# CSE 4705 - Artificial Intelligence

## Multiple Regression

Joe Johnson M.S., Ph.D., A.S.A.

Department of Computer Science and Engineering
University of Connecticut

UCONN | UNIVERSITY OF CONNECTICUT

# Table of Contents

# Table of Contents

# Table of Contents

# Training Set and Notation

Our dataset for the univariate linear regression case:

# Training Set and Notation

Our dataset for the univariate linear regression case:

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

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

# Training Set and Notation

Now we have the following for our dataset:

# Training Set and Notation

Now we have the following for our dataset:

## Multiple features (variables)

| Size in feet$^2$ | Number of bedrooms | Number of floors | Age of home in years | Price ($) in $1000's |
|---|---|---|---|---|
| 2104 | 5 | 1 | 45 | 460 |
| 1416 | 3 | 2 | 40 | 232 |
| 1534 | 3 | 2 | 30 | 315 |
| 852 | 2 | 1 | 36 | 178 |
| ... | ... | ... | ... | ... |

# Training Set and Notation

Now, let's consider notation:

# Training Set and Notation

Now, let's consider notation:

## Multiple features (variables)

| | 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 |
| $i=2$ | 1416 | 3 | 2 | 40 | 232 |
| | 1534 | 3 | 2 | 30 | 315 |
| | 852 | 2 | 1 | 36 | 178 |
| | ... | ... | ... | ... | ... |

# Training Set and Notation

## Multiple features (variables)

| | Size in feet$^2$ $X_1$ | 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 |
| $i=2$ | 1416 | 3 | 2 | 40 | 232 |
| | 1534 | 3 | 2 | 30 | 315 |
| | 852 | 2 | 1 | 36 | 178 |
| | ... | ... | ... | ... | ... |

Notation:

# Training Set and Notation

## Multiple features (variables)

| | 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 |
| $i=2$ | 1416 | 3 | 2 | 40 | 232 |
| | 1534 | 3 | 2 | 30 | 315 |
| | 852 | 2 | 1 | 36 | 178 |
| | ... | ... | ... | ... | ... |

Notation:

- $n$ = number of features

# Training Set and Notation

## Multiple features (variables)

| | 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 |
| $i=2$ | 1416 | 3 | 2 | 40 | 232 |
| | 1534 | 3 | 2 | 30 | 315 |
| | 852 | 2 | 1 | 36 | 178 |
| | ... | ... | ... | ... | ... |

Notation:

- $n$ = number of features
- $m$ = number of samples

# Training Set and Notation

## Multiple features (variables)

| | 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 |
| $i=2$ | 1416 | 3 | 2 | 40 | 232 |
| | 1534 | 3 | 2 | 30 | 315 |
| | 852 | 2 | 1 | 36 | 178 |
| | ... | ... | ... | ... | ... |

Notation:

- $n$ = number of features
- $m$ = number of samples (or data objects)

# Training Set and Notation

## Multiple features (variables)

| | 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 |
| $i=2$ | 1416 | 3 | 2 | 40 | 232 |
| | 1534 | 3 | 2 | 30 | 315 |
| | 852 | 2 | 1 | 36 | 178 |
| | ... | ... | ... | ... | ... |

Notation:

- $n$ = number of features
- $m$ = number of samples (or data objects) in our training set

# Training Set and Notation

## Multiple features (variables)

| | 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 |
| $i=2$ | 1416 | 3 | 2 | 40 | 232 |
| | 1534 | 3 | 2 | 30 | 315 |
| | 852 | 2 | 1 | 36 | 178 |
| | ... | ... | ... | ... | ... |

Notation:

- $n$ = number of features
- $m$ = number of samples (or data objects) in our training set
- $\vec{x_j}$ = vector of values for the $j^{th}$ feature

# Training Set and Notation

## Multiple features (variables)

| | 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 |
| $i=2$ | 1416 | 3 | 2 | 40 | 232 |
| | 1534 | 3 | 2 | 30 | 315 |
| | 852 | 2 | 1 | 36 | 178 |
| | ... | ... | ... | ... | ... |

Notation:

- $n$ = number of features
- $m$ = number of samples (or data objects) in our training set
- $\vec{x_j}$ = vector of values for the $j^{th}$ feature
- $\vec{x}^{(i)}$ = vector of features for the $i^{th}$ training example

# Training Set and Notation

## Multiple features (variables)

| | Size in feet$^2$ | 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 |
| | ... | ... | ... | ... | ... |

Notation:

- $n$ = number of features
- $m$ = number of samples (or data objects) in our training set
- $\vec{x_j}$ = vector of values for the $j^{th}$ feature
- $\vec{x}^{(i)}$ = vector of features for the $i^{th}$ training example
- $x_j^{(i)}$ = value of feature $j$ for the $i^{th}$ training example

# Table of Contents

# Table of Contents

# Multiple Regression Model

Model Notation:

# Multiple Regression Model

Model Notation:

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

# Multiple Regression Model

Model Notation:

- Model: $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + \text{b}$
- Model parameters:

# Multiple Regression Model

Model Notation:

- Model: $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + \text{b}$
- Model parameters:
  - $\vec{w} = [w_1 \ w_2 \ w_3 \ \ldots w_n]$

# Multiple Regression Model

Model Notation:

- Model: $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + \text{b}$
- Model parameters:
  - $\vec{w} = [w_1 \ w_2 \ w_3 \ \ldots w_n]$ (row vector)

# Multiple Regression Model

Model Notation:

- Model: $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + \text{b}$
- Model parameters:
  - $\vec{w} = [w_1 \ w_2 \ w_3 \ \ldots w_n]$ (row vector)
  - $b$

# Multiple Regression Model

Model Notation:

- Model: $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + \text{b}$
- Model parameters:
  - $\vec{w} = [w_1 \ w_2 \ w_3 \ \ldots w_n]$ (row vector)
  - $b$
- Model (Vector Notation):

# Multiple Regression Model

Model Notation:

- Model: $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + \text{b}$
- Model parameters:
  - $\vec{w} = [w_1 \ w_2 \ w_3 \ \ldots w_n]$ (row vector)
  - $b$
- Model (Vector Notation): $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

# Table of Contents

# Univariate Regression Example: House Prices Model

Recall gradient descent for the univariate case, $f_{w,b}(x)$:

# Univariate Regression Example: House Prices Model

Recall gradient descent for the univariate case, $f_{w,b}(x)$:

1. Initialize $w, b$ to random values.

# Univariate Regression Example: House Prices Model

Recall gradient descent for the univariate case, $f_{w,b}(x)$:

1. Initialize $w, b$ to random values.
2. Repeat until convergence (simultaneous update):

# Univariate Regression Example: House Prices Model

Recall gradient descent for the univariate case, $f_{w,b}(x)$:

1. Initialize $w, b$ to random values.
2. Repeat until convergence (simultaneous update):

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

# Univariate Regression Example: House Prices Model

Substituting the partials for the univariate case, we have:

# Univariate Regression Example: House Prices Model

Substituting the partials for the univariate case, we have:

1. Initialize $w, b$ to random values.

# Univariate Regression Example: House Prices Model

Substituting the partials for the univariate case, we have:

1. Initialize $w, b$ to random values.
2. Repeat until convergence (simultaneous update):

# Univariate Regression Example: House Prices Model

Substituting the partials for the univariate case, we have:

1. Initialize $w, b$ to random values.
2. Repeat until convergence (simultaneous update):

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

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

# Univariate Regression Example: House Prices Model

Now, for the multiple regression case, $f_{\vec{w},b}(\vec{x})$, we have:

# Univariate Regression Example: House Prices Model

Now, for the multiple regression case, $f_{\vec{w},b}(\vec{x})$, we have:

1. Initialize $w_1, w_2, w_3, \ldots w_n, b$ to random values.

# Univariate Regression Example: House Prices Model

Now, for the multiple regression case, $f_{\vec{w},b}(\vec{x})$, we have:

1. Initialize $w_1, w_2, w_3, \ldots w_n, b$ to random values.
2. Repeat until convergence (simultaneous update):

# Univariate Regression Example: House Prices Model

Now, for the multiple regression case, $f_{\vec{w},b}(\vec{x})$, we have:

1. Initialize $w_1, w_2, w_3, \ldots w_n, b$ to random values.
2. Repeat until convergence (simultaneous update):

$$w_1 := w_1 - \alpha \frac{\partial J(\vec{w}, b)}{\partial w_1}$$
$$w_2 := w_2 - \alpha \frac{\partial J(\vec{w}, b)}{\partial w_2}$$
$$\vdots$$
$$w_n := w_n - \alpha \frac{\partial J(\vec{w}, b)}{\partial w_n}$$
$$b := b - \alpha \frac{\partial J(\vec{w}, b)}{\partial b}$$

# Univariate Regression Example: House Prices Model

Substituting the partials for the multiple regression case, we have:

# Univariate Regression Example: House Prices Model

Substituting the partials for the multiple regression case, we have:

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

$$\vdots$$

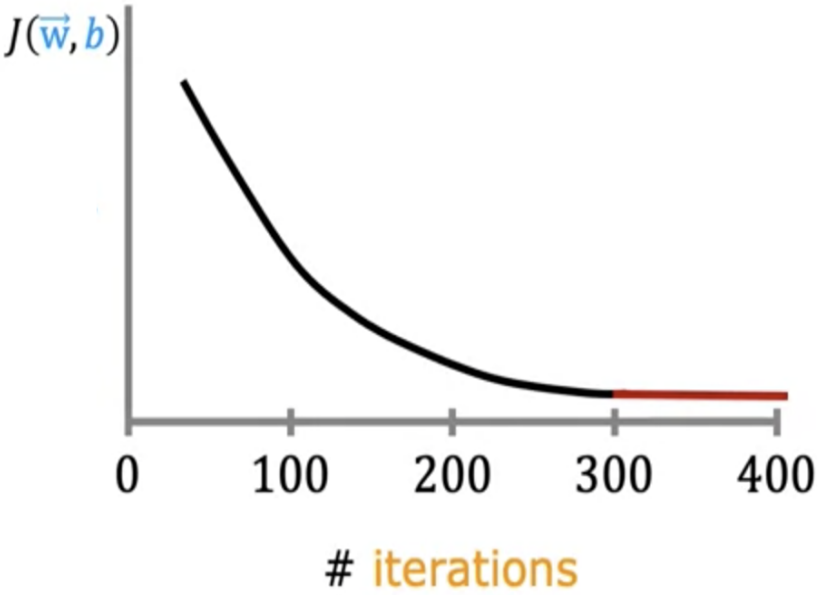
$$w_n := w_n - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) \cdot x_n^{(i)}}_{\frac{\partial J(\vec{w},b)}{\partial w_n}}$$

