Module 3: Regression with multiple variables

Linear Regression with multiple variables

Multiple Features

Content

Multiple Linear Regression

- Multiple Features
- Vectorization
- Optional Lab: Python, NumPy and Vectorization
- Gradient Descent for Multiple Regression
- Optional Lab: Multiple Linear Regression
- Practical Quiz: Multiple Linear Regression

Gradient Descent in practice

- Feature Scaling
- Checking Gradient Descent for Convergence
- Choosing the Learning Rate
- Optional Lab: Feature Scaling and Learning rate
- Feature Engineering
- Polynomial Regression
- Optional Lab: Feature engineering and Polynomial Regression
- Optional Lab: Linear regression with Scikit-learn

What is this module is all about ...

- In this Module, you'll extend linear regression from single input to multiple input features.
- You'll also learn some methods for improving your model's training and performance, such as vectorization, feature scaling, feature engineering and polynomial regression.
- At the end of the module, you'll get to practice implementing linear regression using python.

Multiple Features (variables)

Multiple features (variables) is an extension of single feature regression

one feature →

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

← Output (Label)

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

Uptill now we have discussed a regression model with only single input variabe, in this module we will introduce a regression model with multiple input variables

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
	…	…	…	…	…

$j = 1 \ldots 4$

$n = 4$

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

x_j = j^{th} feature j = 1 , ..., 4, n = 4

n = number of features

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

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

$x_3^{(2)} = 2$ this is 3rd value of the 2nd training example, of the 3rd feature X3

Model with Multiple Features

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

General Model

$$f_{w,b}(X) = w_1 X_1 + w_2 X_2 + w_3 X_3 + w_4 X_4 + b$$

Example

$$f_{w,b}(X) = 0.1 X_1 + 4 X_2 + 10 X_3 + -2 X_4 + 80$$

X_1: size; X_2: #bedrooms; X_3: #floors; X_4: years; 80: base price

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

How to interprete coefficients

$$Y=80+w_1 \cdot X1 + w_2 \cdot X_2 + w_3 \cdot X_3 + w_4 \cdot X_4$$

Estimated Price=80 + w_1·Size + w_2·Bedrooms + w_3·Floors + w4·Age

Estimated Price=80 + 0.1·Size + 4·Bedrooms + 10·Floors + -2·Age

- Size: The unit for the coefficient 0.1 would be $100 (dollars) per square foot.
- Number of Bedrooms: The unit for the coefficient 4 would be 4x1000 (dollars) per additional bedroom.
- Number of Floors: The unit for the coefficient 10 would be 10X $1000 (dollars) per additional floor.
- House Age : The unit for the coefficient −2 would be -2x$1000 (dollars) per additional year of age.

Question:

In the training set below, what is $x_1^{(4)}$, $x_3^{(3)}$, $x_5^{(1)}$, $x_2^{(2)}$

Size in feet² X_1	Number of bedrooms X_2	Number of floors X_3	Age of home in years X_4	Price ($) in $1000's
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

Multiple linear regression model expression (Vector Notation)

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

$$\vec{w} = [w_1 \; w_2 \; w_3 \; \cdots \; w_n] \text{ (Vector)}$$

b is a number

parameters of the model

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

So the model can be written as

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

dot product

Multiple Linear Regression

Vectorization Part 1

What is Vectorization

- Vectorization in multivariate regression involves using mathematical operations on entire sets of data at once, taking advantage of linear algebra principles. Instead of individually processing each data point and feature in loops.
- Vectorization allows for efficient computations by operating on matrices as a whole.
- This approach improves computational speed and is particularly useful for handling large datasets in machine learning, making the code more concise and leveraging optimized, hardware-accelerated operations on matrices.

Benefits of vectorization

Benefit	Description
Computational Efficiency	Vectorization significantly reduces computation time by processing multiple data points simultaneously.
Scalability	Vectorized algorithms can handle large datasets effectively, making them scalable to real-world applications.
Simplicity	Vectorized code is often more concise and easier to understand compared to iterative code.

Parameters and features with vectorization

for n = 3 $\vec{w} = [w_1 \quad w_2 \quad w_3]$ $\vec{x} = [x_1 \quad x_2 \quad x_3 \quad]$ b is a number

linear algebra: count from 1

```
                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

NumPy Python Linear algebra library

Without vectorization vs. with Vectorization

no vectorization Direct multiplication

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

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

np library uses parallel processing

No vectorization For loop

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

range (0,n) = 0, 1, 2, 3, … n-1

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

with Vectorization

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

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

Question

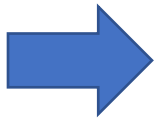
Which of the following is a vectorized implementation for computing a linear regression model's prediction?

- `f = np.dot(w,x) + b`
- ```
  f=0
    For j in range(n):
     f = f + w[j] * x[j]
    f=f + b
  ```
  ```

To conclude ...

