Linear Regression

Outline

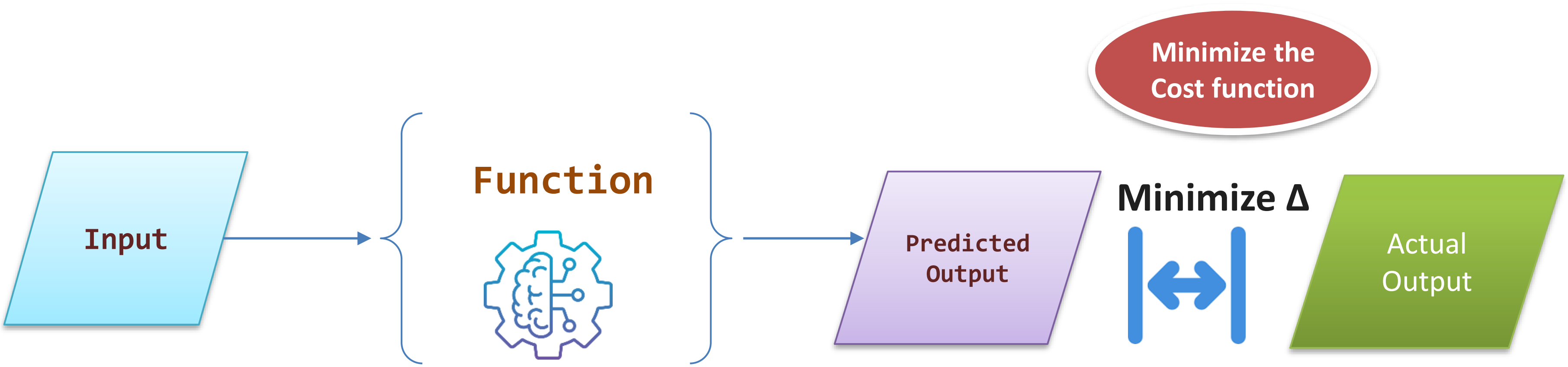
- Linear Regression with One Variable
- Multiple Linear Regression
- Polynomial Regression
- Regression Practical Considerations

Some of slides are based on the slides from
https://www.deeplearning.ai/courses/machine-learning-specialization/

