multiple variable for linear regression

| size in feet $x^1$ | number of bedroom $x^2$ | number of floor $x^3$ | Age of house $x^4$ | price |
|---|---|---|---|---|
| 210 | 5 | 1 | 40 | 460 |
| 220 | 3 | 2 | 41 | 232 |

$i=2$

$x_j = j^{th}$ feature $j = x_1\ x_2\ \dots\ x_4$

$n =$ number of features $n=4$

$\vec{x}^{(i)} =$ features of $i^{th}$ training example $\vec{x}^{(2)} = 220\ 3\ 2\ 41\ 232$

$\vec{x}^{(i)}_{(j)} =$ value of feature $j$ in $i^{th}$ training example $\vec{x}^{(2)}_{(2)} = 3$

multiple linear regression

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

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

$\vec{w} = [w_1\ w_2\ w_3\ \dots\ w_n] \Leftarrow$ vector of w

b is a number

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

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

dot product

---

Vectorization

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

python code

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

numpy

best and fastest

without using vectorization

$$f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} w_j x_j\right) + b$$

Python

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

slow

Gradient descent

repeat {

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

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

}

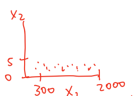
---

Feature scaling

- aim for featuring $-1 \le x_j \le 1$, but $0 \le x_1 \le 3$ is okey
- In order to make the prediction run faster, we need to scale features,
- If the value of x has a large difference from each other

Ex:

| $x_1$ | $x_2$ | Price |
|---|---|---|
| 200000 | 5 | 500 |
| 300 | 0 | 200 |




Gradient descent

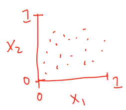
Feature scaling

$x_{1,scaled} = \frac{x_1}{max}$ $\quad x_{2,scaled} = \frac{x_2}{max}$

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

$x_{1,scaled} = \frac{x_1}{2000}$ $\quad x_{2,scaled} = \frac{x_2}{5}$

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

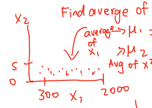



You can also do mean normalization

Find averge of $x_1, x_2$

average of $x_1 \to \mu_1 = 600$

$\mu_2 = 2.3 \leftarrow$ Avg of $x^2$



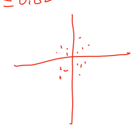

$$x_1 = \frac{x_1 - \mu_1}{2000 - 300}$$

max, min

$$-0.18 \le x_1 \le 0.82$$

$$x_2 = \frac{x_2 - \mu_2}{5 - 0}$$

$$-0.46 \le x_2 \le 0.54$$



Z-score normalization

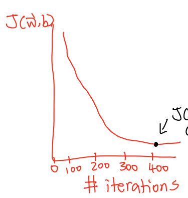
- calculate standard deviation of $x_1, x_2$ & $\mu_1, \mu_2$

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

Standard deviation

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

Making sure gradient descent is working is by plotting




$J(\vec{w},b)$ converges by 400 iteration (it doesn't change anymore)

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

- If $J(\vec{w},b)$ increases at some iteration
  - $\alpha$ is too large
  - code have bug

Automatic convergence Test

let $\varepsilon$ be a very small number like $10^{-3}$

epsilon

If $J(\vec{w},b)$ decreases by $\le \varepsilon$ in one iteration, it convergence

Finding good learning rate Technique

Try small number first and then increase

$$0.001 \xrightarrow{\times 3} 0.003 \xrightarrow{\times 3} 0.01 \xrightarrow{\times 3} \dots$$

then choose a good learning rate

# Feature Engineering

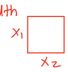
- using intuition to design new features, by transforming or combining original features

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

we know that area = width x width

so we can combine $x_1$ and $x_2$

$x_3 = x_1(x_2)$




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

---

# Polynomial regression

$f_{\vec{w}b}(x) = w_1x_1 + w_2x_2^2$