Vectorization is a critical technique that makes your code simpler, easier to write and to read and it runs much faster

But what is the magic behind vectorization that makes it run so much faster ???
lets see in the next section what computers do behind the seens to make computers run much faster



Implementing Vectorization

The secret that makes vectorization make computers run faster

Without vectorization for loop

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

t_0 (operates on the 1st term

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

t_1

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

…

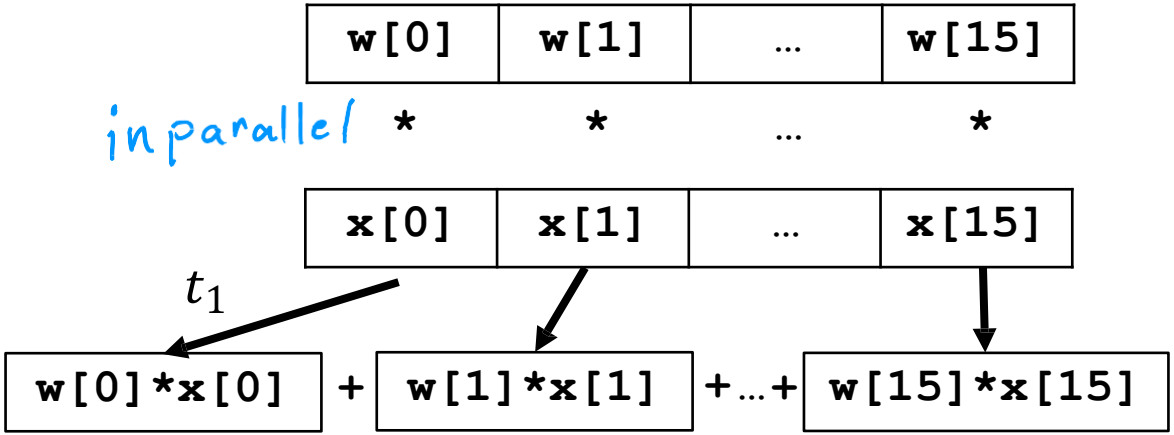
t_{15}

```
f + w[15] * x[15]
```

Vectorization

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

t_0 (process done at one time



Vectorization: efficient calculation and scale to large data set

Gradient descent with vectorization

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

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

Store values in arrays

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

coefficients

derivatives

compute (update) $w_j = w_j - 0.1 d_j$ for $j = 1 \ldots 16$

learning rate

update

Without vectorization

$w_1 = w_1 - 0.1 d_1$

$w_2 = w_2 - 0.1 d_2$

$\vdots$

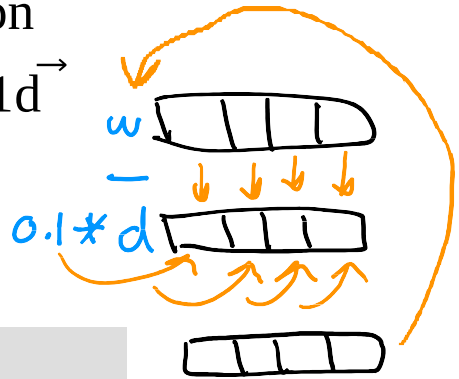
$w_{16} = w_{16} - 0.1 d_{16}$

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

With vectorization

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

do the calculation at the same time




```
w = w – 0.1 * d
```

Question

Which of the following is a vectorized implementation for computing a linear regression model's prediction?

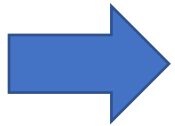
```
f = np.dot(w,x) + b
```

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

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

To conclude ...

- You've learned one of the most important and useful techniques in implementing machine learning algorithms.
- In the next section, we'll put the math of multiple linear regression together with vectorization, so that you will influence gradient descent for multiple linear regression with vectorization.
- Let's go on to the next section.



Gradient Descent for Multiple Regression With Vectorization

You've learned about gradient descents about multiple linear regression and also vectorization.
Let's put it all together to implement gradient descent for multiple linear regression with vectorization.

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

Gradient descent

repeat {

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

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

}

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 {

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

$$\rightarrow \frac{\partial}{\partial w} J(w, b)$$

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

simultaneously update w, b

}

n features ($n \geq 2$)

repeat {

j = 1

$$w_1 = w_1 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m} f_{w,b}\left(x^{(i)}\right) - y^{(i)}\,x^{(i)}}$$

$$\rightarrow \frac{\partial}{\partial w_1} J(\boldsymbol{w}, b)$$

.

.

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

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

}

simultaneous update

w_j for j = 1 ... 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

To conclude ...

- You now know multiple linear regression.
- This is probably the single most widely used learning algorithm in the world today.
- But there's more. With just a few tricks such as picking and scaling features appropriately and also choosing the learning rate alpha appropriately, you'd really make this work much better.
- Let's go on to the next section to see those little tricks that will help you make multiple linear regression work much better.

Practical Tips for Linear Regression

Feature Scaling

Simple definiton of feature scaling

- Imagine you have a dataset with two features: age and income. The age feature might range from 10 to 100, while the income feature might range from $10,000 to $1,000,000. When you train a machine learning model on this data, the model will give more weight to the income feature because it has a larger scale. This can make it difficult for the model to learn the relationship between the age feature and the target variable.
- Feature scaling is a technique that can be used to normalize the features in a dataset so that they all have a similar scale. This can help to improve the performance of machine learning models.

Feature and parameter values

$price = w_1 x_1 + w_2 x_2 + b$

size #bedrooms

x_1: size (feet^2)

range: $300 - 2{,}000$

x_2: # bedrooms

range: $0 - 5$

small

House: $x_1 = 2000$, $x_2 = 5$, $price = \$500\text{k}$ one training example

size of the parameters w_1, w_2?