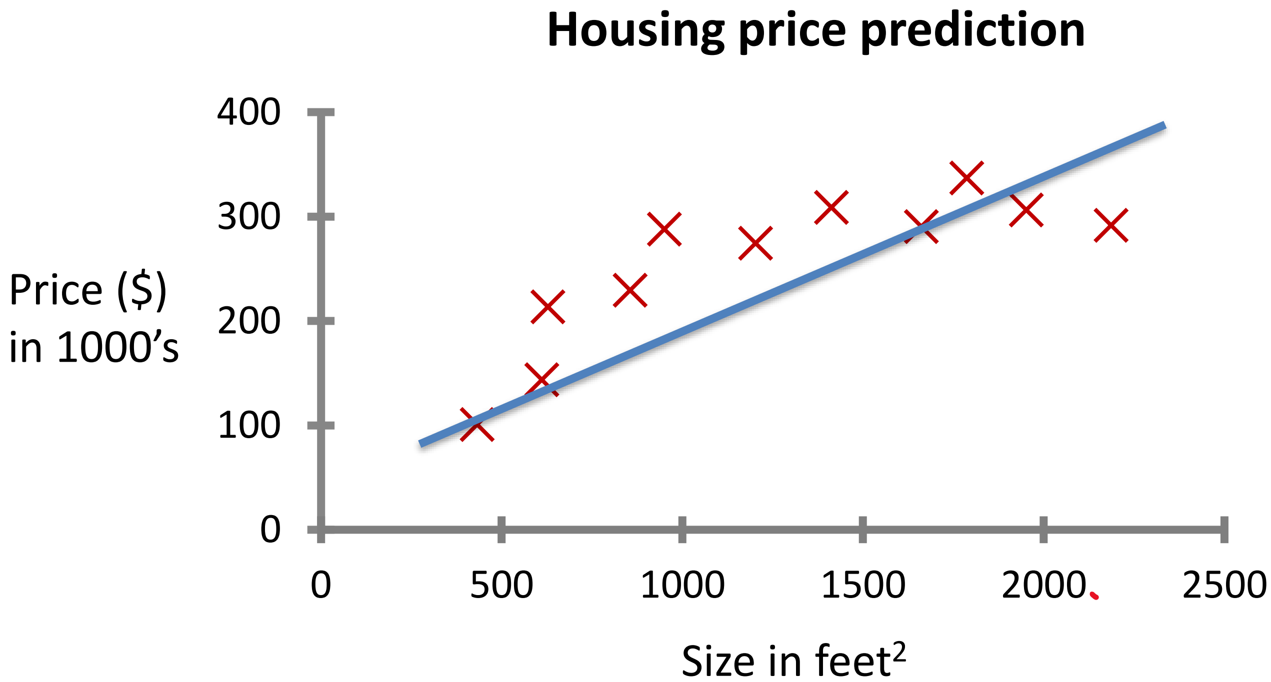
Linear Regression with One Variable

ML: learn a Function that minimizes the cost

- Start with random function parameters
- Repeat intelligent guessing/approximation of the Function parameters such that the difference between the Predicted Output the Actual Output is reduced
  - i.e., minimize a Cost function a.k.a loss, or error function



Linear Regression with One Variable



Linear regression with one variable aka **Univariate linear regression**

- **<u>Regression</u>**: Predict continuous output value (e.g., price)
- Linear regression is used to predict the value of a variable based on the value of another variable(s)
- Linear regression **fits** a straight line that minimizes the discrepancies between predicted and actual output values

Training set of housing prices

Size in feet^2 (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Notation:

