regression

	Previous notation	Vector notation
Parameters	$w_1, \cdots, w_n$ b	vector of length n $\vec{w} = [w_1 \ \cdots \ w_n]$ b still a number
Model	$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + \cdots + w_n x_n + b$	$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$ dot product
Cost function	$J(\underbrace{w_1, \cdots, w_n}, b)$	$J(\vec{w}, b)$

Gradient descent

Previous notation	Vector notation

repeat {
$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\underbrace{w_1, \cdots, w_n}, b)$
$b = b - \alpha \frac{\partial}{\partial b} J(\underbrace{w_1, \cdots, w_n}, b)$
}
	repeat {
	$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$
	$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$
	}

Gradient descent

One feature

repeat {

$$\underbrace{w} = w - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right) x^{(i)}}_{\frac{\partial}{\partial w} J(w,b)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$$

simultaneously update w, b

}

n features ($n \geq 2$)

repeat {

$j=1$

$$\underbrace{w_1} = w_1 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_1^{(i)}}_{\frac{\partial}{\partial w_1} J(\vec{w},b)}$$

$\vdots$

$j=n$

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_n^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

simultaneously update
w_j (for $j = 1, \cdots, n$) and b

}

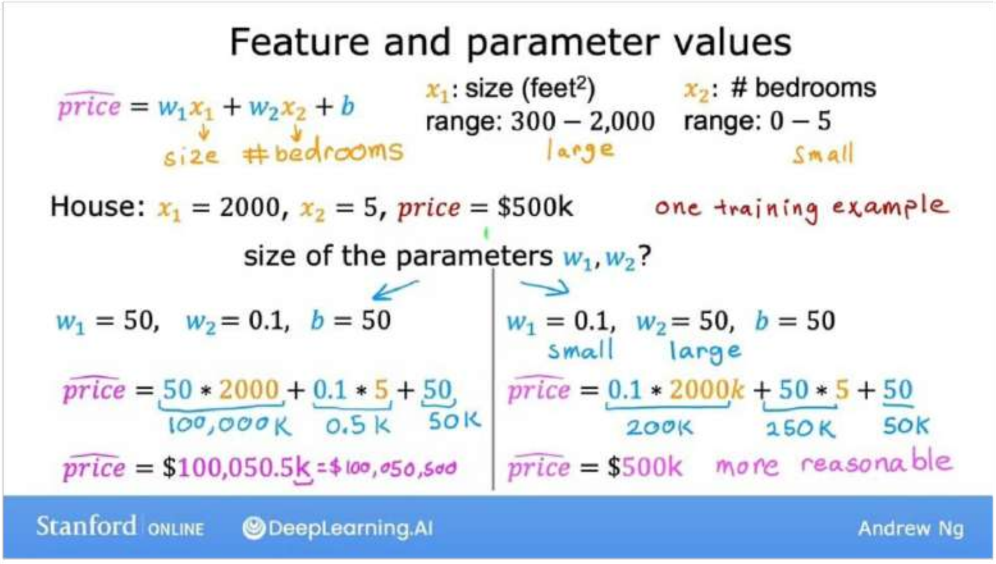
An alternative to gradient descent

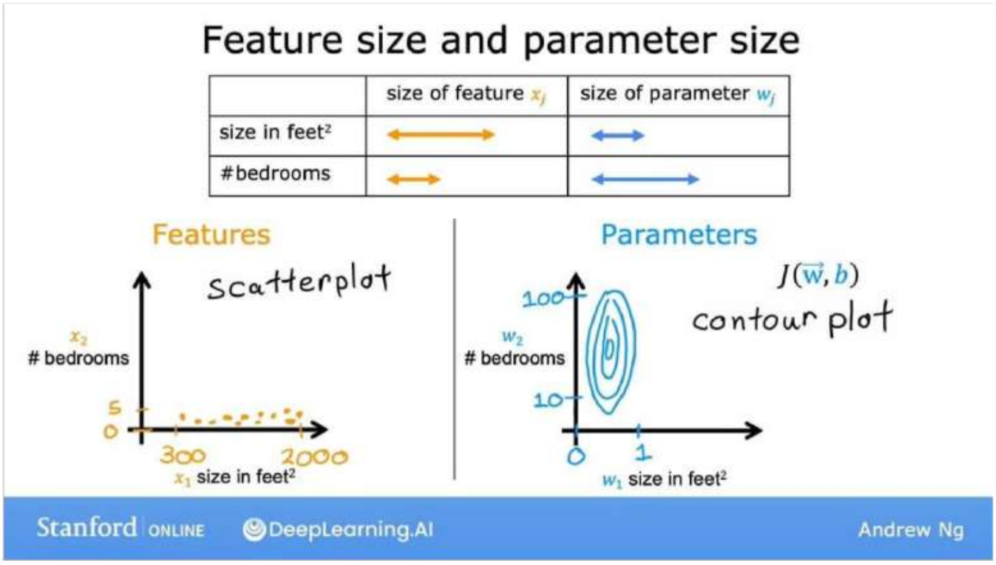
→ Normal equation
- Only for linear regression
- Solve for w, b without iterations

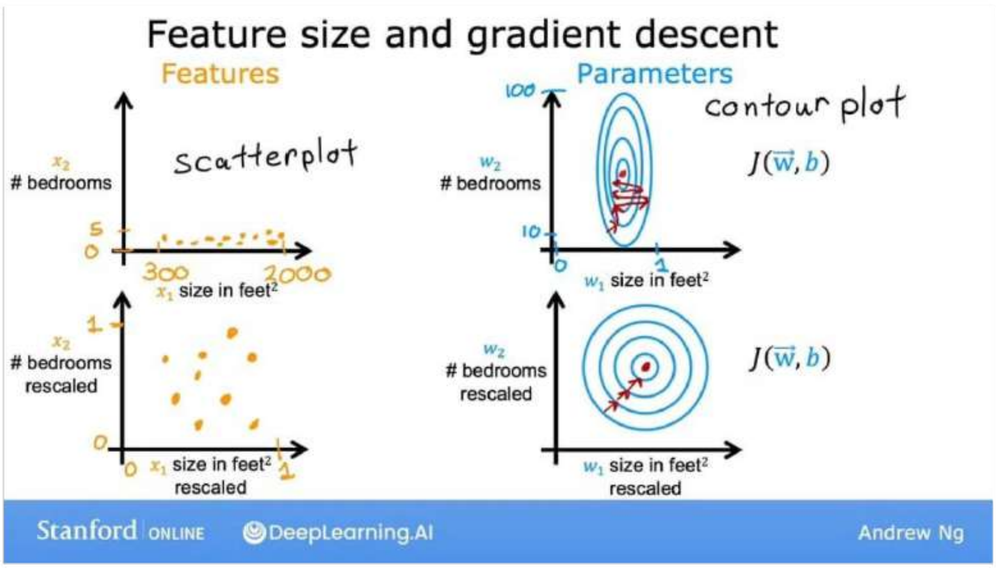
Disadvantages
- Doesn't generalize to other learning algorithms.
- Slow when number of features is large (> 10,000)

What you need to know
- Normal equation method may be used in machine learning libraries that implement linear regression.
- Gradient descent is the recommended method for finding parameters w,b

