Supervised ML Dr Andrew Ng

Week 1

Applications of ML

Machine Learning → Field of Study that gives computers the ability to learn without being explicitly programmed

Arthur Samuel (1959)

Machine Learning Algorithms

Supervised Learning — used most in real-world Applications

unsupervised learning

Supervised Learning

$X \rightarrow Y$

input — output Label

→ Learns from being given "right Answers"

Example

Input (X)		output (Y)	Applications
email	→	spam (0/1)	spam filtering
audio	→	text transcripts	speech recognition
English	→	Spanish	Machine translation
ad, user info	→	click? (0/1)	online Advertising
image, radar info	→	Position of other cars	Self-driving car
image of Phone	→	defect? (0/1) →	visual inspection

Regression: Housing price Prediction

Q) Choose Most appropriate line to fit the data

Regression: Predict a number infinitely many possible outputs

Supervised learning Classification

Example Breast Cancer detection

Size	diagnosis
2	0
5	1
1	0
7	1

⊗ We can use two or more inputs

To Recap

supervised learning: Learns from being given "right Answers"

Regression → predict a number infinitely many possible output

Classification → predict Categories small number of possible output

unsupervised learning

supervised Learning: Learn from data labeled with the "right Answers"

unsupervised Learning: find something interesting in unlabeled data

example: Clustering → we use it in Google news
→ DNA microarray
→ Grouping Customers

unsupervised Learning: Data only Comes with input x but not o/p label y.

Algorithm has to find structure in the data

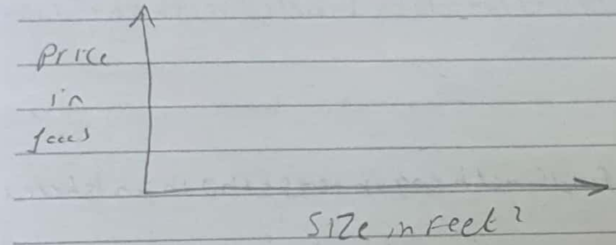
1) Clustering: Group similar data points together

2) Anomaly detection: Find unusual data points

3) Diemensionality reduction: Compress data using fewer numbers

Regression model

Linear regression



Regression model predicts numbers
Classification model predicts categories

Terminology
Training set: Data used to train the model

Size in feet²	Price in $ 1000's
2104	400
1416	232
1534	315
852	172
...	...
3210	870

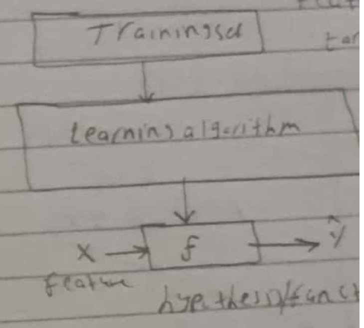
Notation
x → "input" variable "feature"
y → "output" variable "target variable"

m → number of training ex

(x, y) → Single train

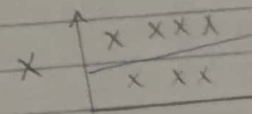
$(x^{(i)}, y^{(i)})$ → i-th t

feat
tar



Q | How to represen

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



Linear regr
equivalent

Notation

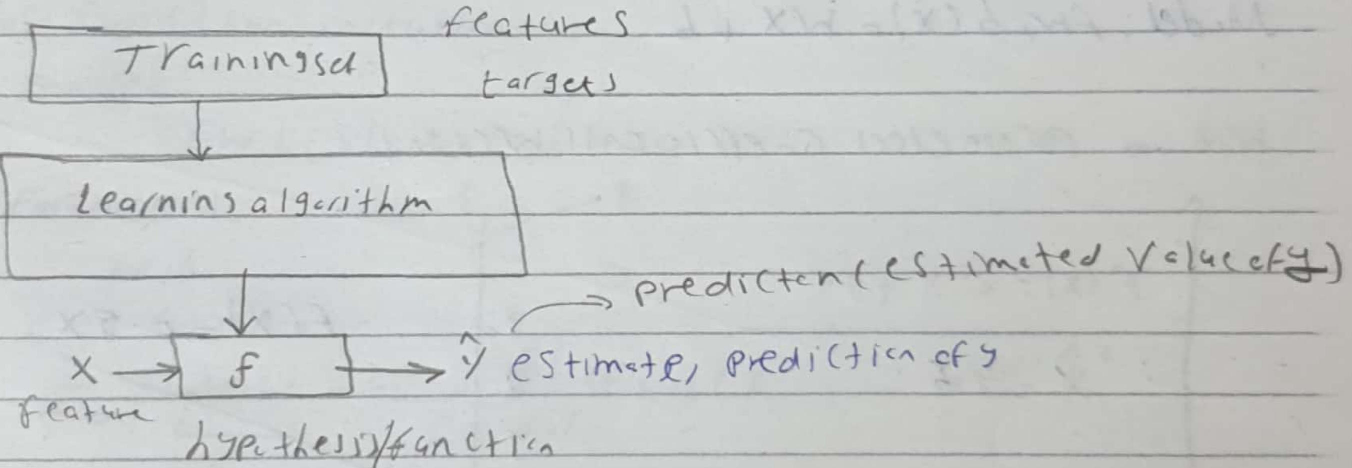
x → "input" variable "feature"