$w_1 = 50$, $w_2 = 0.1$, $b = 50$

w_1 large, w_2 small

$price = 50 * 2000 + 0.1 * 5 + 50$

100,000 k 0.5 k 50 k

$price = \$100{,}050.5\text{k}$

$w_1 = 0.1$, $w_2 = 50$, $b = 50$

w_1 small, w_2 large

$price = 0.1 * 2000k + 50 * 5 + 50$

200 k 250 k 50 k

$price = \$500\text{k}$

More Reasonable

What do we conclude

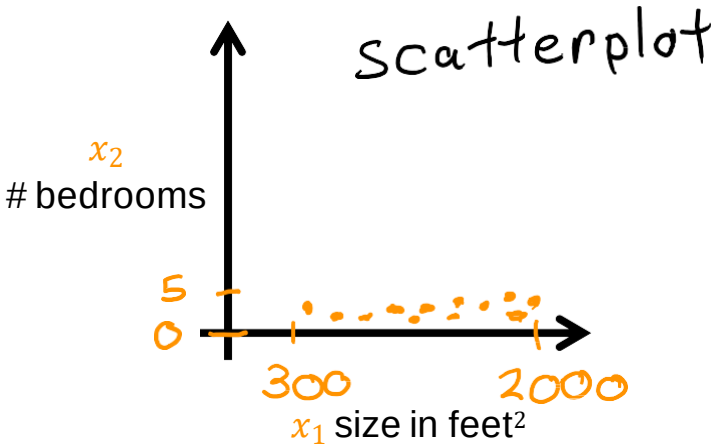
- So hopefully you might notice that when a possible range of values of a feature is large, like the size and square feet which goes all the way up to 2000. It's more likely that a good model will learn to choose a relatively small parameter value, like 0.1.
- Likewise, when the possible values of the feature are small, like the number of bedrooms, then a reasonable value for its parameters will be relatively large like 50.

So how is that relate to gradient Descent ...?