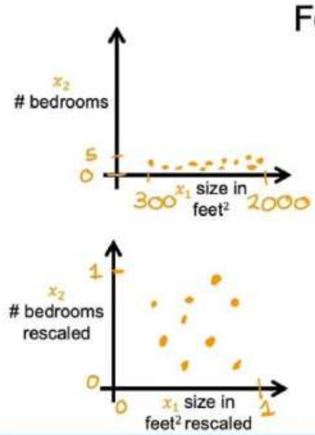
Feature Scaling







Feature scaling

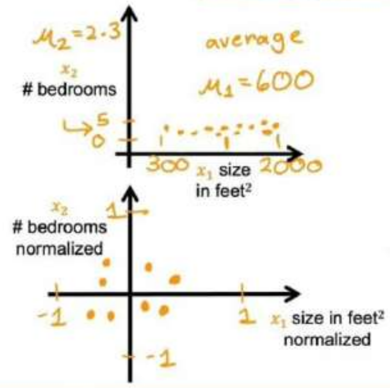


$300 \le x_1 \le 2000$ $\quad$ $0 \le x_2 \le 5$

$x_{1,scaled} = \dfrac{x_1}{\underbrace{2000}_{max}}$ $\quad$ $x_{2,scaled} = \dfrac{x_2}{\underbrace{5}_{max}}$

$0.15 \le x_{1,scaled} \le 1$ $\quad$ $0 \le x_{2,scaled} \le 1$

Mean normalization



$300 \le x_1 \le 2000$ $\quad$ $0 \le x_2 \le 5$

$x_1 = \dfrac{x_1 - \mu_1}{\underbrace{2000-300}_{max-min}}$ $\quad$ $x_2 = \dfrac{x_2 - \mu_2}{\underbrace{5-0}_{max-min}}$

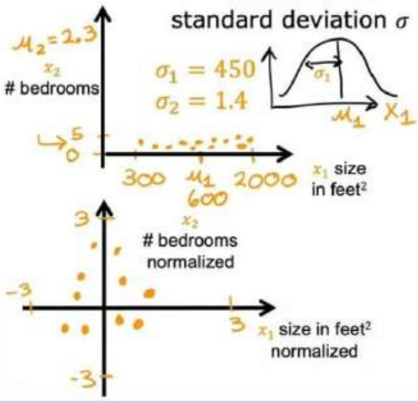
$-0.18 \le x_1 \le 0.82$ $\quad$ $-0.46 \le x_2 \le 0.54$

Z-score normalization

standard deviation σ

$\sigma_1 = 450$

$\sigma_2 = 1.4$



$300 \le x_1 \le 2000$ $\quad$ $0 \le x_2 \le 5$

$x_1 = \dfrac{x_1 - \mu_1}{\sigma_1}$ $\quad$ $x_2 = \dfrac{x_2 - \mu_2}{\sigma_2}$