y → "output" variable
"target variable

m → number of training examples

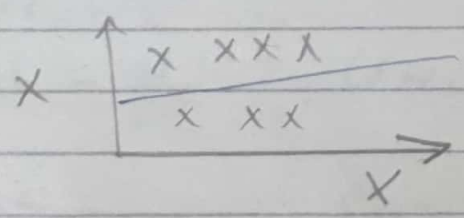
(x, y) → single training example

$(x^{(i)}, y^{(i)})$ → i^{th} training example



Q | How to represent f

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



single feature x

Linear regression with one variable

equivalent to univariate linear regression

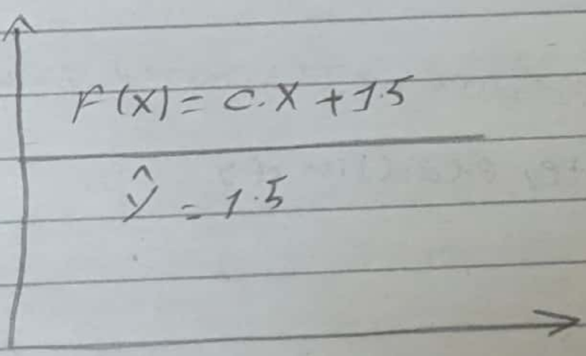
Cost function

Training Set

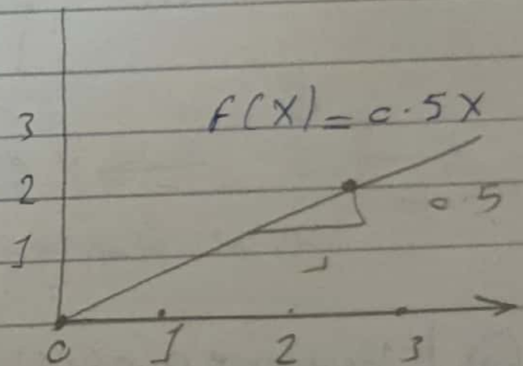
features size in feet^2 (x)	target price \$1000's (y)
2104	460
1416	232
1534	315
852	178

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

w, b → parameters coefficients weight

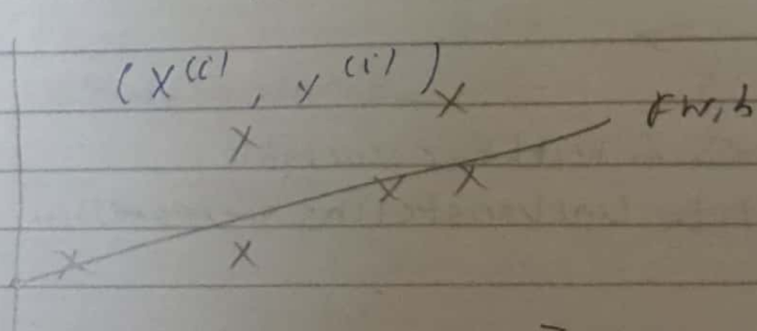


w = 0
b = 1.5
y intercept



w = 0.5
b = 0

f(x) = 0.5x + 1



$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$

$f_{w,b}(x^{(i)}) = wx^{(i)} + b$

Q) Find w, b: $\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$

Cost function: squared error cost function

$$\underbrace{\frac{1}{2m}\sum_{i=1}^{m}}\left(\underbrace{\hat{y}^{(i)} - y^{(i)}}_{\text{error}}\right)^2 \Rightarrow m \Rightarrow \text{number of training example}$$