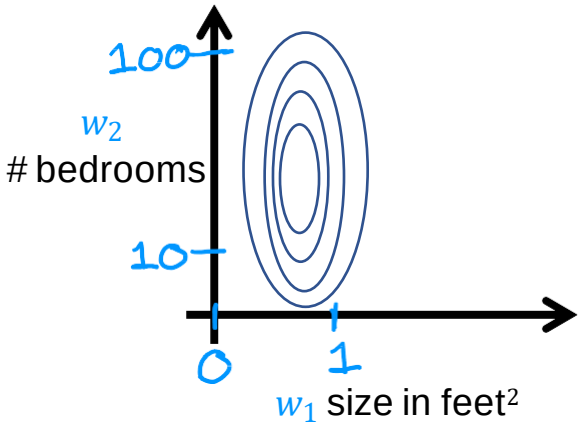
Feature size and parameter size

	size of feature x_j	size of parameter w_j
size in feet^2	⟷	⟷
#bedrooms	⟷	⟷

Features



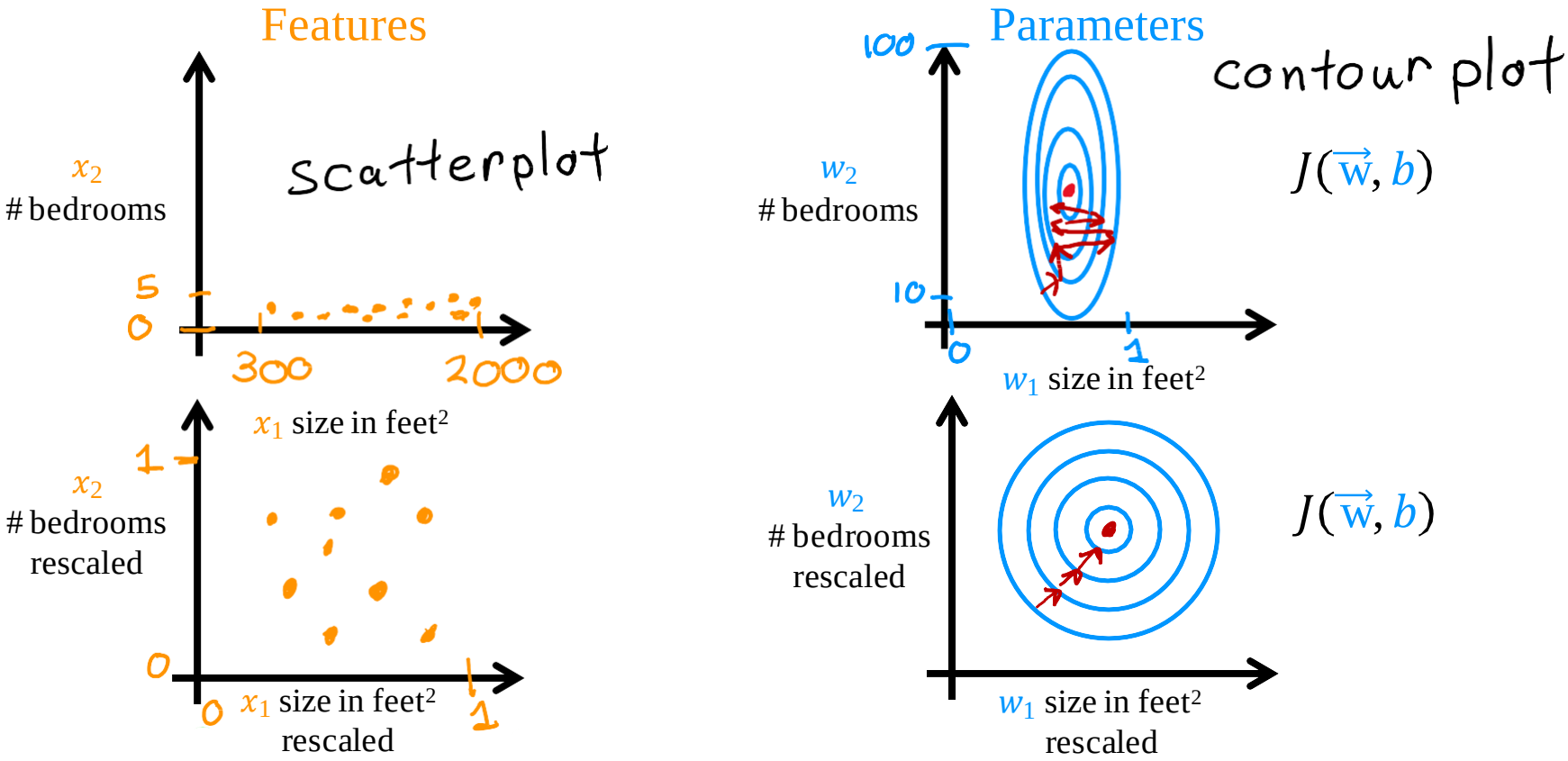
Parameters



$J(w, b)$

Contour plot

Feature size and gradient descent

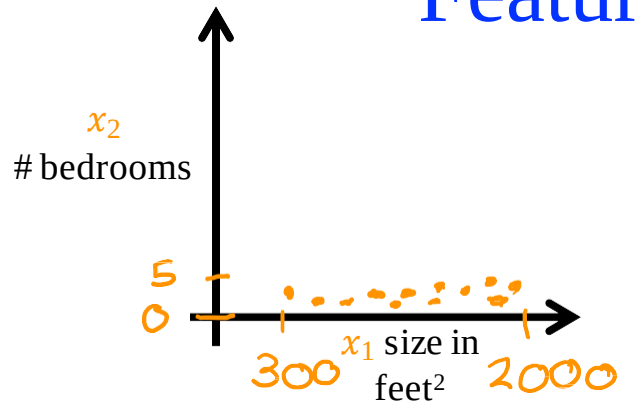


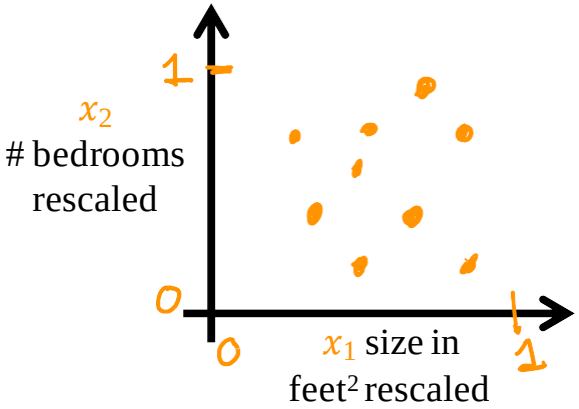
The rescaled values of the parameters are taking comparable ranges to each others

