Machine Learning

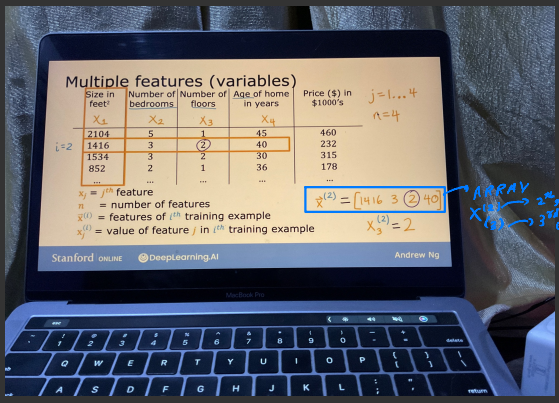
by - Andrew Ng

Sun, 10 Jul

Week 2

Making linear Regression powerful

Multiple features



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

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

eg

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

x_1: size; x_2: bedrooms; x_3: floors; x_4: years; 80: base price

n features:

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

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

b is a no.

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

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

$\cdot$: dot product

$$\vec{A} \cdot \vec{B} = AB\cos\theta$$
$$\vec{A} \cdot \vec{B} = A_x B_x + A_y B_y$$

Vectorization :

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

b is a no.

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

NumPy

```
w = np.array([1.0, 2.5, -3.3])
b = 4
x = np.array([10, 20, 30])
w[0] w[1] w[3]
```

Without vectorization

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

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

Or

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

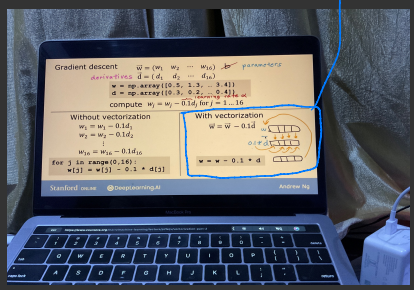
range (n)

→ Parallel processing hardware

With vectorization :

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

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



$w_1, w_2, w_3 \cdots w_n$ $\quad$ $\vec{w} = [w_1 \; w_2 \; w_3 \cdots)$

b $\quad$ b

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

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

cost fun

$J(w_1, w_2, \cdots w_n, b)$ $\quad$ $J(\vec{w}, b)$

Gradient Descent:

$$w_j = w_j - \alpha \frac{\delta}{\delta w_j} J(w_1, w_2 \cdots w_n, b)$$

$$b = b - \alpha \frac{\delta J(w_1, w_2 \cdots w_n, b)}{\delta b}$$

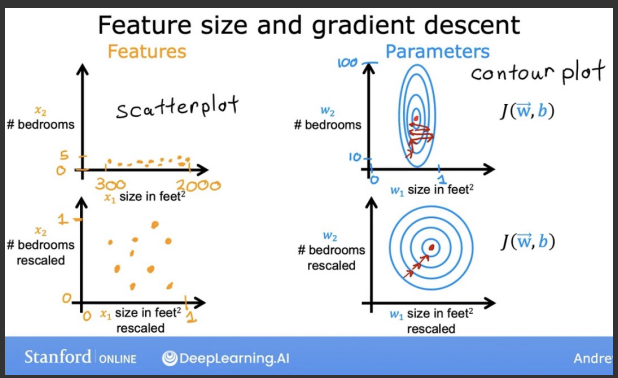
$$w_j = w_j - \alpha \frac{\delta}{\delta w_j} J(\vec{w}, b)$$

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

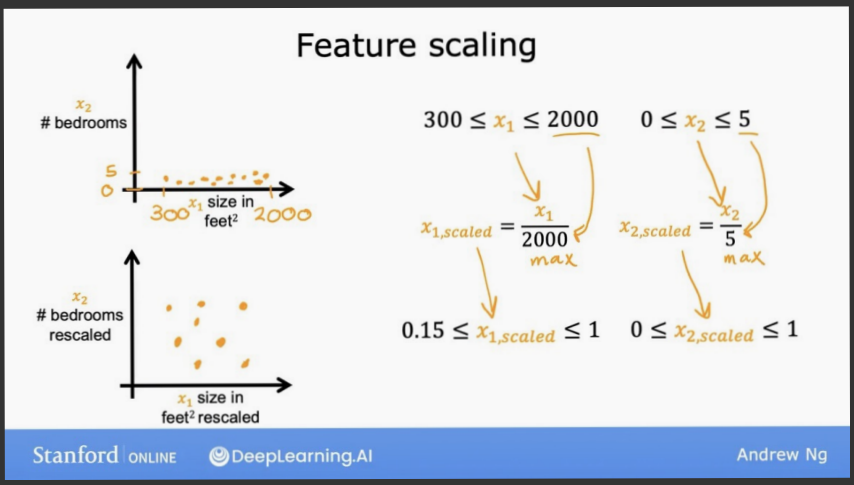
normal eq^n ?