→ to make the calculation neater

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

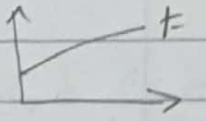
Cost function intuition

model:

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

Parameters:

w, b



Cost function:

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

goal:

Minimize $J(w,b)$

Simplified:

$f_w(x) = wx$

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)^2$$

goal:

minimize $J(w,b)$

Simplified

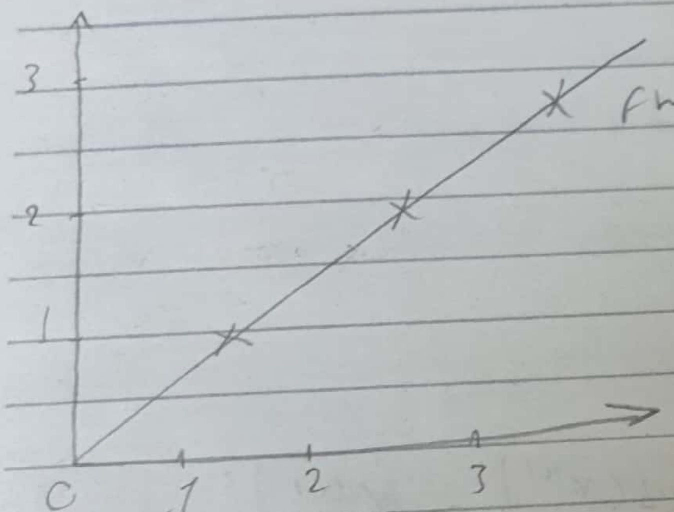
$f_w(x) = wx$

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_w(x^{(i)}) - y^{(i)} \right)^2$$

minimize $J(w)$

$f_w(x)$
(for fixed w, function of x)

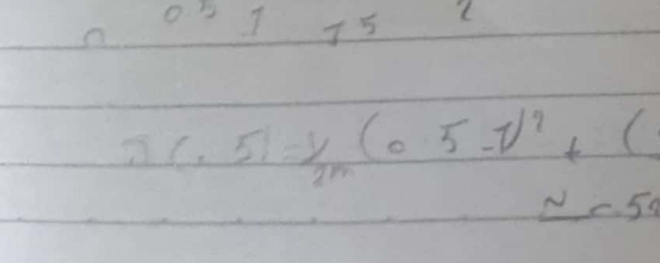
$J(w)$
(function of w)



$w = 1$

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_w(x^{(i)}) - y^{(i)} \right)^2 = \frac{1}{2m} \sum_{i=1}^{m} \left(w x^{(i)} - x^{(i)} \right)^2$$

$J(w)$ (function of w)



$$J(0.5) = \frac{1}{2m} \left((0.5-1)^2 + (1-2)^2 + (1.5-3)^2 \right) = \frac{3.5}{6}$$

$\approx .58$

goal of linear regression: minimize J(w)

general case:
minimize J(w,b)

visualizing cost function

gradient descent ⟹ we can use it to train deep learning model!