Practical Tips for Linear Regression

How to implement Feature scaling

Feature scaling





$$300 \le x_1 \le 2000$$

$$x_{1,scaled} = \frac{x_1}{2000} \quad \text{max}$$

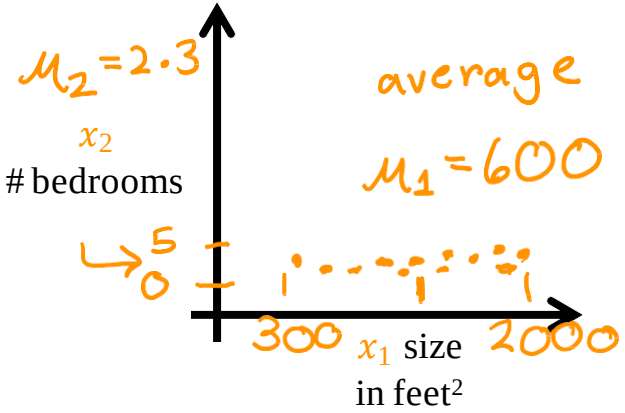
$$0.15 \le x_{1,scaled} \le 1$$

$$0 \le x_2 \le 5$$

$$x_{2,scale} = \frac{x_2}{5} \quad \text{max}$$

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

Mean normalization

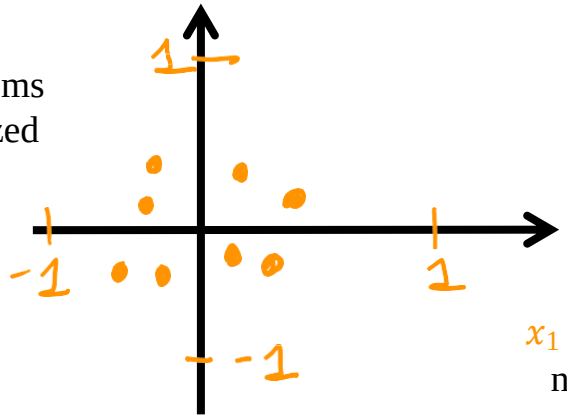


$$300 \le x_1 \le 2000 \qquad 0 \le x_2 \le 5$$

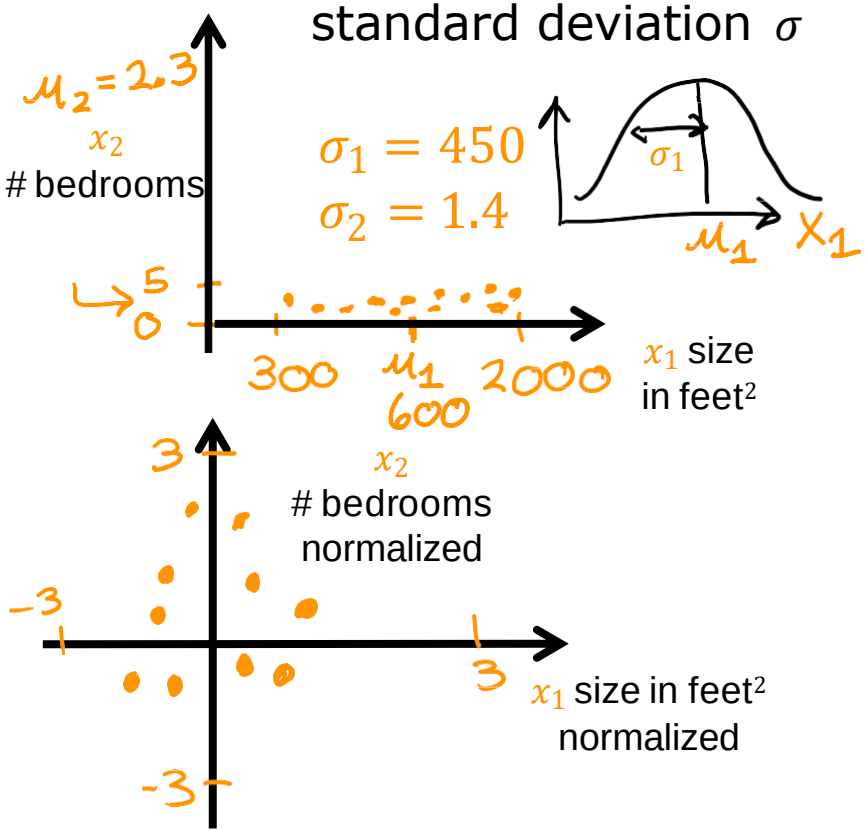
$$x_1 = \frac{x_1 - \mu_1}{2000 - 300} \qquad x_2 = \frac{x_2 - \mu_2}{5 - 0}$$

max-min max-min

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



Z-score normalization



$$300 \le x_1 \le 2000$$

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$$

$$-0.67 \le x_1 \le 3.1$$

$$0 \le x_2 \le 5$$

$$x_2 = \frac{x_2 - \mu_2}{\sigma_2}$$

$$-1.6 \le x_2 \le 1.9$$

Feature scaling

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

$-3 \le x_j \le 3$

$-0.3 \le x_j \le 0.3$

} 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

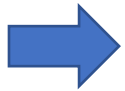
$-0.001 \le x_4 \le 0.001$ too small → rescale

$98.6 \le x_5 \le 105$ too large → rescale

To conclude ...

- There's almost never any harm to carrying out feature re-scaling. When in doubt, I encourage you to just carry it out.
- That's it for feature scaling. With this little technique, you'll often be able to get gradient descent to run much faster. That's features scaling.
- With or without feature scaling, when you run gradient descent, how can you know, how can you check if gradient descent is really working?
- If it is finding you the global minimum or something close to it.

In the next section, let's take a look at how to recognize if gradient descent is converging, and then in the video after that, this will lead to discussion of how to choose a good learning rate for gradient descent.



Checking Gradient Descent for Convergence

Gradient descent

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

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

these are the gradient descent algorithm equations, and one of the main choices of the parameters is the choice of alpha (a) the learning rate

So the question is how do we choose alpha

1. The number of iterations in gradient descent significantly impacts the cost function.
2. As the number of iterations increases, the cost function generally decreases.
3. The optimal number of iterations depends on various factors and may not be linear.

Make sure gradient descent is working correctly

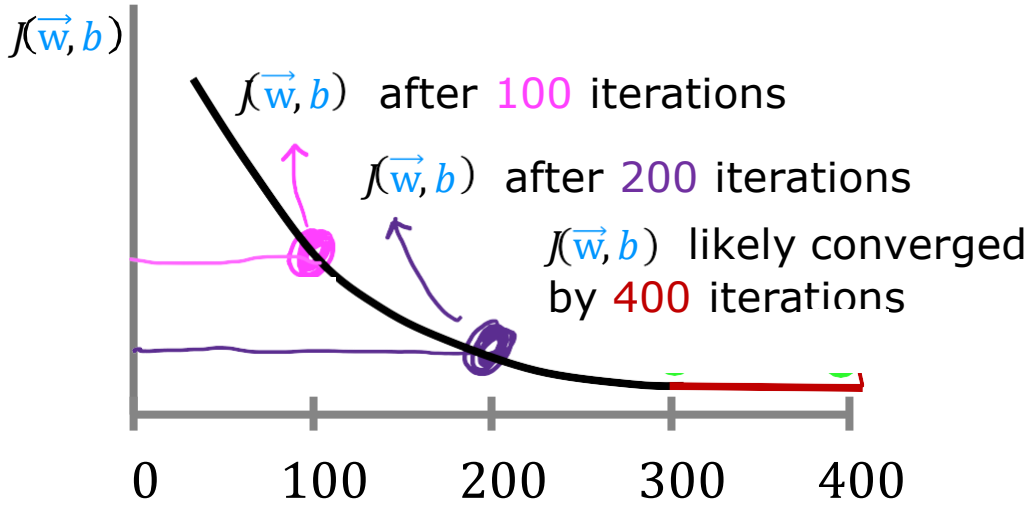
objective: $\min_{\vec{w},b} J(\vec{w}, b)$