- only for linear eq^n
- solve for w, b without iteration

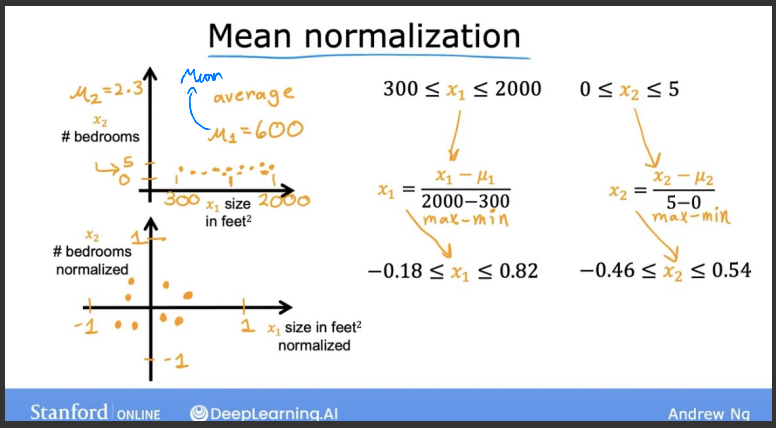
rescaled features => less eclipse more circular



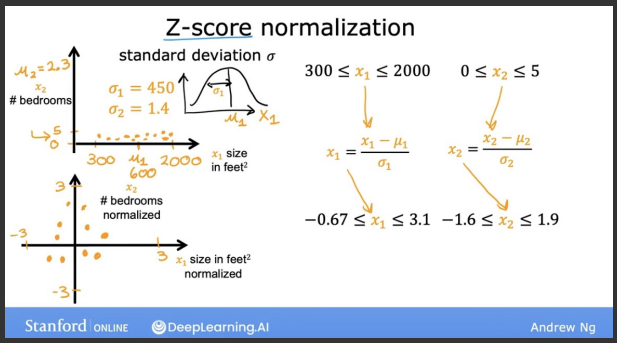
Feature scaling:



Mean normalization:



Z-score normalization



$$\sigma = \sqrt{\frac{\sum (x - \text{mean})^2}{n}}$$

x is a set of numbers

mean is the average of the set of numbers

n is the size of the set

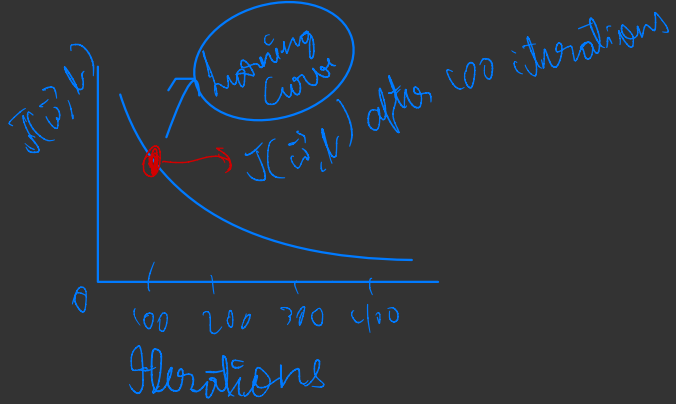
σ is the standard deviation

Feature scaling → makes gradient descent run faster

aim for $-1 \le x_i \le 1$

Acceptable $-3 \le x_i \le 3$

$-0.3 \le x_i \le 0.3$



J shouldn't increase after an iteration.