Have some function J(w,b) for

want $\min_{w,b} J(w,b)$

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

implementing gradient descent

learning rate

assignment

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

old value

cost function

Assignment	Truth assertion
a = c	a = c
	a = a + 1
Code	math
	a == c

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

Repeat until convergence

Simultaneously
update w and b

Correct simultaneous update

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

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

$w = tmp\_w$

$b = tmp\_b$

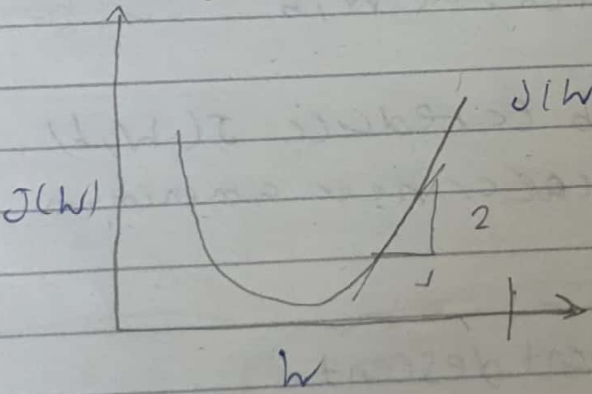
Gradient descent Algorithm

repeat until convergence {

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

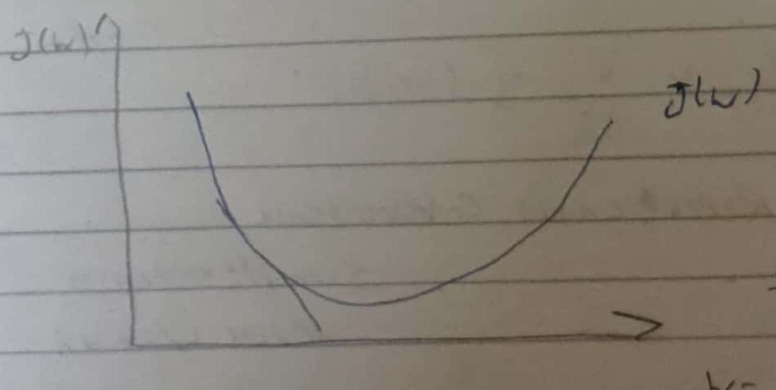
Learning rate

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



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

$w = w - \alpha \cdot$ (positive number)



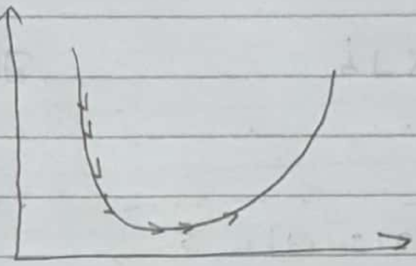
$\frac{d}{dw} J(w)$

$w = w - \alpha \cdot$ (negative number)

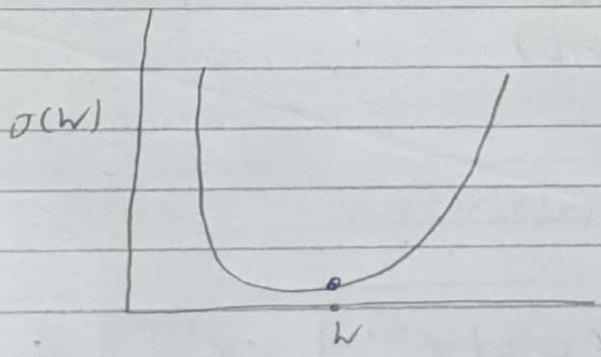
Learningrate

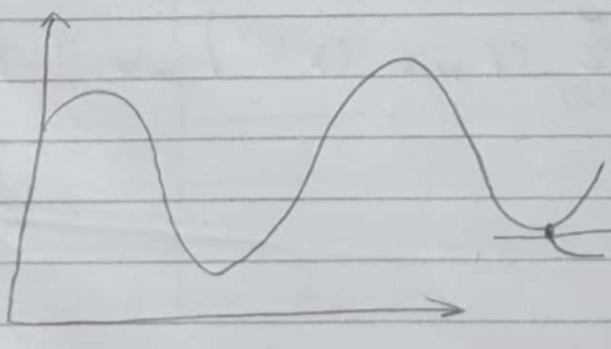
$$w = w - \alpha \frac{\partial}{\partial w} J(w)$$

if α is too small → very small baby step
→ Gradient descent maybe slow



if α is too large





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

Can reach local minimum with fixed learning rate

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

Gradient descent for linear regression

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

Gradient descent Algorithm {
repeat until convergence {

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

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

}

after applying derivative
repeat until convergence {

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

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

PAGE
DATE

week2

Multiple features (variables)

Single feature

Size in feet^2 (x)	Price (\$) in 1000's (y)
2104	400
1416	232
1534	315
852	178

Multiple features

Size in feet^2	Number of bedrooms	Number of floors	Age of home in years	Price (\$) in 1000's
2104	5	1	45	
		2	40	
1416	3			
1534	3	2	30	
852	2	1	36	

x_j = jth feature

n = number of features

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

Model

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

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

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

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

Parameters and features

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

b is a number

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

linear algebra count from 1

w = np.array([1.0, 2.5, -3.3])

b = 4

x = np.array([10, 20, 30])

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

without vectorization

vectorization

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

np.dot(w, x)

Gradient descent $\vec{w} = (w_1, w_2, \ldots, w_{16})$ b parameters

$\vec{d} = (d_1, d_2, \ldots d_{16})$

w = np.array([0.5, 1.3, ..., 3.4])

d = np.array([0.3, 0.2, ..., 0.4])

$w_j = w_j - 0.1 d_j$ for $j = 1, \ldots 16$

with vectorization

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

w = w - 0.1 * d

np.random.random_sample(4)   random vector لعمل

a = np.arange(4.)