$J(\vec{w}, b)$ should decrease after every iteration

plot J(w,b) as a function of number of iterations.
→ the cost function should decrease constantly

if the cost function increase after some iterations, then either alpha is chosen poorly, or there is a bug in the code

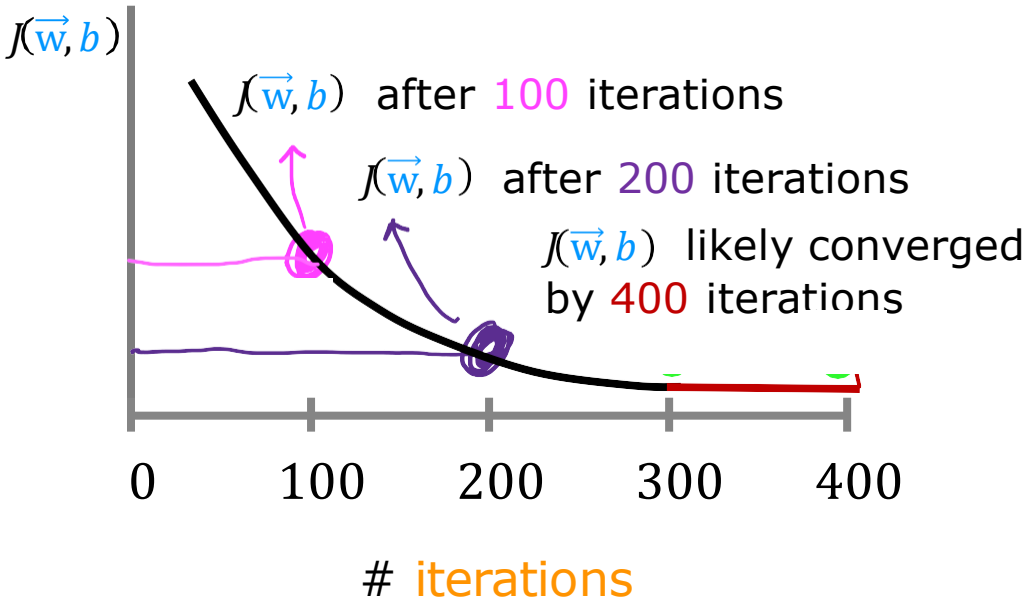
Learning Curve



iterations needed varies

Make sure gradient descent is working correctly

Learning Curve



Automatic convergence test

Let ε "epsilon" be 10^{-3}.
If $J(\vec{w}, b)$ decreases by $\leq \varepsilon$ in one iteration, you may declare convergence

Which means that you found parameters w and b to get close to global minimum)

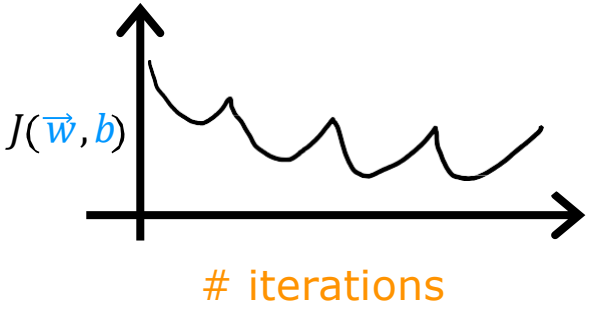
- Number of iterations varies for different applications, it may take 30, some other applications might take 1000, or 10,000 iterations.
- it might be difficult to tell in advance how many iterations gradient descent might need to converge

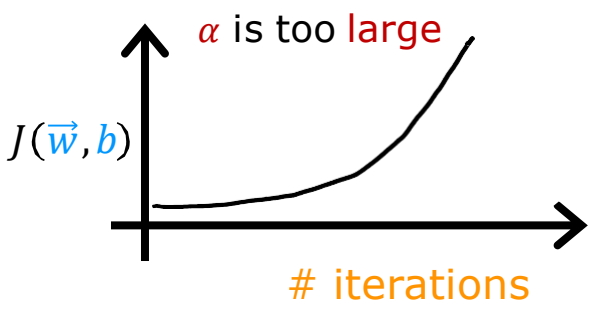
to conclude ...

- You've now seen what the learning curve should look like when gradient descent is running well.
- Let's take these insights and in the next part, take a look at how to choose an appropriate learning rate.