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

# Checking for Convergence

## Make sure gradient descent is working correctly

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

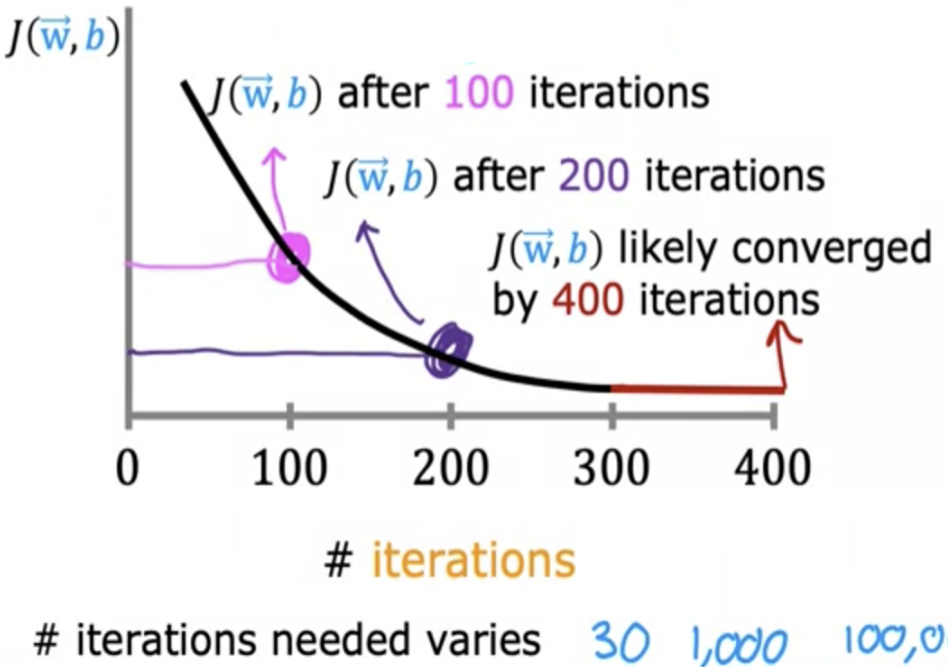


# Checking for Convergence

## 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




# iterations needed varies 30 1,000 100,000

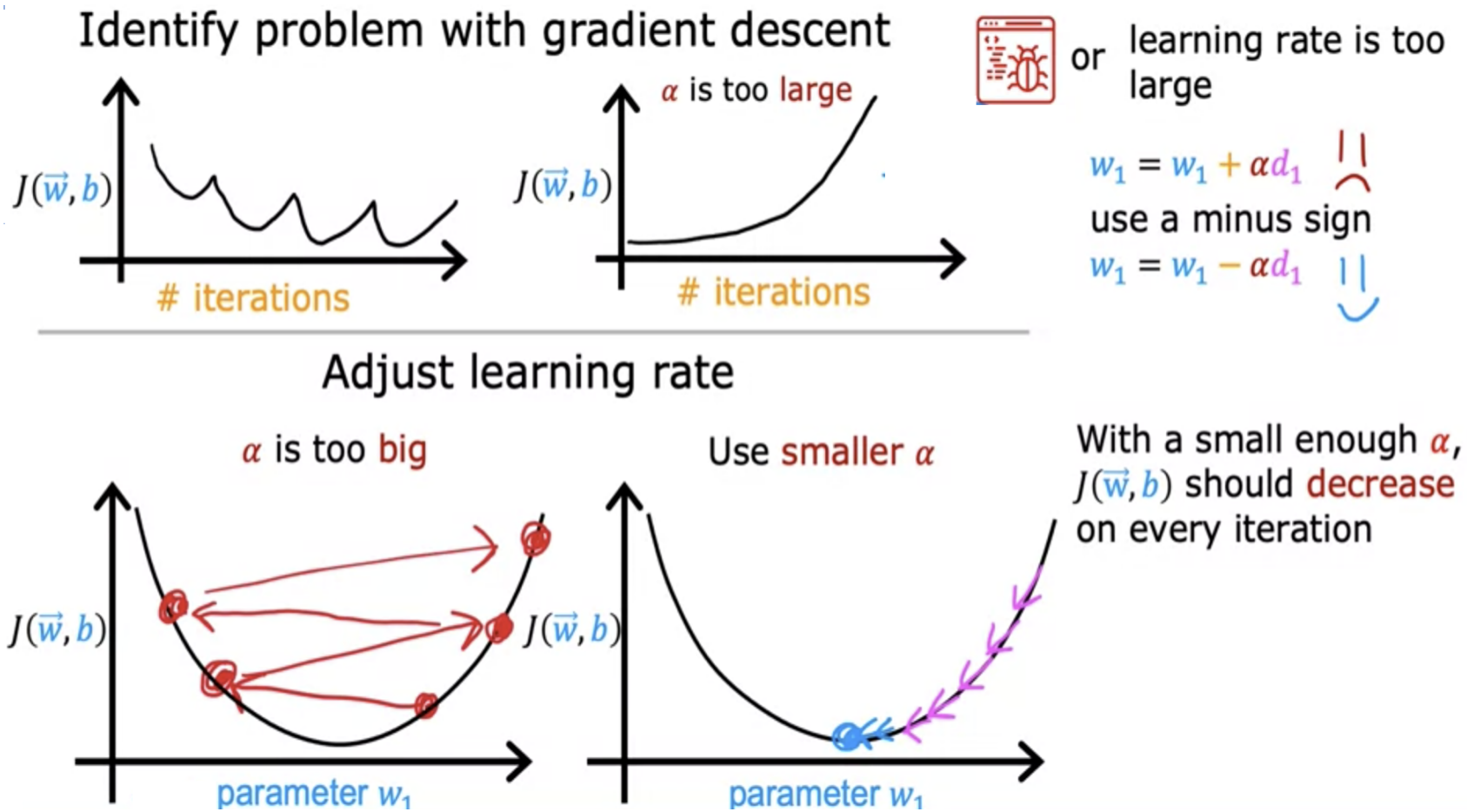
Automatic convergence test

Let $\varepsilon$ "epsilon" be $10^{-3}$.

If $J(\vec{w},b)$ decreases by $\leq \varepsilon$ in one iteration, declare convergence.

(found parameters $\vec{w},b$ to get close to global minimum)

# Checking Learning Rate

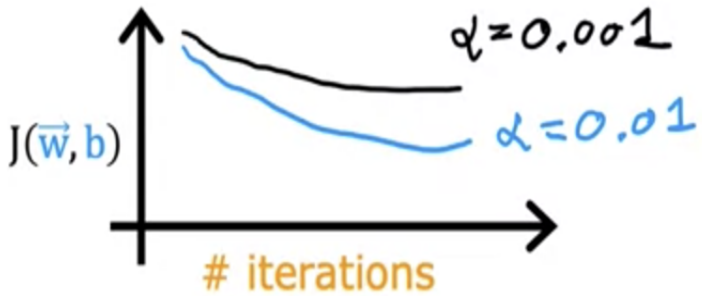


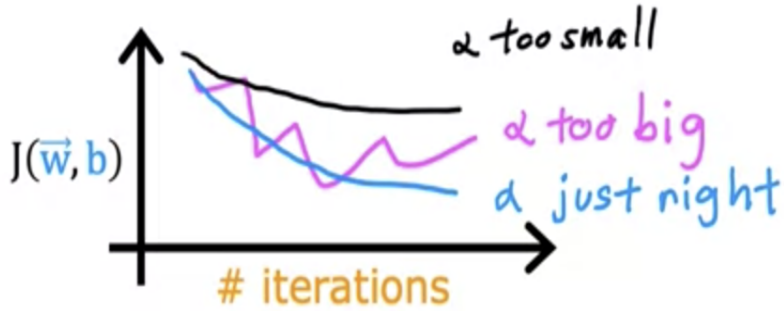
# Learning Rate Optimization

Values of $\alpha$ to try:

... 0.001 0.003 0.01 0.03 0.1 0.3 1 ...

3X ≈3X 3X ≈3X 3X ≈3X



# Table of Contents

# Table of Contents

# Feature Scaling

## Feature and parameter values

$\hat{price} = w_1 x_1 + w_2 x_2 + b$

size #bedrooms

$x_1$: size (feet$^2$) range: 300 – 2,000 large

$x_2$: # bedrooms range: 0 – 5 small

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

size of the parameters $w_1, w_2$?

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

$\hat{price} = \underbrace{50 * 2000}_{100{,}000\,k} + \underbrace{0.1 * 5}_{0.5\,k} + \underbrace{50}_{50k}$

$\hat{price} = \$100{,}050.5k = \$100{,}050{,}500$

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

$w_1$ small, $w_2$ large

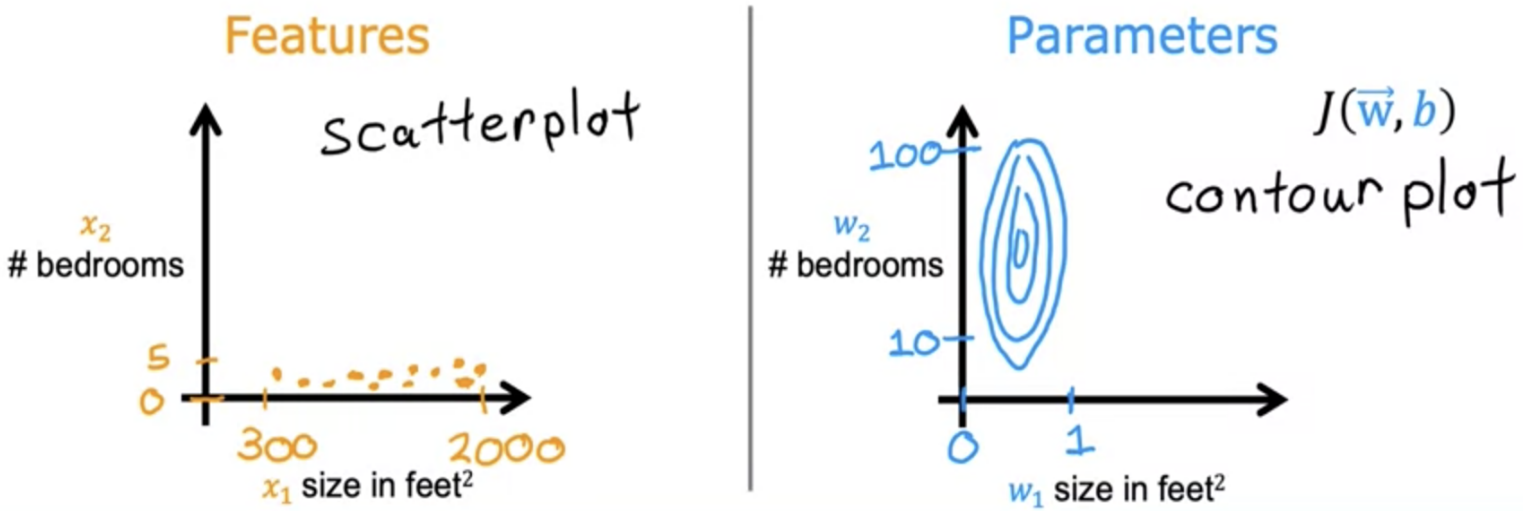
$\hat{price} = \underbrace{0.1 * 2000k}_{200k} + \underbrace{50 * 5}_{250k} + \underbrace{50}_{50k}$