m = Number of training examples

x = "input" variable / features

y = "output" variable / "target" variable

$x^{(1)} = 2104$

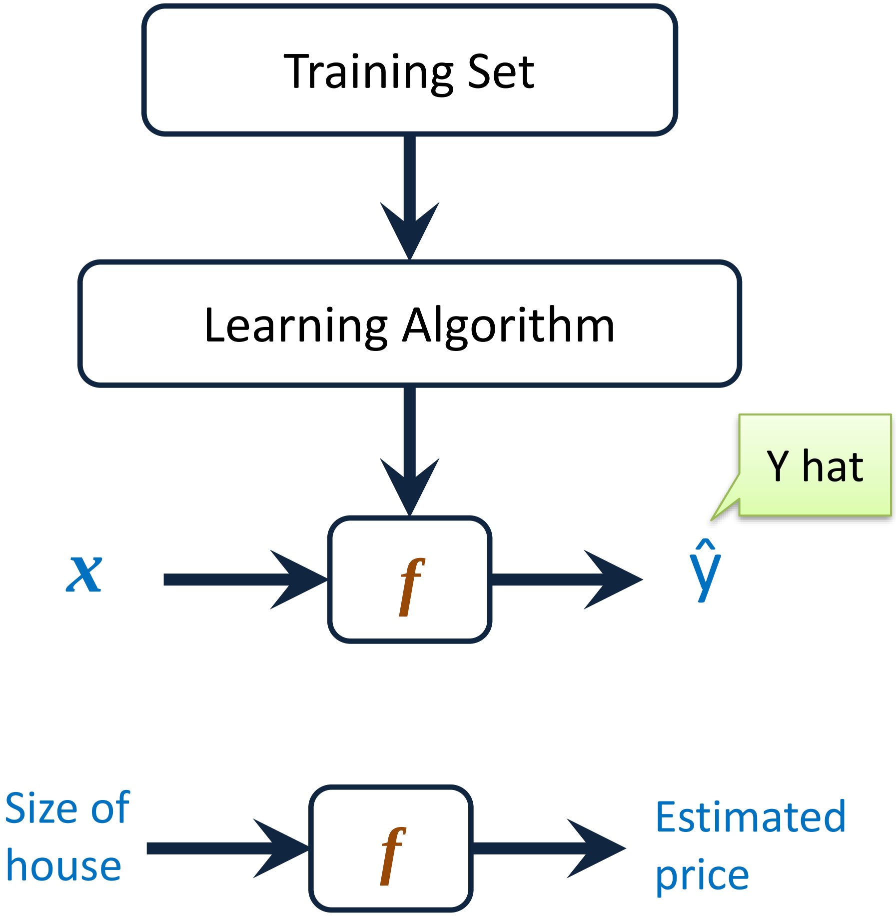
$x^{(2)} = 1416$

$y^{(1)} = 460$

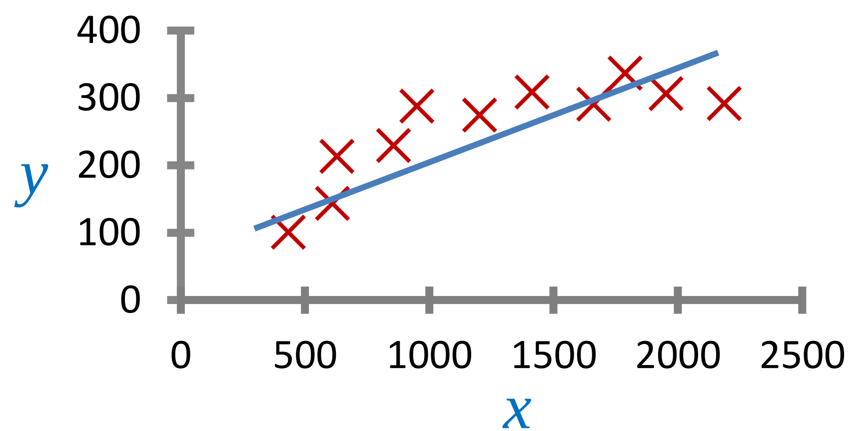
$(x^{(i)}, y^{(i)})$ – the i^{th} training example

How do we represent *f* ?

$$f(x) = wx + b$$



w, b are parameters (coefficients) to learn from the training set

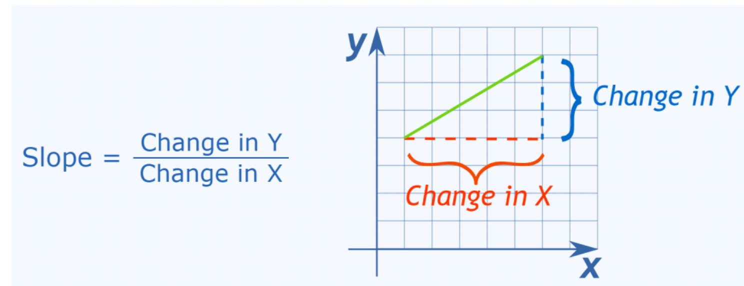


- Given a training set, **learn a function** *f* so that *f(x)* is a “good” predictor for the corresponding value of y
- Find w, b parameters that **minimize** the error between predicted and actual values (i.e., minimize $\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$ for all dataset instances)

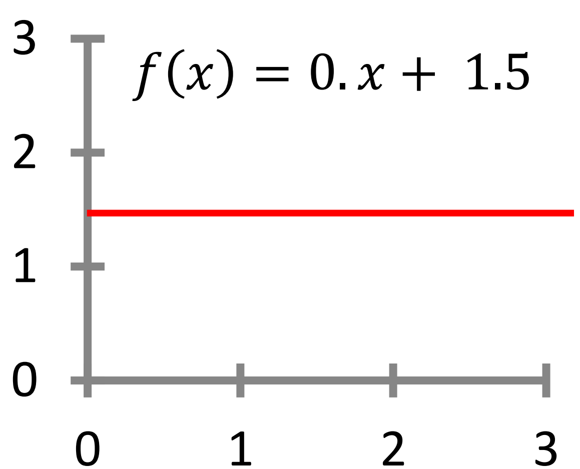
Univariate Linear Regression - Model Representation

$$\boldsymbol{f(x) = \color{blue}{w}x + \color{blue}{b}}$$

- $\boldsymbol{w}$ is the slope of the line
- $\boldsymbol{b}$ is the y-intercept of the line

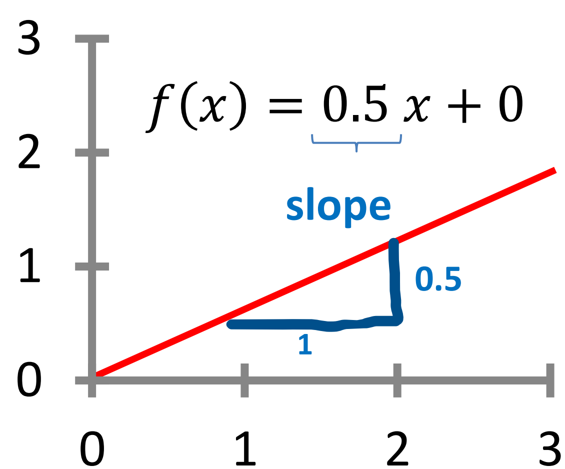


How to choose $\boldsymbol{w}$ and $\boldsymbol{b}$?