Choosing the Learning Rate

Identify problem with gradient descent





$w_1 = w_1 + \alpha d_1$

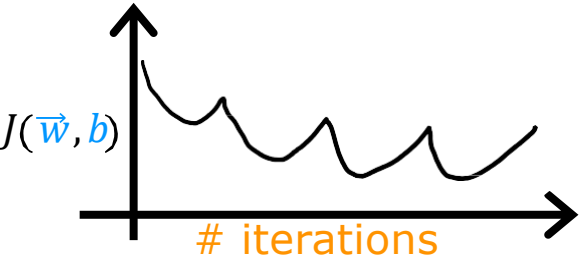
use a minus sign

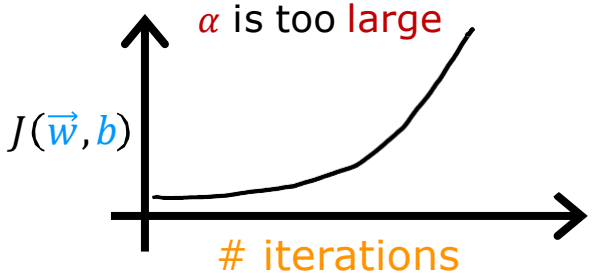
$w_1 = w_1 - \alpha d_1$

or

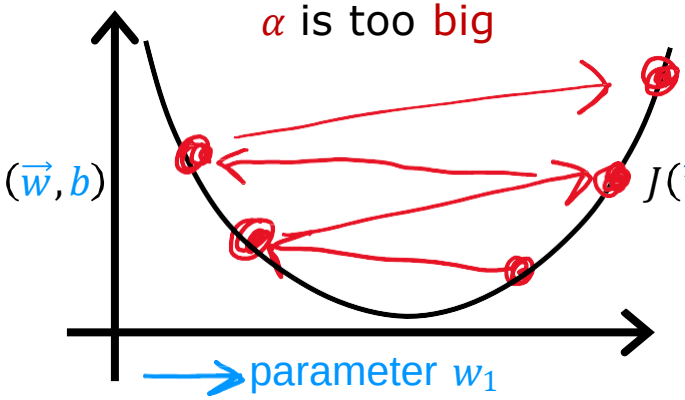
learning rate is too large

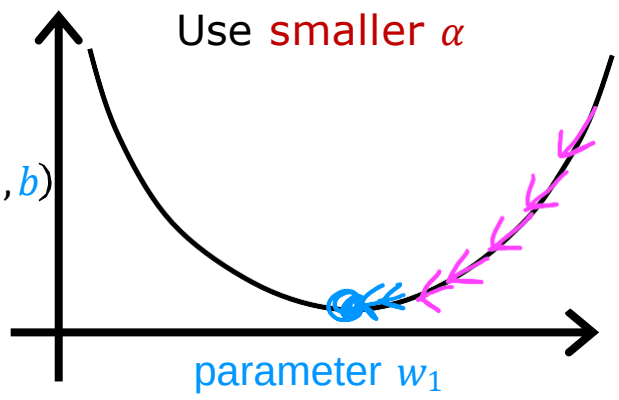
Identify problem with gradient descent





With a small enough α, $J(\vec{w}, b)$ should decrease on every iteration





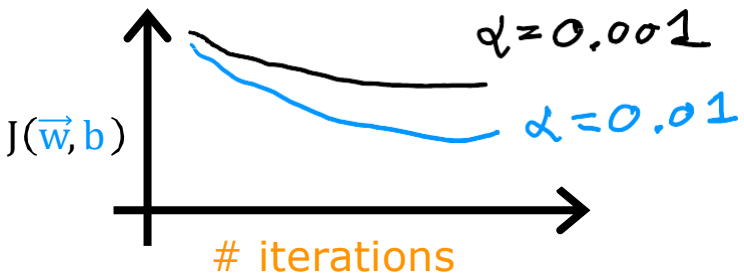
If α is too small, gradient descent takes a lot more iterations to converge

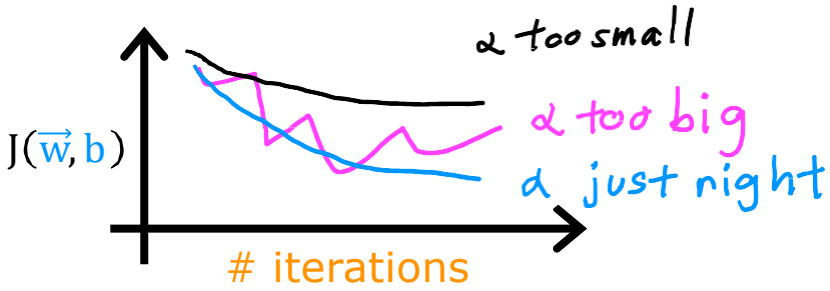
Since alpha is big you may skip the minimum, and that will make the cost function increases

Values of α to try:

… 0.001 0.003 0.01 0.03 0.1 0.3 1 …

3x ≈3x 3x ≈3x 3x ≈3x





Feature Engineering

Feature engineering is a crucial step in machine learning that involves transforming raw data into features suitable for machine learning algorithms.

Key objectives of feature engineering:

- Select relevant features: Identify and select the most relevant features from the dataset to eliminate noise and focus on important factors.
- Create new features: Derive new features from existing data to capture underlying relationships and patterns that may not be directly present in the raw data.
- Transform existing features: Transform existing features using techniques like scaling, normalization, and binning to improve model convergence and stabilize the training process.