$\hat{price} = \$500k$ more reasonable

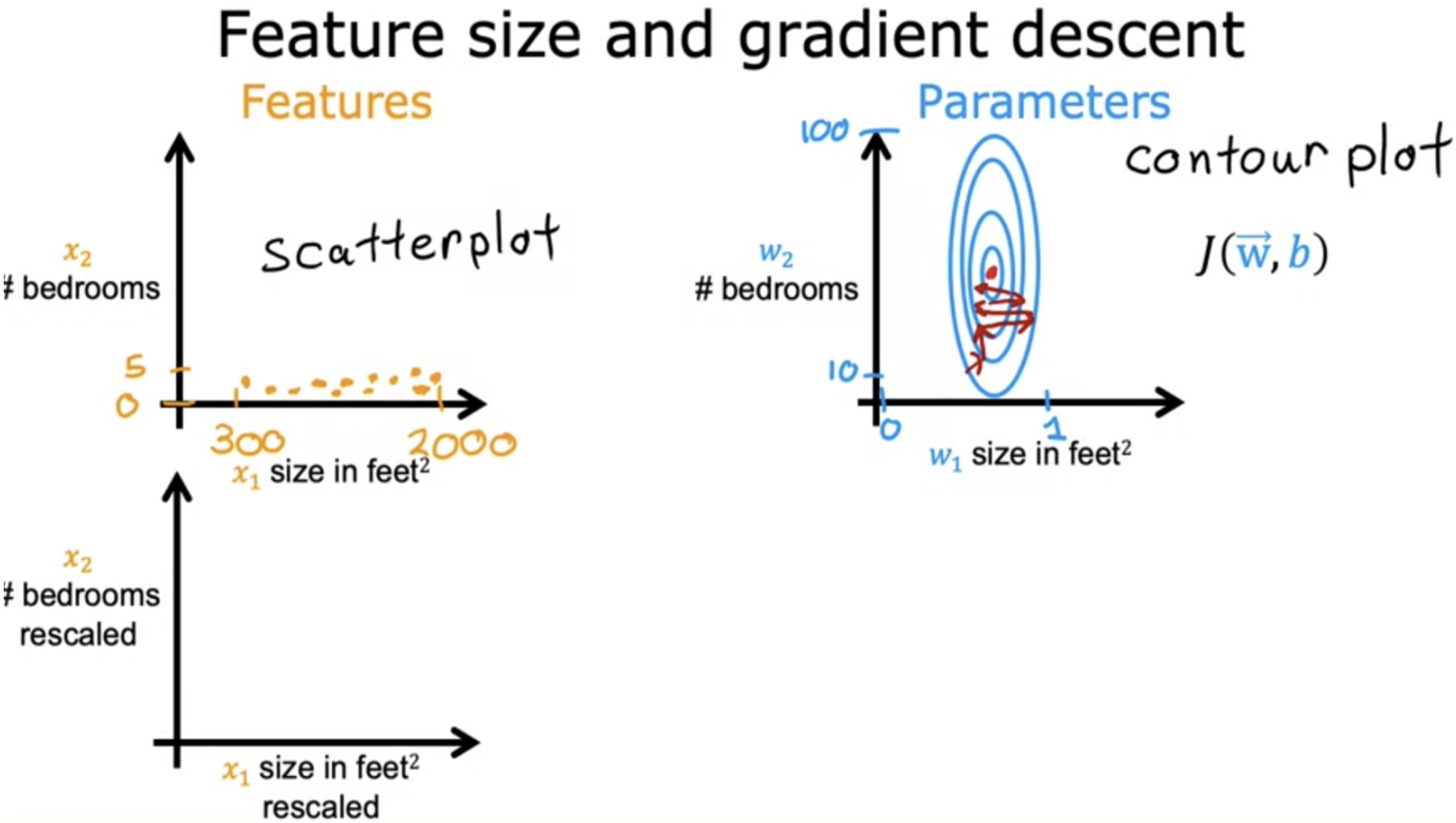
# Feature Scaling

## Feature size and parameter size

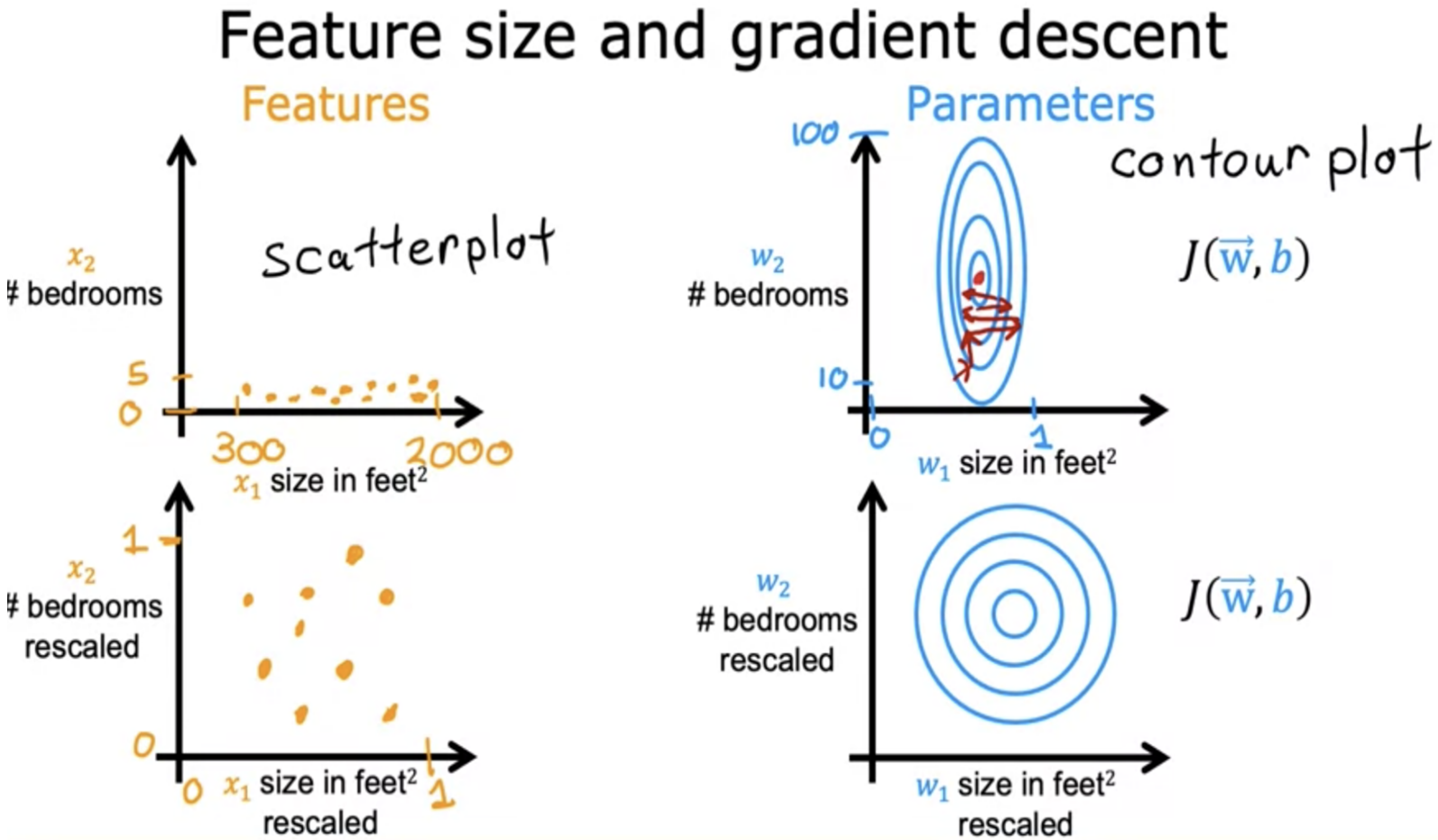
| | size of feature $x_j$ | size of parameter $w_j$ |
|---|---|---|
| size in feet$^2$ | ⟷ | ⟷ |
| #bedrooms | ⟷ | ⟷ |