Gradient descent

repeat {

$$WJ = WJ - \alpha \frac{\partial}{\partial WJ} J(W1, \dots, Wn, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(W1, \dots, Wn, b)$$

$$WJ = WJ - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_J^{(i)}}_{\frac{\partial}{\partial W_J} J(\vec{W}, b)}$$

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

Gradient descent practice

$$\text{Price} = \underset{\text{size}}{\underset{\downarrow}{W_1 X_1}} + \underset{\#\text{bedrooms}}{\underset{\downarrow}{W_2 X_2}} + b$$

X_1: size (feet^2)
range: 300 – 2,000

X_2: # bedrooms
range: 0:5

House: $X_1 = 2000$, $X_2 = 5$, Price $= 500$ k — one training example

size of the Parameters W_1, W_2

$W_1 = 50$, $W_2 = 0.1$, $b = 50$

$$\overline{\text{Price}} = \underbrace{50 * 2000}_{100{,}000\text{K}} + \underbrace{0.1 * 5}_{0.5\text{K}} + \underbrace{50}_{50\text{K}}$$

Price = \$ 100,050.5 K = \$ 100,050,500

$W_1 = 0.1$ (small) $\quad W_2 = 50$ (large) $\quad b = 50$

$$\text{Price} = \underbrace{0.1 * 2000\text{K}}_{200\text{K}} + \underbrace{50 * 5}_{250\text{K}} + \underbrace{50}_{50\text{K}} = 500\ \$\ \text{K}$$

Reasonable

في بعض الأحيان بتكون مضطر تعمل features ال scaling

عشان ده بيسهل عليا إني أوصل لل ~~local~~ global minimum بسرعة
بدل ما يحصل bouncing

هيكون ال contour plot شكله circles بدل ellipse

How do we Actually scale features

$300 \leq X_1 \leq 2000$ $\qquad$ $0 \leq X_2 \leq 5$

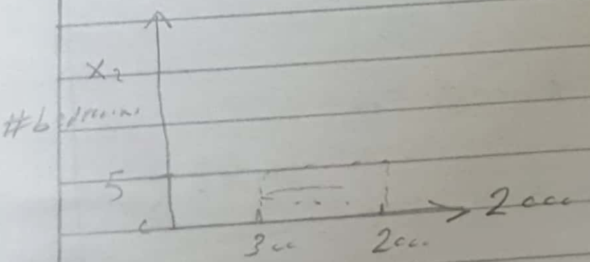
$X_{1,\text{scaled}} = \frac{X_1}{2000}$ $\qquad$ $X_{2,\text{scaled}} = \frac{X_2}{5}$

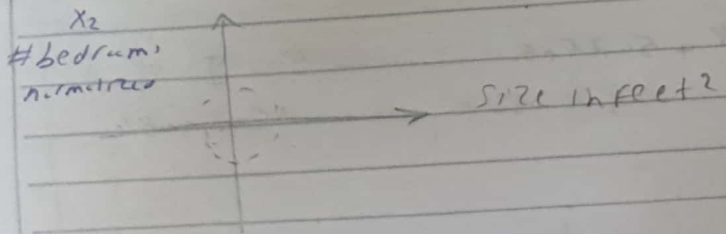
$0.15 \leq X_{1,\text{scaled}} \leq 1$ $\qquad$ $0 \leq X_{2,\text{scaled}} \leq 1$

ممكن بدل ما نقسم divide على Maximum ممكن نعمل

Mean normalization

$300 \leq X_1 \leq 2000$ $\qquad$ $0 \leq X_2 \leq 5$





Using Mean normalization

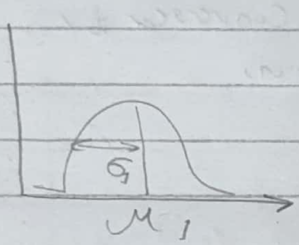
$300 \leq x_1 \leq 2000$

let $\mu_1 = 600$

$$x_1 = \frac{300 - 600}{2000 - 300} \leq x_1 \leq \frac{2000 - 600}{2000 - 300}$$