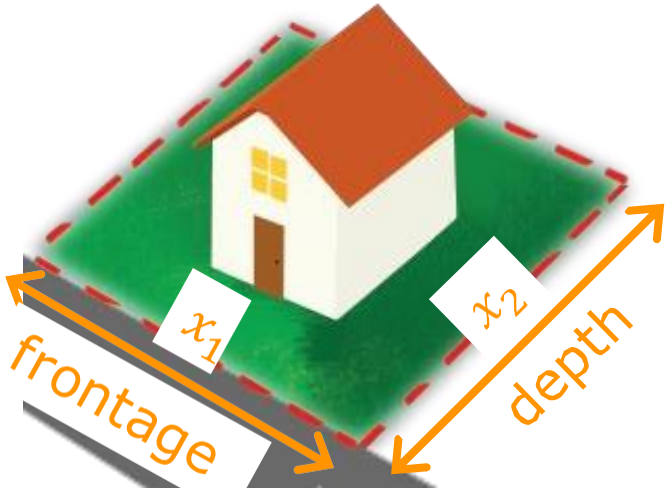
Benefits of feature engineering:

- Improved model performance: Enhances the accuracy and generalizability of machine learning models.
- Faster model training: Reduces the complexity of data and improves model convergence.
- Better understanding of data: Reveals underlying relationships and patterns in the data, leading to more meaningful insights.

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

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

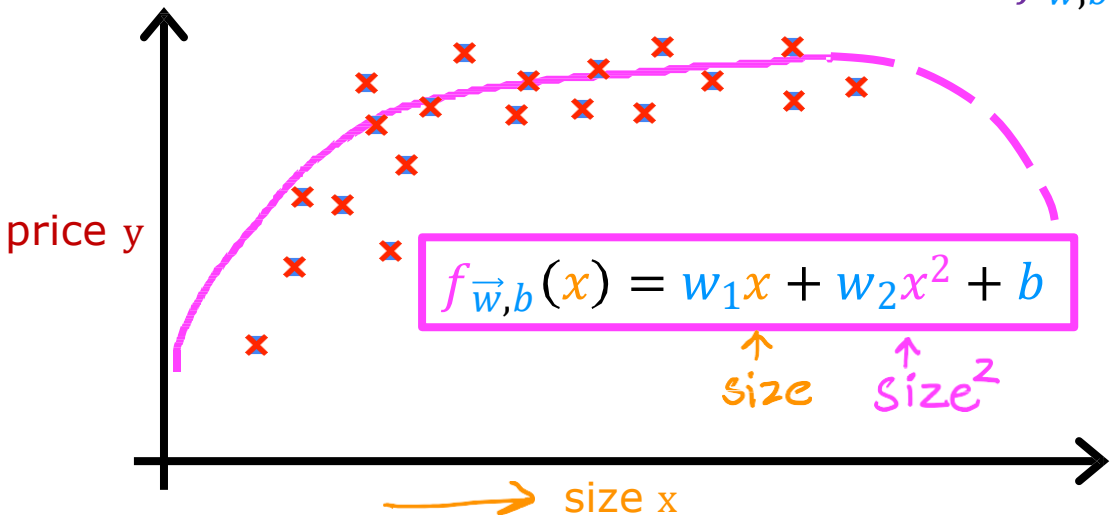
Feature engineering: Using intuition to design new features, by transforming or combining original features.

So by designing a new feature you migh get a better model

Polynomial Regression

Polynomial regression is a type of regression analysis that models the relationship between a response variable and one or more independent variables using a polynomial equation.
In other words, it assumes that the relationship between the variables can be expressed as a sum of terms, where each term is the product of a constant and a variable raised to an integer power.

Polynomial regression

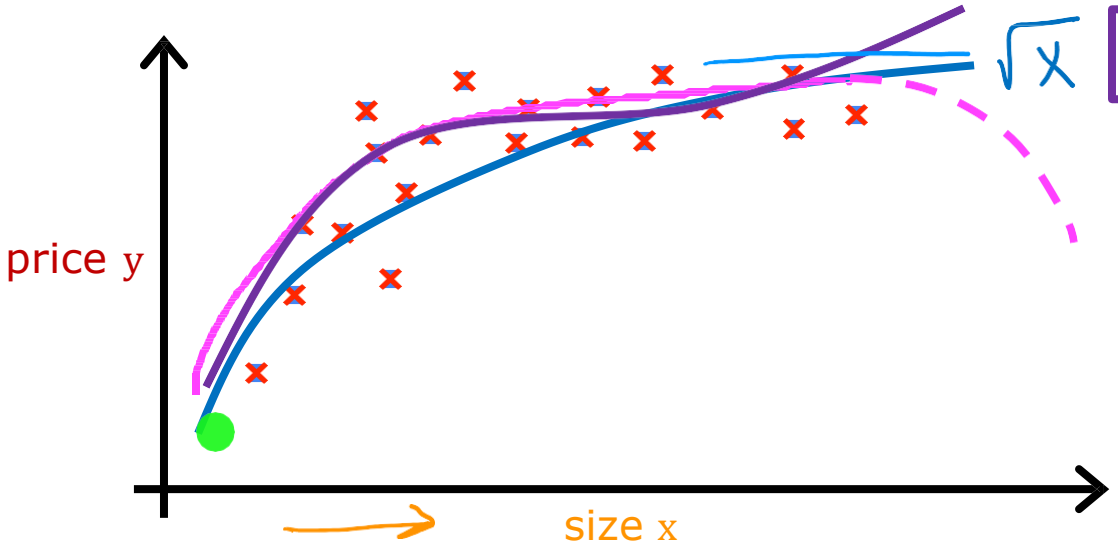


$$f_{w,b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + b$$

size $1-10^3$ | size^2 $1-10^6$ | size^3 $1-10^9$

feature scaling

Choice of features



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

size $\sqrt{\text{size}}$

what features to use?
↳ course 2