# Feature Scaling

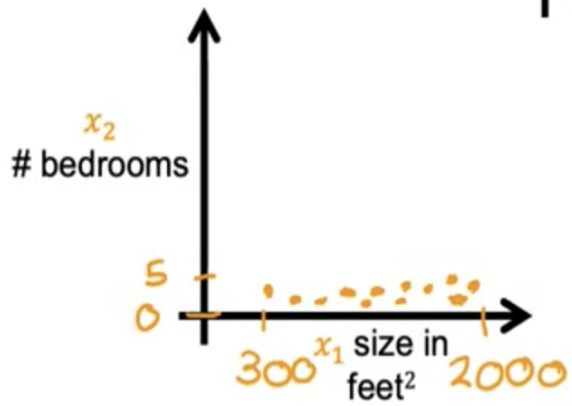


# Feature Scaling



# Feature Scaling

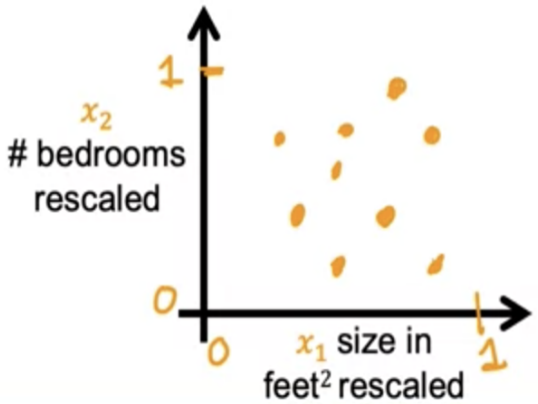
## Feature scaling




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

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

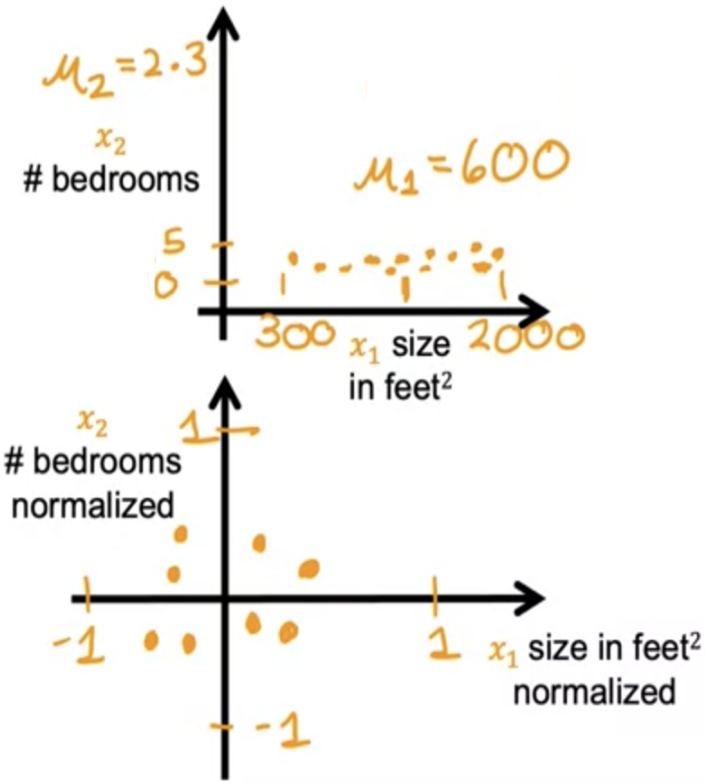
max max




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

# Feature Scaling

## Mean normalization




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

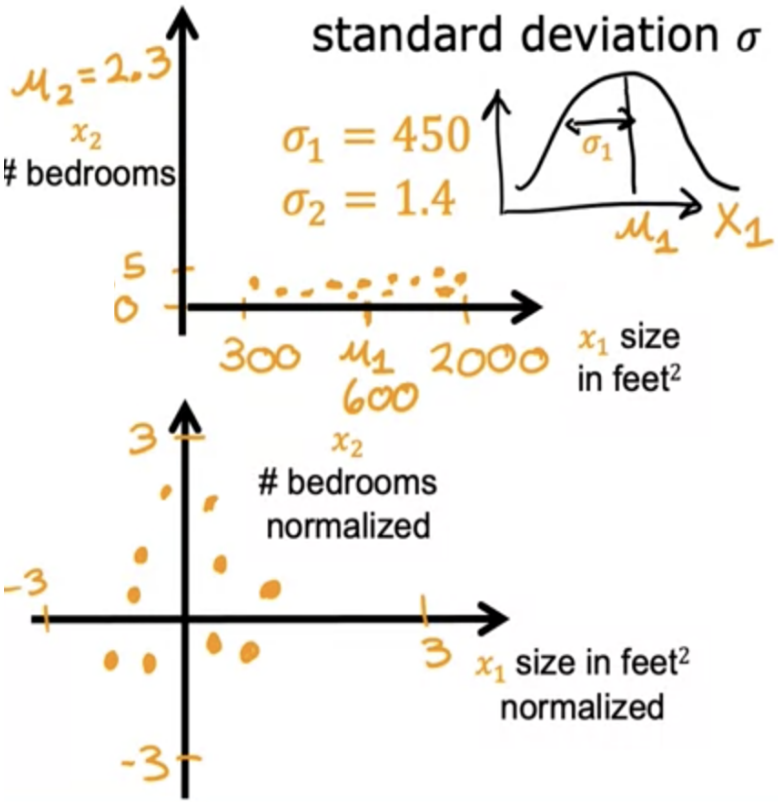
$$x_1 = \frac{x_1 - \mu_1}{2000 - 300} \qquad \text{max-min}$$

$$x_2 = \frac{x_2 - \mu_2}{5 - 0} \qquad \text{max-min}$$

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

# Feature Scaling

## Z-score normalization




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

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1} \qquad x_2 = \frac{x_2 - \mu_2}{\sigma_2}$$

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

# Feature Scaling

## Feature scaling

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

$-3 \leq x_j \leq 3$

$-0.3 \leq x_j \leq 0.3$

} acceptable ranges

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

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

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

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

$98.6 \leq x_5 \leq 105$ too large

# Table of Contents

# Feature Engineering - Example

## 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 - Definition

Feature Engineering:

# Feature Engineering - Definition

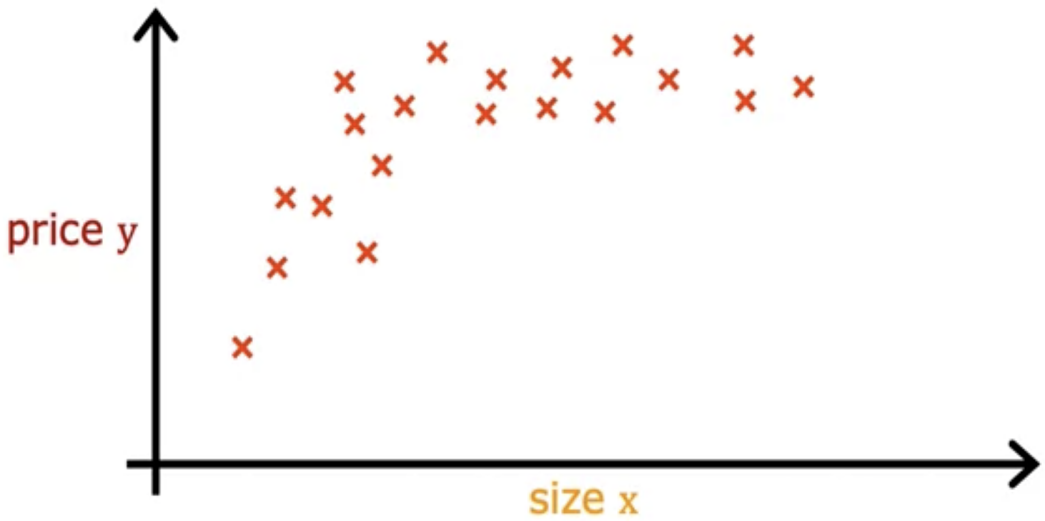
Feature Engineering: is the use of intuition and problem domain knowledge

# Feature Engineering - Definition

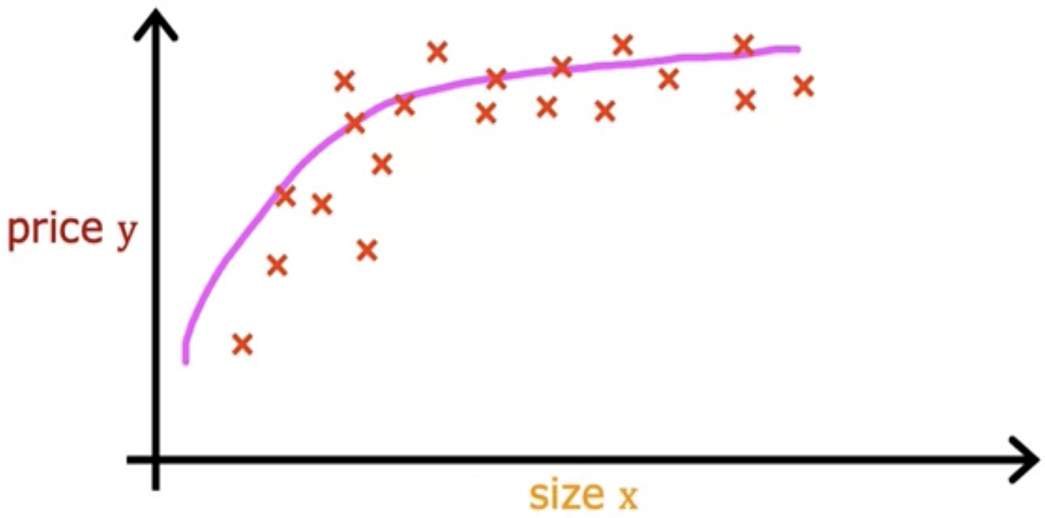
Feature Engineering: is the use of intuition and problem domain knowledge to design new features by transforming or combining original features in order to improve the predictive power of the model