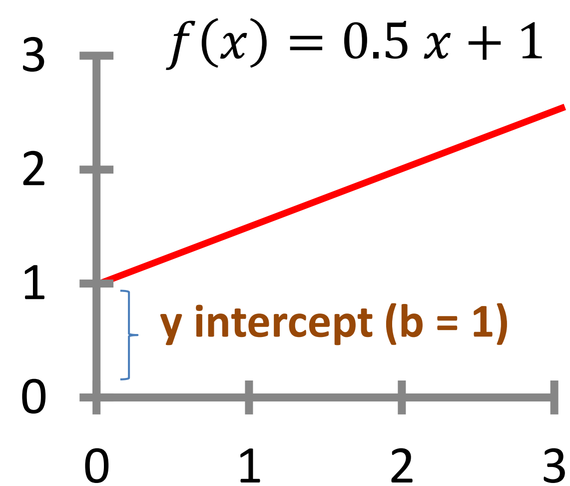
$w = 0$

$b = 1.5$



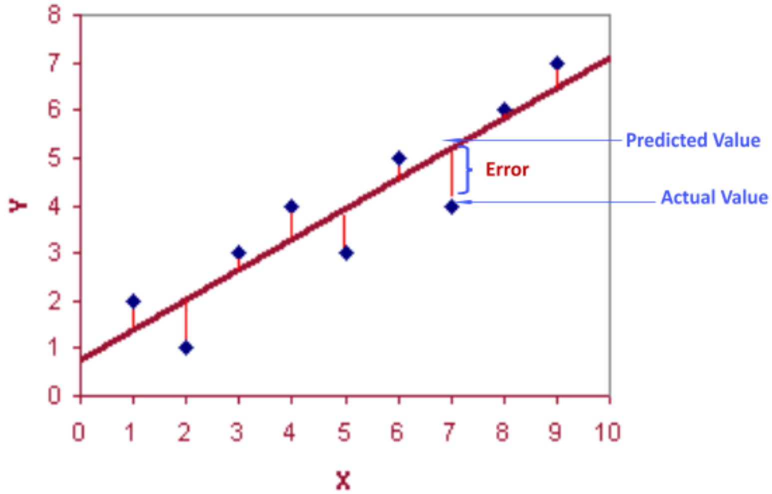
$w = 0.5$

$b = 0$



$w = 0.5$

$b = 1$



Idea: Find $\boldsymbol{w}$ and $\boldsymbol{b}$ so that $f(x)$ is close to y for our training examples (x, y)

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

Cost (mean squared error)
Function:

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

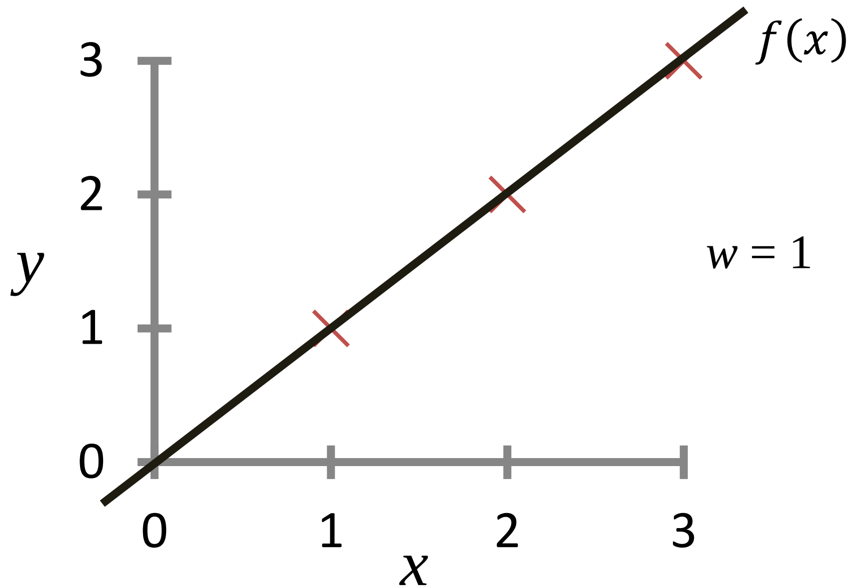
Goal: $\underset{w,\,b}{\text{minimize}}\ J(w,b)$

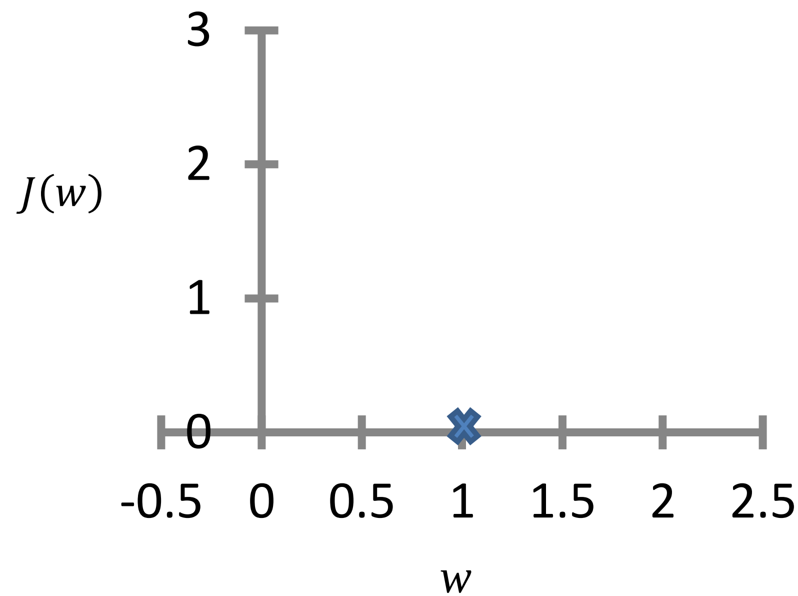
With **m** = number of training examples