$-0.67 \le x_1 \le 3.1$ $\quad$ $-1.6 \le x_2 \le 1.9$

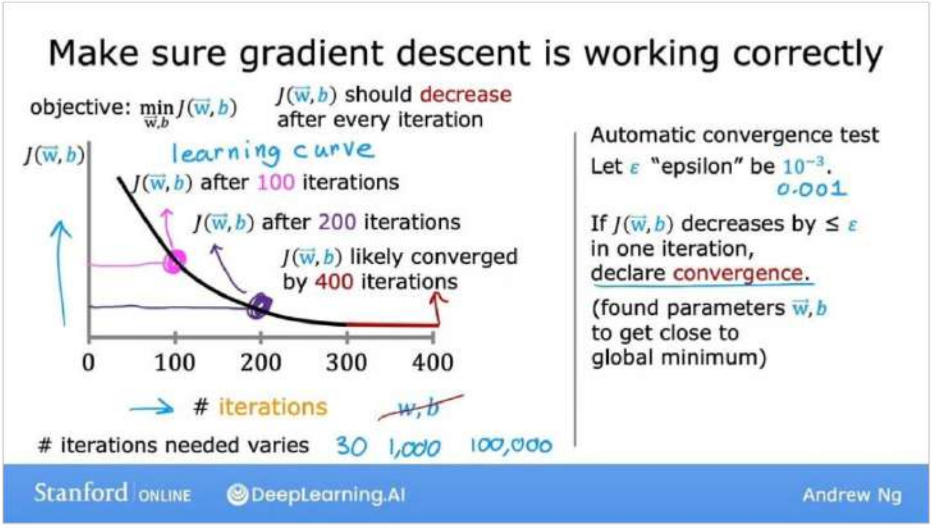
Feature scaling

aim for about $-1 \le x_j \le 1$ for each feature x_j

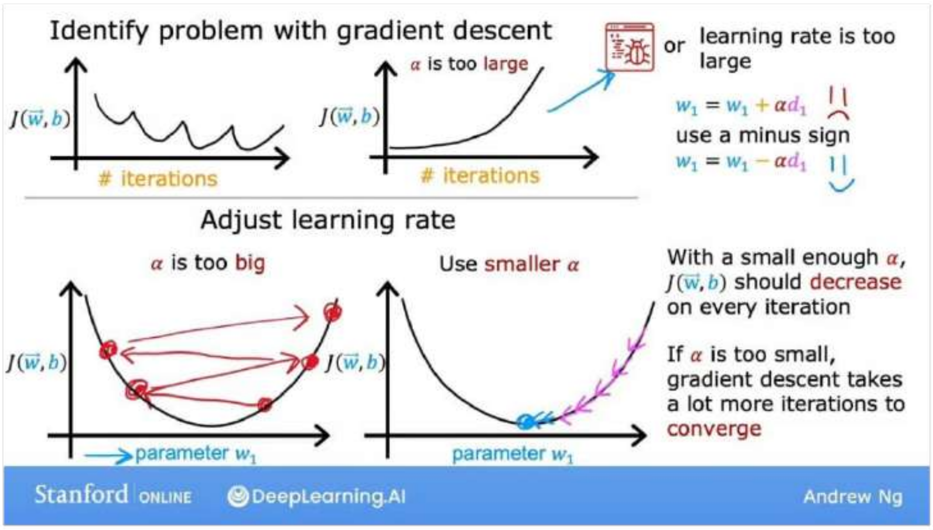
$\left.\begin{array}{c} -3 \le x_j \le 3 \\ -0.3 \le x_j \le 0.3 \end{array}\right\}$ acceptable ranges

$0 \le x_1 \le 3$	okay, no rescaling
$-2 \le x_2 \le 0.5$	okay, no rescaling
$-100 \le x_3 \le 100$	too large → rescale
$-0.001 \le x_4 \le 0.001$	too small → rescale
$98.6 \le x_5 \le 105$	too large → rescale

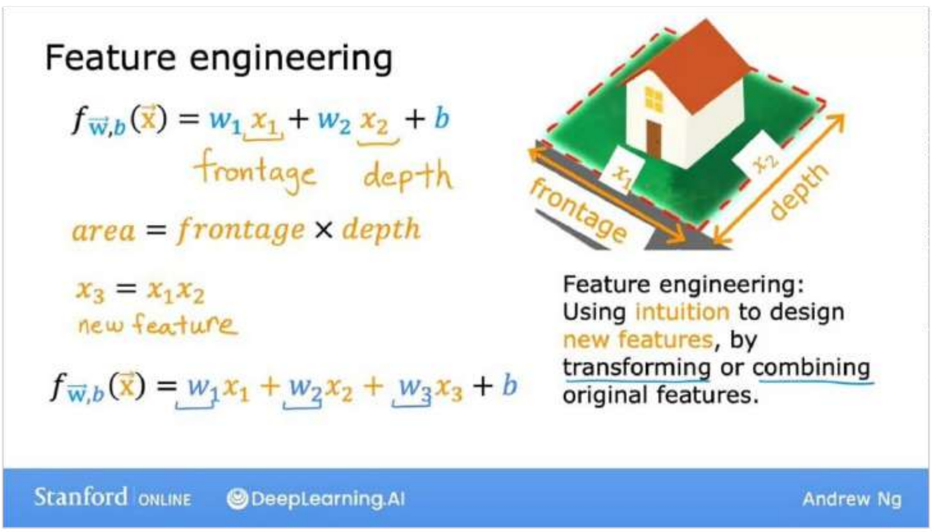
convergence of gradient descent



choosing learning rate



Feature Engineering



Polynomial Regression