# Table of Contents

# Polynomial Regression

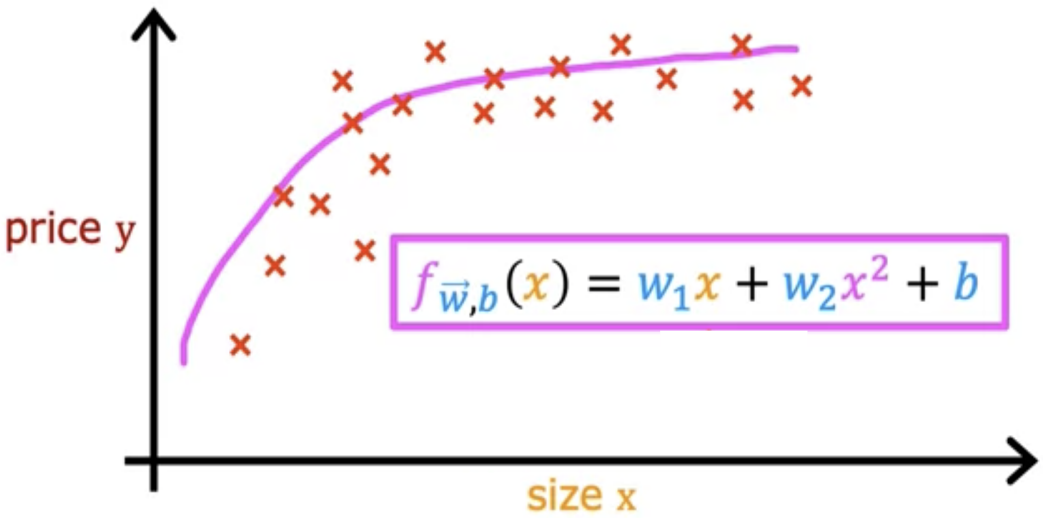


# Polynomial Regression



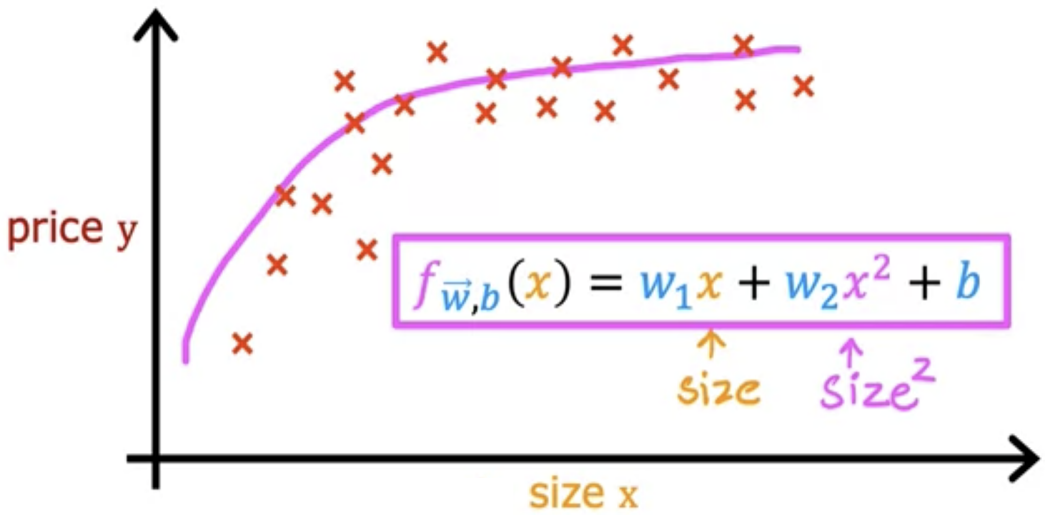
# Polynomial Regression

## Polynomial regression



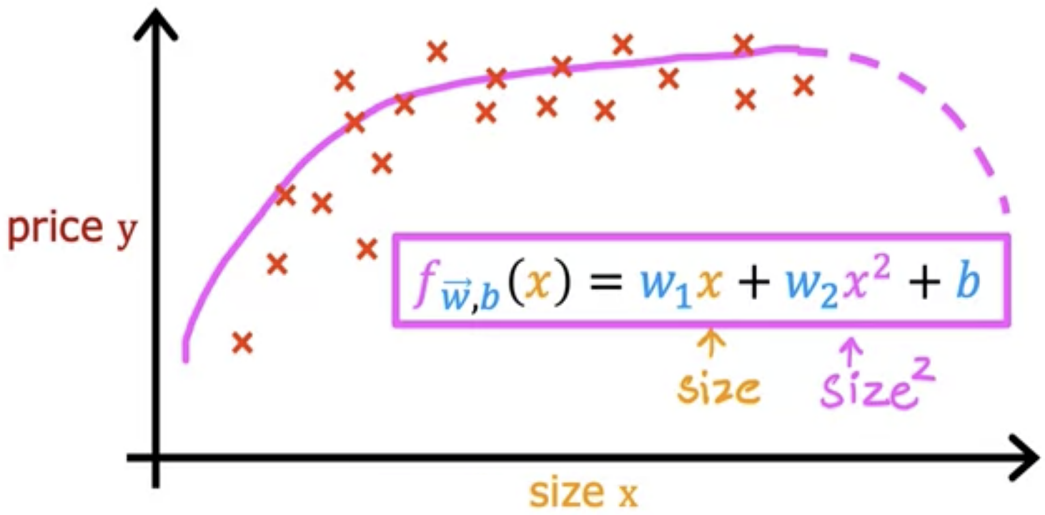
# Polynomial Regression

## Polynomial regression



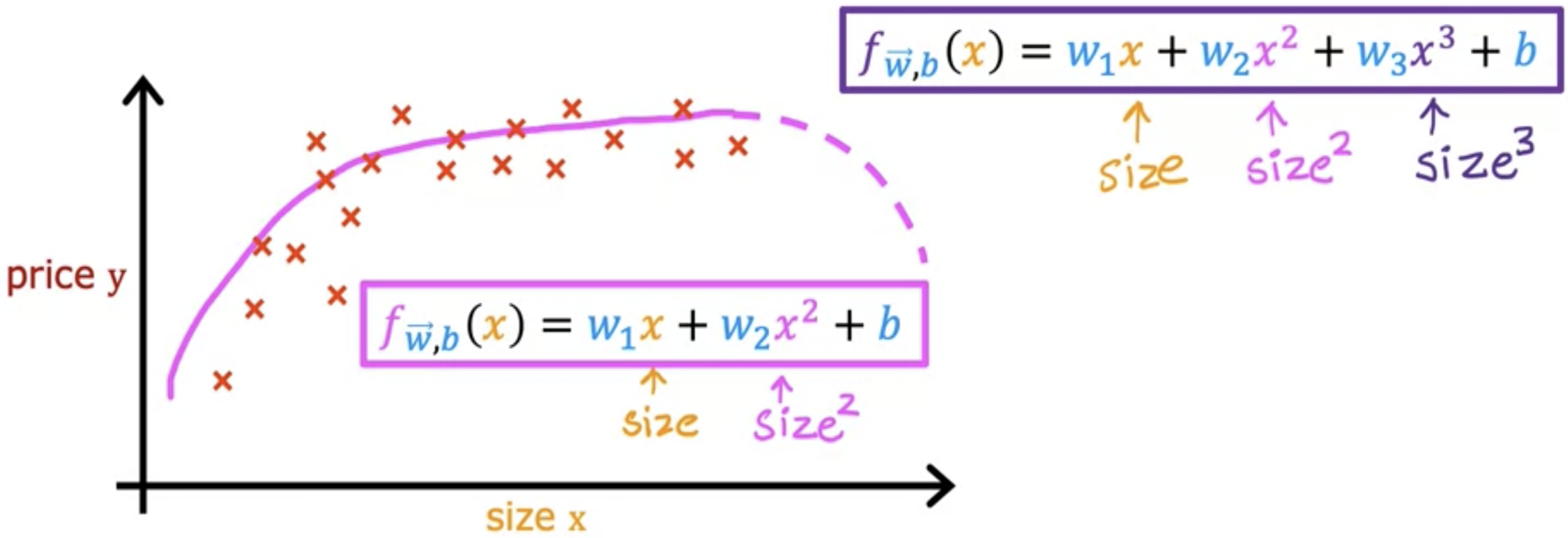
# Polynomial Regression

## Polynomial regression

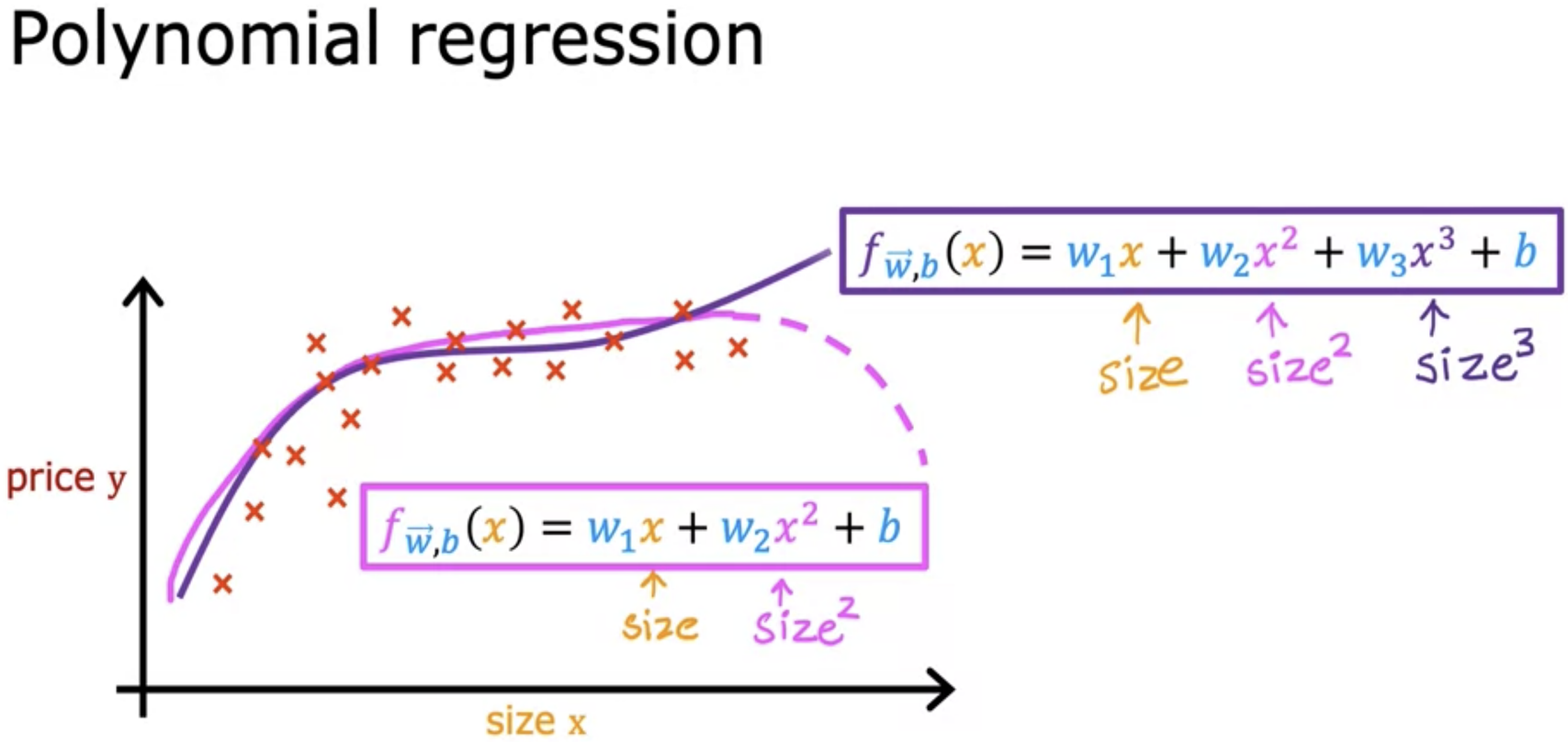


# Polynomial Regression

## Polynomial regression

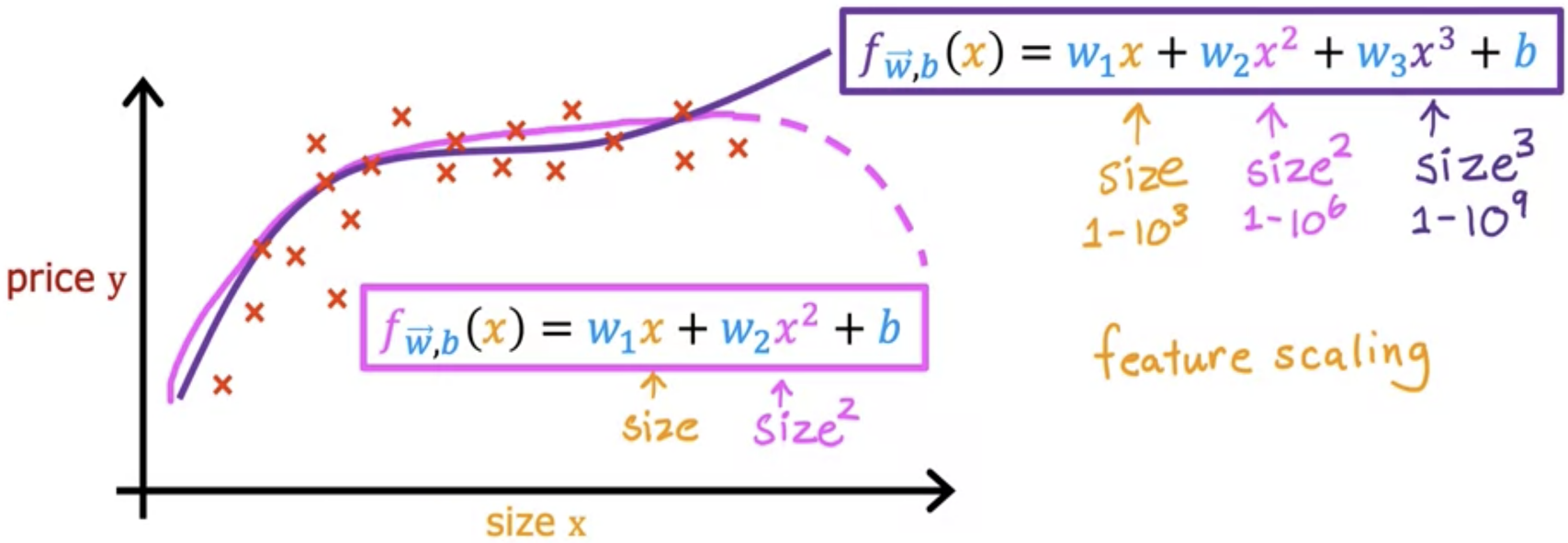


# Polynomial Regression



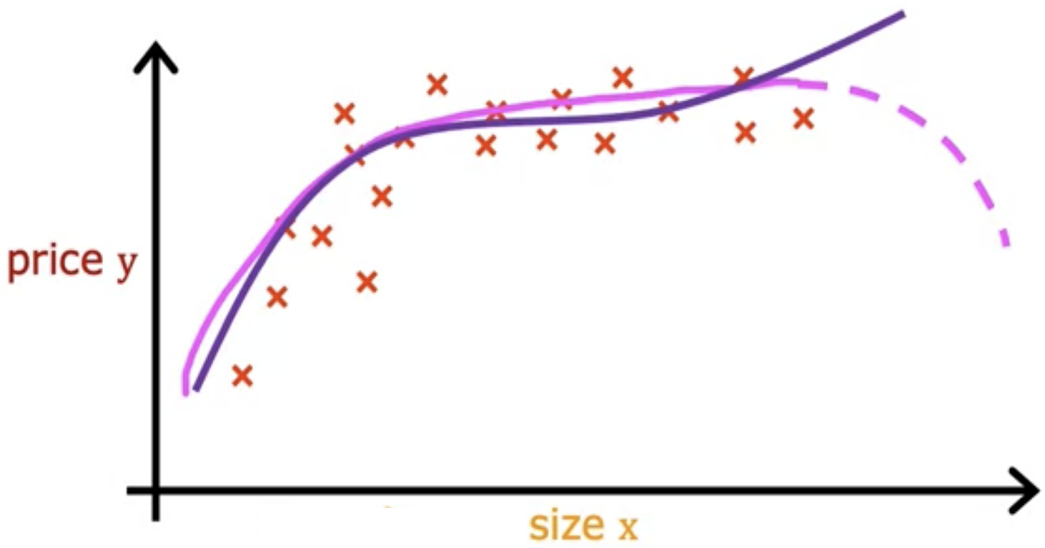
# Polynomial Regression

## Polynomial regression



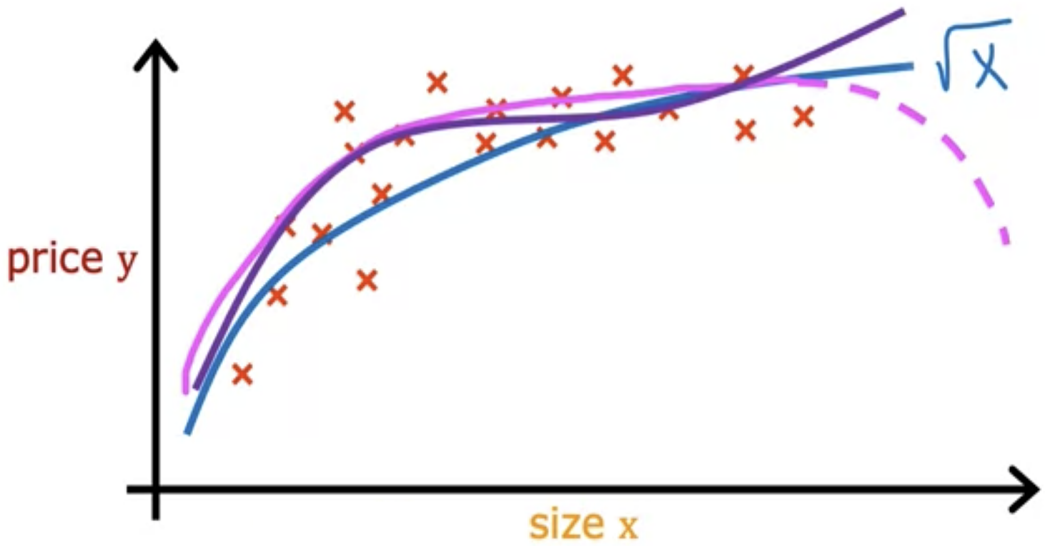
# Polynomial Regression

## Choice of features



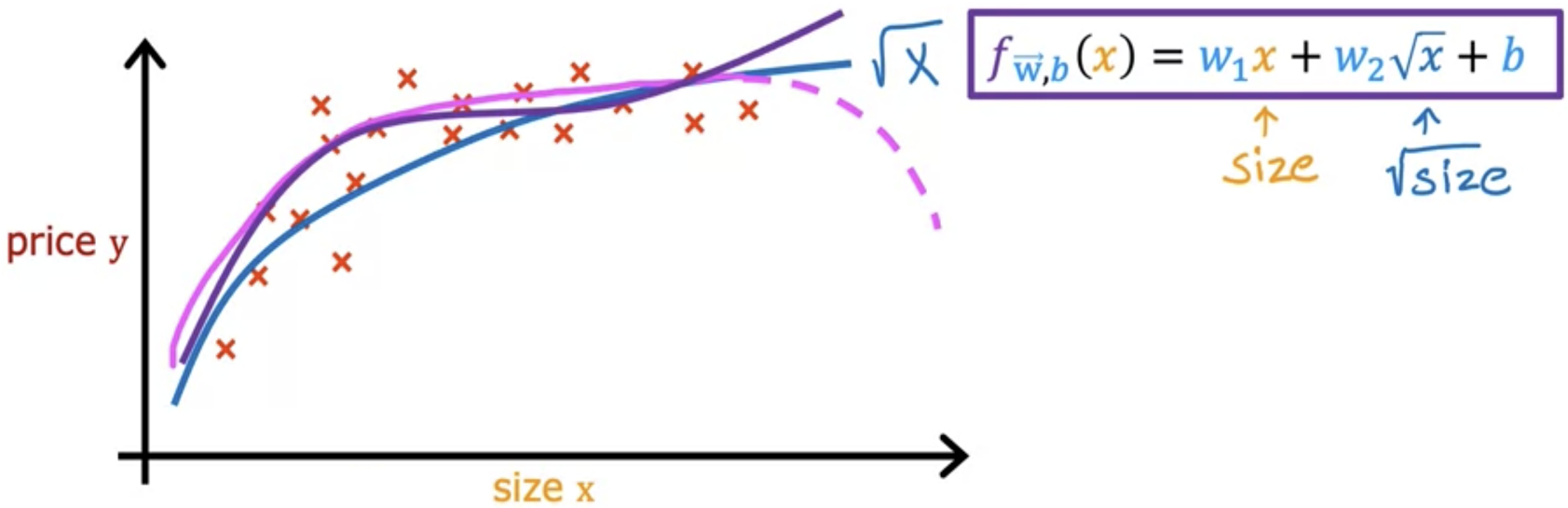
# Polynomial Regression

## Choice of features



# Polynomial Regression

## Choice of features



# Table of Contents

# Table of Contents

# Vectorization

- Computation speed is an important issue in ML,

# Vectorization

- Computation speed is an important issue in ML, especially as the size of our datasets gets large

# Vectorization

- Computation speed is an important issue in ML, especially as the size of our datasets gets large both in terms of the number of samples and the number of features.

# Vectorization

- Computation speed is an important issue in ML, especially as the size of our datasets gets large both in terms of the number of samples and the number of features.
- We leverage the engineering of computing platforms to address this -

# Vectorization

- Computation speed is an important issue in ML, especially as the size of our datasets gets large both in terms of the number of samples and the number of features.
- We leverage the engineering of computing platforms to address this - **Numpy**.

# Vectorization

**Numpy:**

# Vectorization

**Numpy:**

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

# Vectorization

**Numpy:**

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

- 0-index in code vs. 1-index in linear algebra

# Vectorization

Consider once again our model:

# Vectorization

Consider once again our model:

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

# Vectorization

Consider once again our model:

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

How do we implement the computation of $f_{\vec{w},b}$ in code in code?