$$-0.18 \leq x_1 \leq 0.82$$

ن حاجة (اسمها) ZScore normalization هنا بنحسب Standard deviation



$$x_1 = \frac{x_1 - \mu_1}{\sigma_1} \qquad \text{let } \sigma = 450,\ \mu_1 = 600$$

$$\boxed{\underbrace{\frac{300 - 600}{450}}_{-0.67}} \leq x \leq 3.1$$

نفس الكلام بالنسبة لل second feature

→ Note for feature scaling

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

$-3 \leq x_j \leq 3$ Acceptable ranges

$-0.3 \leq x_j \leq 0.3$

$0 \leq x_1 \leq 3$ okay, no rescaling

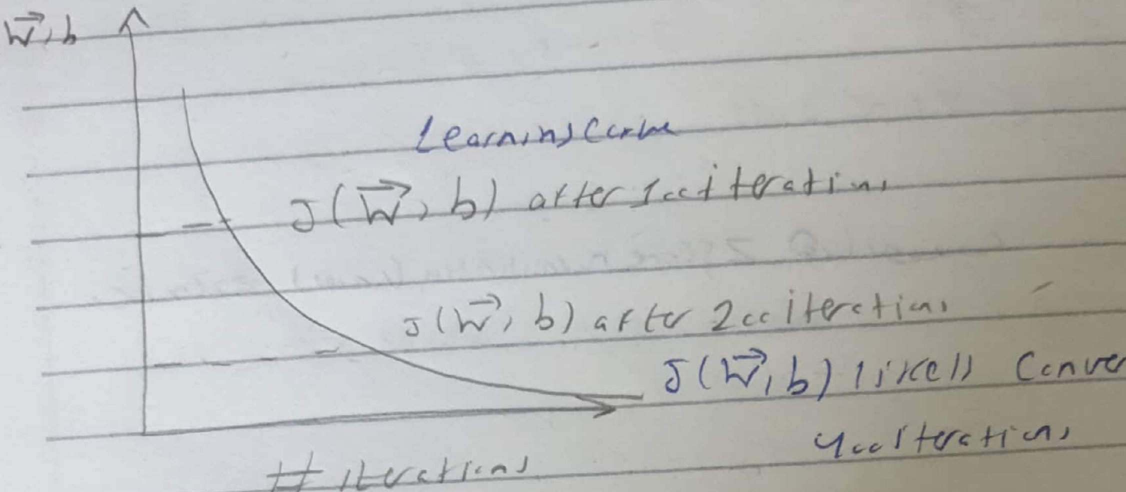
$-2 \leq x_2 \leq 0.5$ okay, no rescaling

$-100 \leq x_3 \leq 100$ too large → rescale

$-0.001 < x_4 < 0.001$ → too small → rescale

Checking gradient descent for convergence

objective $\min_{\vec{w},b} J(\vec{w},b)$

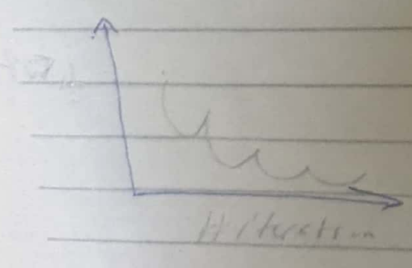


Automatic Convergence test let ε epsilon be 10^{-3}
0.001

if $J(\vec{w},b)$ decrease by $\le \varepsilon$ in one iteration declare

Convergence

Choosing good learning rate



هنا ف حاجة غلط يا
large learning rate أو bug en code

Notice $w_1 = w_1 + \alpha d_1$

$w_1 = w_1 - \alpha d_1$

derivative term el Minus

feature engineering

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

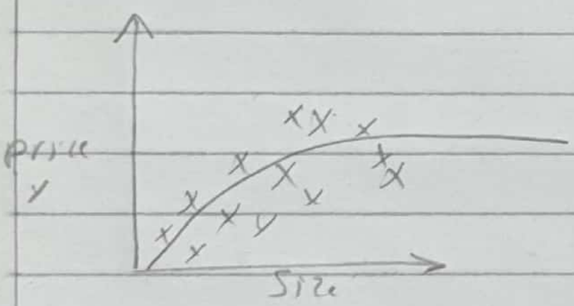
frontage ↓ depth

area = frontage x depth

$x_3 = x_1x_2$ → new feature

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

Polynomial regression



$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + b$$

size ↑ 1-1e3

size² 1-1e6

size³ ↑ 1-1e9

We have choice of each feature we want to build most suitable model