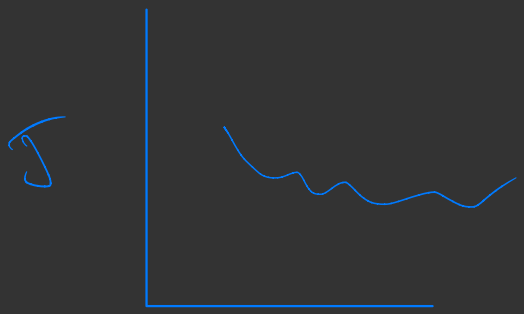
So, we can use learning curve to know when our model is trained OR

we use

Automatic convergence test i.e

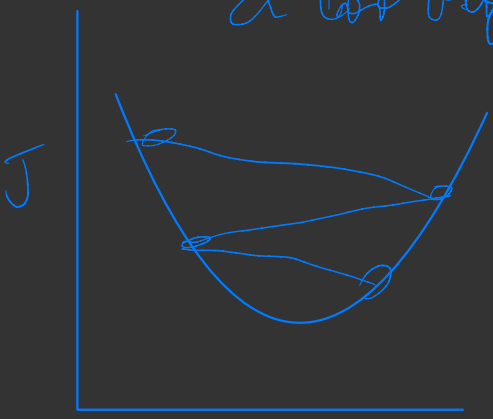
Let ε be 10^{-3} or 0.001

If $J(\vec{w},b)$ dec. by $\leq \varepsilon$ in one iteration, declare <u>convergence</u>.

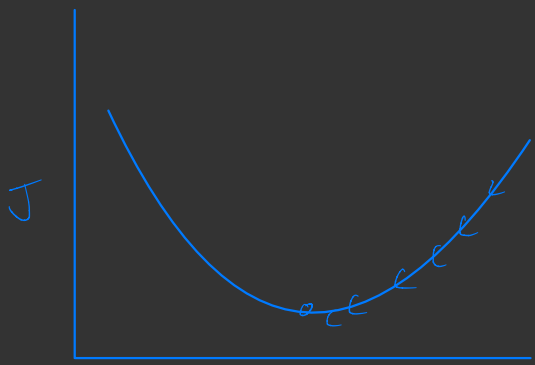


$\Rightarrow$ Bug: α is too large.

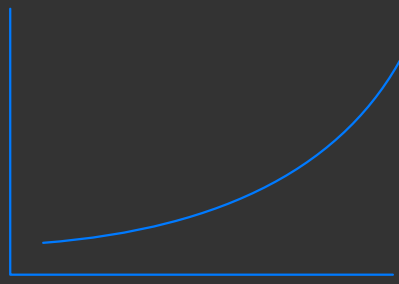
α too big



use smaller α



α is too large (or a bug)



$w_1 = w_1 \oplus \alpha\, d_1$

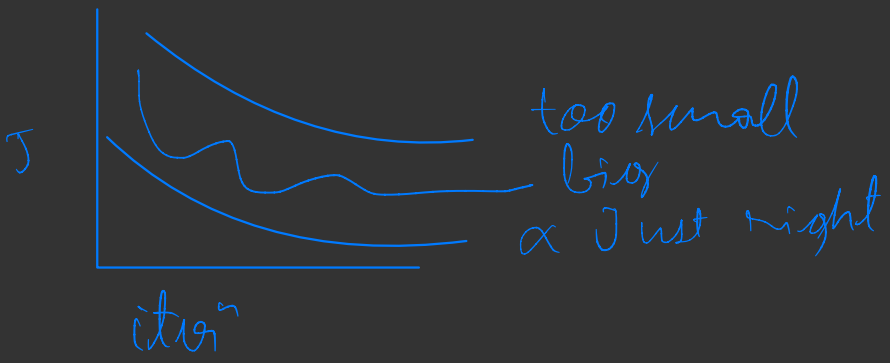
$\vdots$

$w_1 = w_1 - \alpha\, d_1$

$\vdots$

values of α to try:

... 0.001 0.003 0.01 0.1 1 ...



Feature engineering:

Feature engineering

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

frontage depth

$area = frontage \times depth$

$x_3 = x_1 x_2$

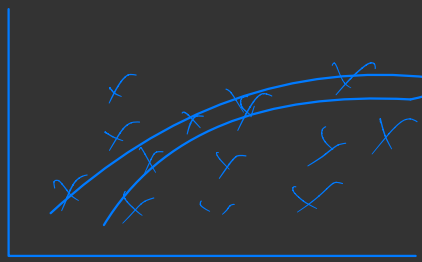
new feature

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

Feature engineering:
Using intuition to design new features, by transforming or combining original features.

Stanford ONLINE DeepLearning.AI Andrew Ng

Polynomial regression:



$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$

size, size^2

feature scaling becomes more important

We can have $\sqrt{x}$ too

Scikit-learn *