# Vectorization

Consider once again our model:

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

How do we implement the computation of $f_{\vec{w},b}$ in code in code? Here's one way:

# Vectorization

Consider once again our model:

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

How do we implement the computation of $f_{\vec{w},b}$ in code in code? Here's one way:

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

# Vectorization

Consider once again our model:

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

How do we implement the computation of $f_{\vec{w},b}$ in code in code? Here's one way:

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

Unfortunately, this is clunky,

# Vectorization

Consider once again our model:

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

How do we implement the computation of $f_{\vec{w},b}$ in code in code? Here's one way:

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

Unfortunately, this is clunky, verbose,

# Vectorization

Consider once again our model:

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

How do we implement the computation of $f_{\vec{w},b}$ in code in code? Here's one way:

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

Unfortunately, this is clunky, verbose, brittle

# Vectorization

Consider once again our model:

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

How do we implement the computation of $f_{\vec{w},b}$ in code in code? Here's one way:

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

Unfortunately, this is clunky, verbose, brittle (we have to change code every time the number of features changes),

# Vectorization

Consider once again our model:

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

How do we implement the computation of $f_{\vec{w},b}$ in code in code? Here's one way:

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

Unfortunately, this is clunky, verbose, brittle (we have to change code every time the number of features changes), and slow.

# Vectorization

Let's reconsider our model, this time using summation syntax:

# Vectorization

Let's reconsider our model, this time using summation syntax:

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

# Vectorization

Let's reconsider our model, this time using summation syntax:

$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$
$$= \sum_{j=1}^{n} w_i x_i + b$$

# Vectorization

Let's reconsider our model, this time using summation syntax:

$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$
$$= \sum_{j=1}^{n} w_i x_i + b$$

This may inspire us to use a for-loop:

# Vectorization

Let's reconsider our model, this time using summation syntax:

$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$
$$= \sum_{j=1}^{n} w_i x_i + b$$

This may inspire us to use a for-loop:

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

# Vectorization

Let's reconsider our model, this time using summation syntax:

$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$
$$= \sum_{j=1}^{n} w_i x_i + b$$

This may inspire us to use a for-loop:

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

This is an improvement, but it still computes the terms of $f$ in series (and thus, is relatively slow).

# Vectorization

Let's reconsider our model, yet again, but this time using vectorization syntax:

# Vectorization

Let's reconsider our model, yet again, but this time using vectorization syntax:

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

# Vectorization

Let's reconsider our model, yet again, but this time using vectorization syntax:

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

# Vectorization

Let's reconsider our model, yet again, but this time using vectorization syntax:

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

This should inspire us to use vectorization syntax in our code:

# Vectorization

Let's reconsider our model, yet again, but this time using vectorization syntax:

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

This should inspire us to use vectorization syntax in our code:

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

# Vectorization

Let's reconsider our model, yet again, but this time using vectorization syntax:

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

This should inspire us to use vectorization syntax in our code:

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

One line of code!

# Vectorization

Let's reconsider our model, yet again, but this time using vectorization syntax:

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

This should inspire us to use vectorization syntax in our code:

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

One line of code! More importantly, this is much faster.

# Vectorization

Let's reconsider our model, yet again, but this time using vectorization syntax:

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

This should inspire us to use vectorization syntax in our code:

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

One line of code! More importantly, this is much faster. Why?

# Vectorization

Consider what happens when Python computes $f$ using a for-loop:

# Vectorization

Consider what happens when Python computes $f$ using a for-loop:

```
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

Consider what happens when Python computes $f$ using a for-loop:

```
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]
```

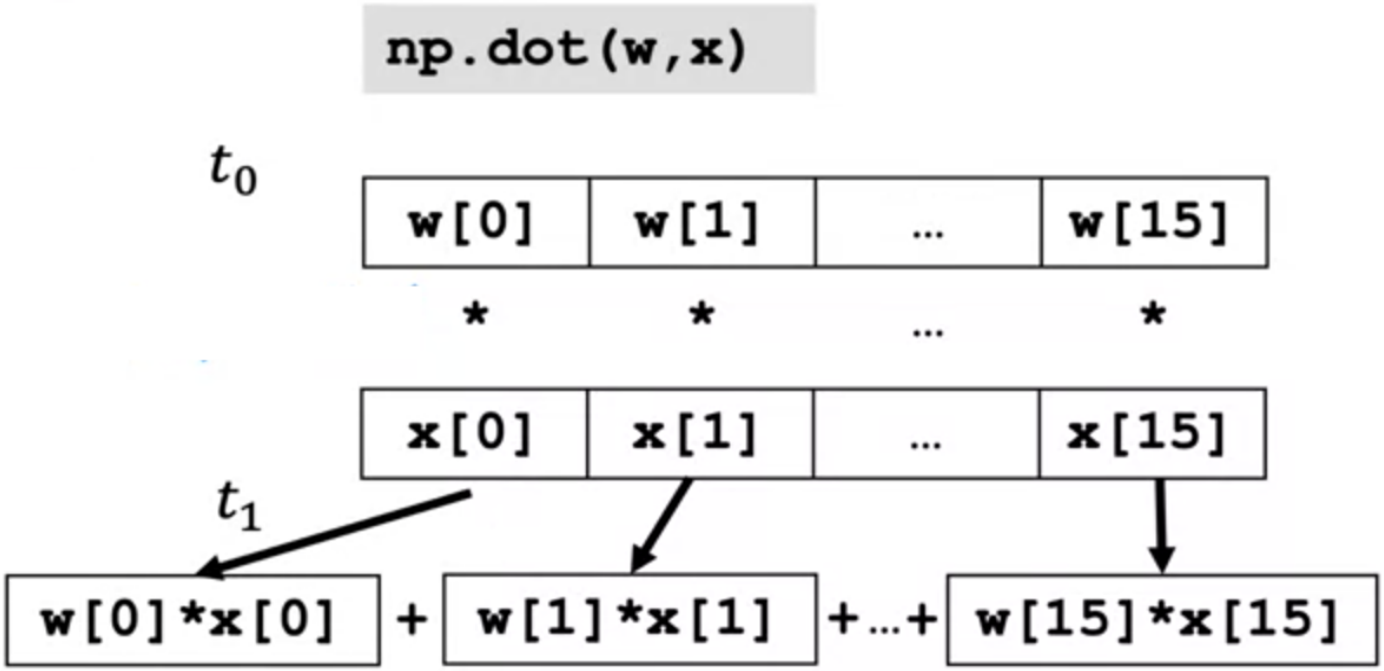
Each term is computed consecutively, (in series).

# Vectorization

Now, consider what happens when Python computes $f$ using vectorization:

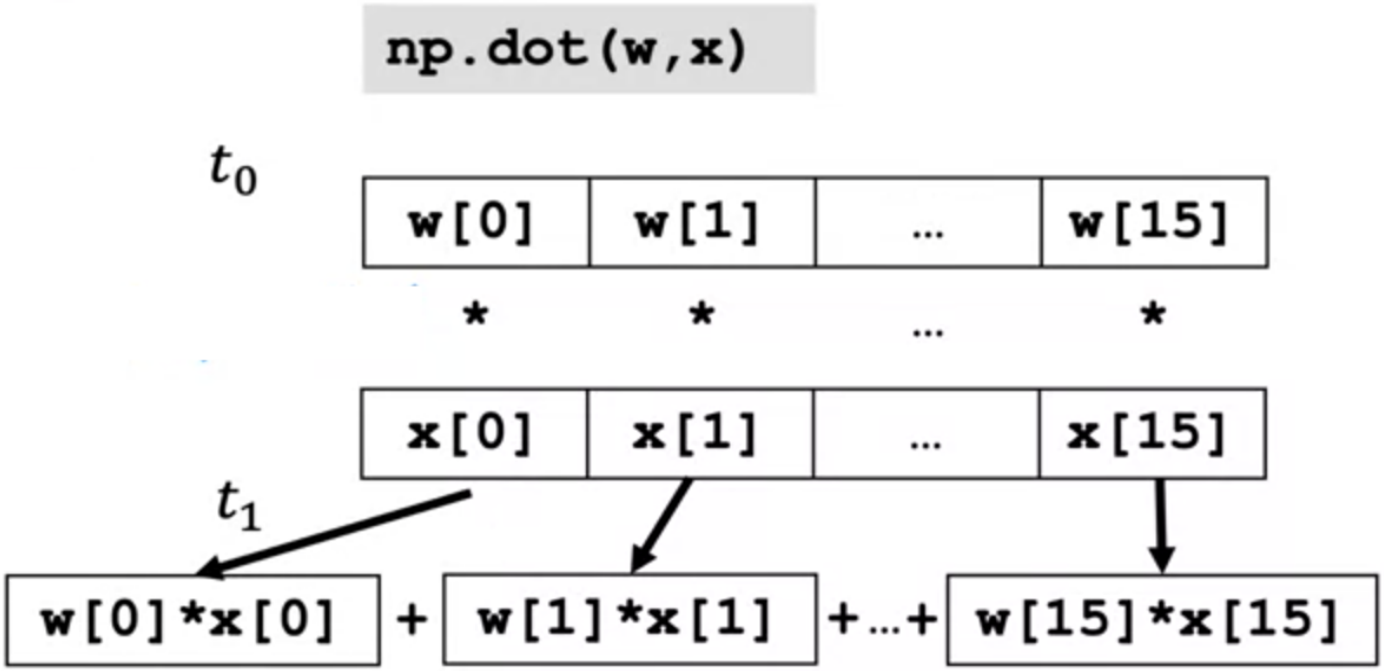
# Vectorization

Now, consider what happens when Python computes $f$ using vectorization:



# Vectorization

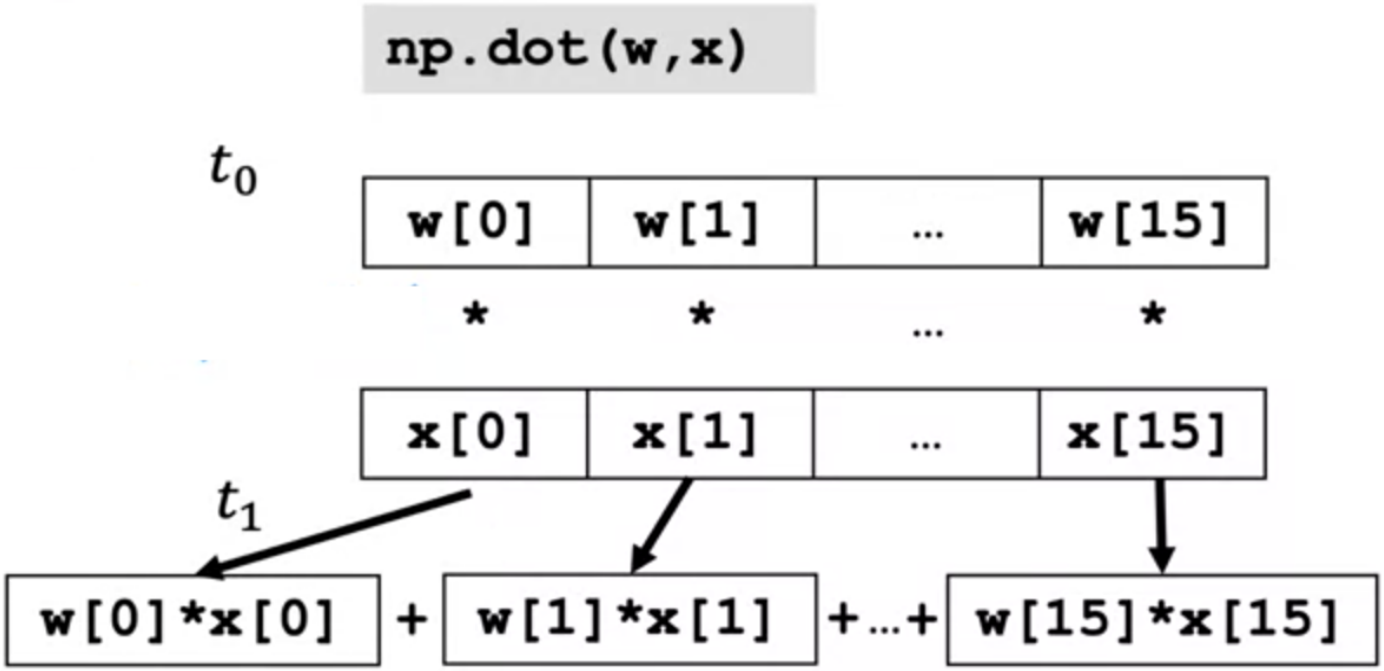
Now, consider what happens when Python computes $f$ using vectorization:




Terms are computed in parallel,

# Vectorization

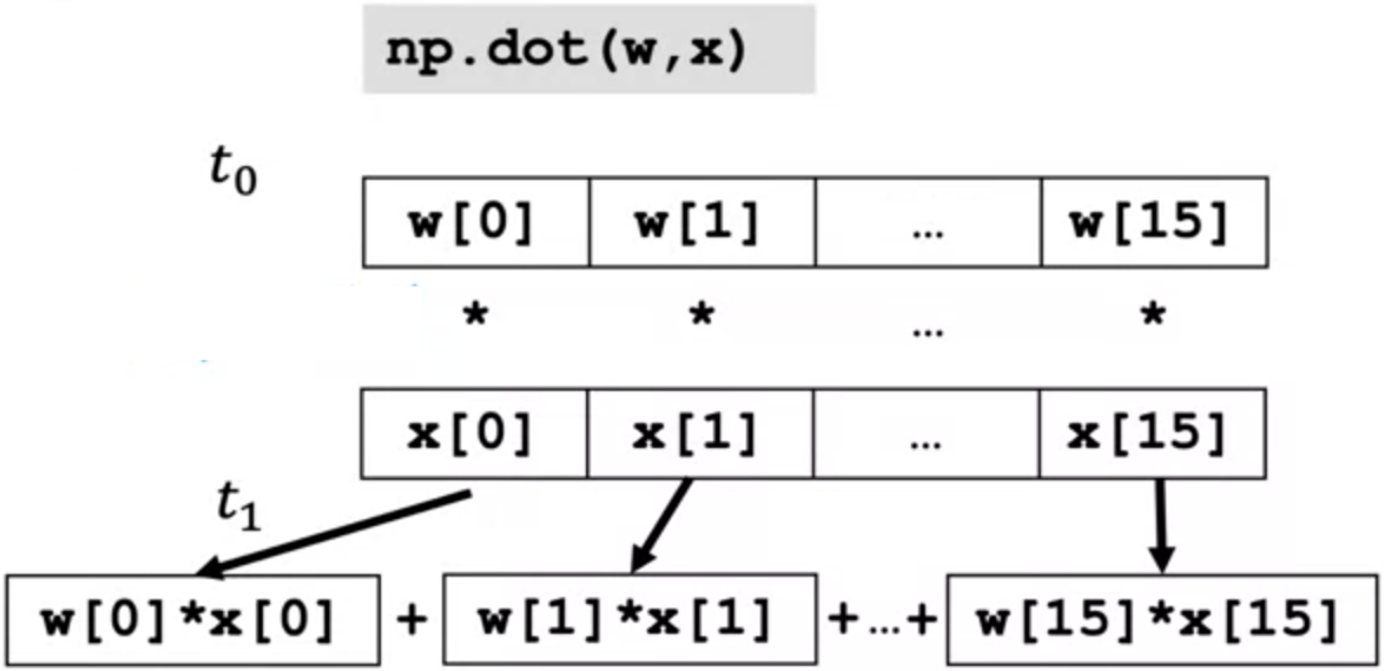
Now, consider what happens when Python computes $f$ using vectorization:




Terms are computed in parallel, using parallel processing features of the underlying computer chip,

# Vectorization

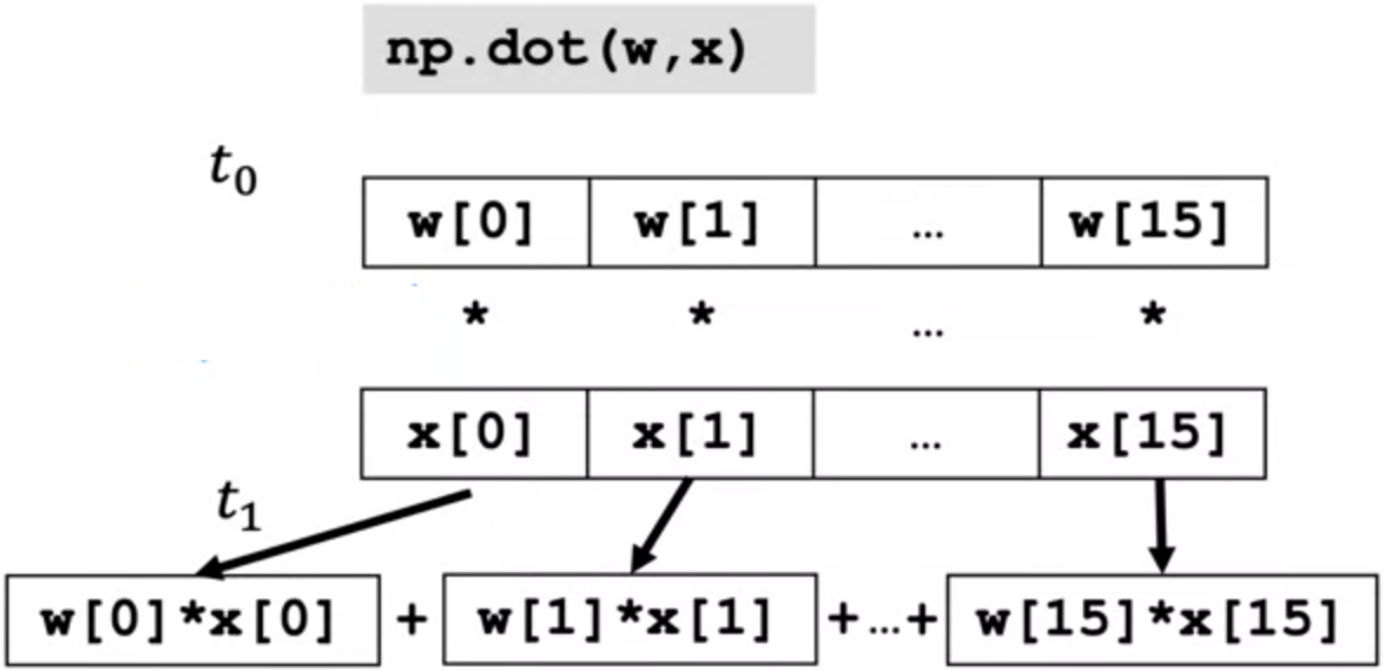
Now, consider what happens when Python computes $f$ using vectorization:




Terms are computed in parallel, using parallel processing features of the underlying computer chip, leading to a dramatic improvement in performance.

# Vectorization

Now, consider what happens when Python computes $f$ using vectorization:




Terms are computed in parallel, using parallel processing features of the underlying computer chip, leading to a dramatic improvement in performance. (Scales well to large datasets.)

# Vectorization

How do we apply apply parallelization in our gradient descent algorithm for multiple regression?

# Vectorization

How do we apply apply parallelization in our gradient descent algorithm for multiple regression?

- Suppose we have the following:
  - $\vec{w} = (w_1, w_2, \cdots w_{16})$

# Vectorization

How do we apply apply parallelization in our gradient descent algorithm for multiple regression?

- Suppose we have the following:
  - $\vec{w} = (w_1, w_2, \cdots w_{16})$ - vector of $w$ parameters

# Vectorization

How do we apply apply parallelization in our gradient descent algorithm for multiple regression?

- Suppose we have the following:
  - $\vec{w} = (w_1, w_2, \cdots w_{16})$ - vector of $w$ parameters
  - $\vec{d} = (d_1, d_2, \cdots d_{16})$

# Vectorization

How do we apply apply parallelization in our gradient descent algorithm for multiple regression?

- Suppose we have the following:
  - $\vec{w} = (w_1, w_2, \cdots w_{16})$ - vector of $w$ parameters
  - $\vec{d} = (d_1, d_2, \cdots d_{16})$ - vector of partial derivatives wrt $w$'s

# Vectorization

How do we apply apply parallelization in our gradient descent algorithm for multiple regression?

- Suppose we have the following:
  - $\vec{w} = (w_1, w_2, \cdots w_{16})$ - vector of $w$ parameters
  - $\vec{d} = (d_1, d_2, \cdots d_{16})$ - vector of partial derivatives wrt $w$'s
- We store these vectors in numpy arrays:

# Vectorization

How do we apply apply parallelization in our gradient descent algorithm for multiple regression?

- Suppose we have the following:
  - $\vec{w} = (w_1, w_2, \cdots w_{16})$ - vector of $w$ parameters
  - $\vec{d} = (d_1, d_2, \cdots d_{16})$ - vector of partial derivatives wrt $w$'s
- We store these vectors in numpy arrays:

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

# Vectorization

- Non-vectorized approach using a for-loop (poor performance):

$$w_1 = w_1 - 0.1d_1$$
$$w_2 = w_2 - 0.1d_2$$
$$\vdots$$
$$w_{16} = w_{16} - 0.1d_{16}$$

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

# Vectorization

- Vectorized approach:

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

# Vectorization

- Vectorized approach:

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

  - Much better performance

# Vectorization

- Vectorized approach:

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

  - Much better performance
  - Less code

# Vectorization

- Vectorized approach:

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

  - Much better performance
  - Less code
  - Flexible - handles vectors of different sizes with not change to code