Visualizing the Cost function

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}$$

For simplicity, let us assume our optimization objective is to $\underset{w,b}{\text{minimize}}\, \boldsymbol{J}(\boldsymbol{w})$, thus, b = 0





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

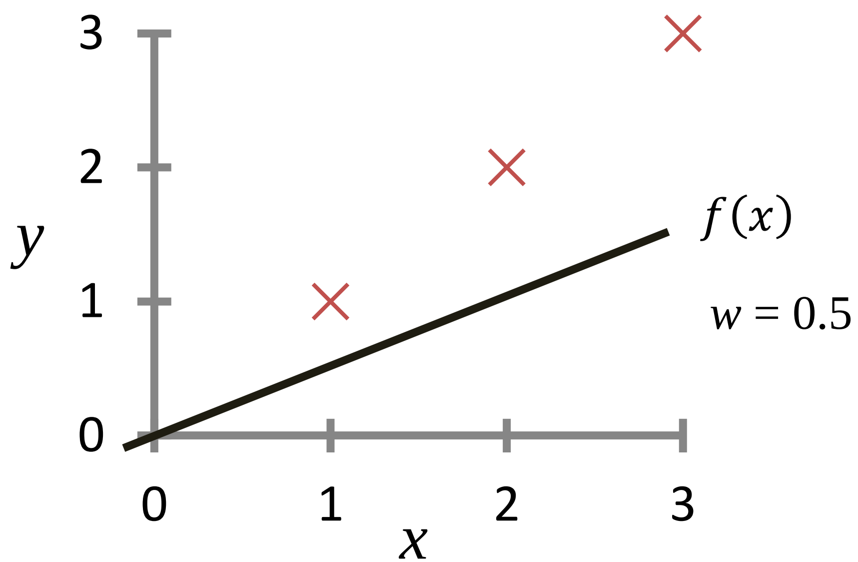
$$= \frac{1}{2m}(0^2 + 0^2 + 0^2) = 0$$

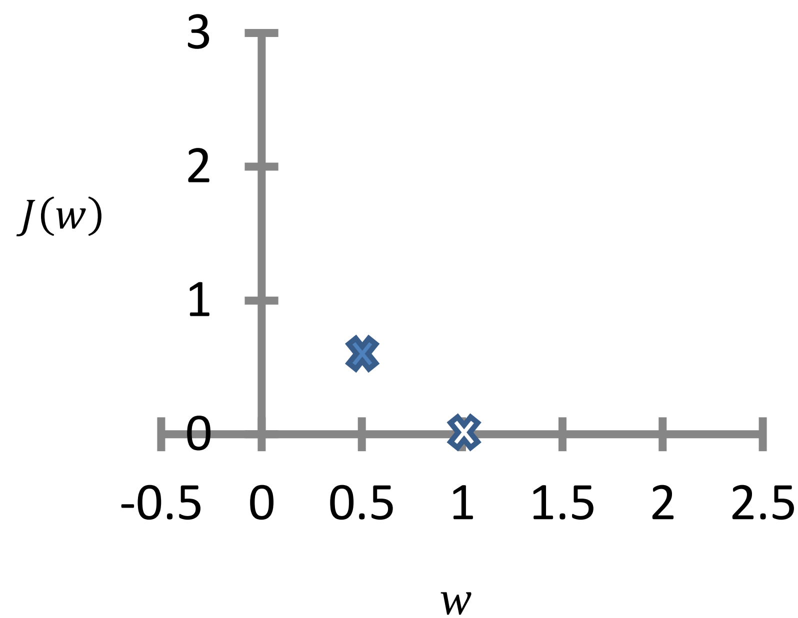
slides**regression.xlsx**

Visualizing the Cost function

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}$$

For simplicity, let us assume our optimization objective is to minimize$_{w,b}$ $\boldsymbol{J}(\boldsymbol{w})$, thus, b = 0





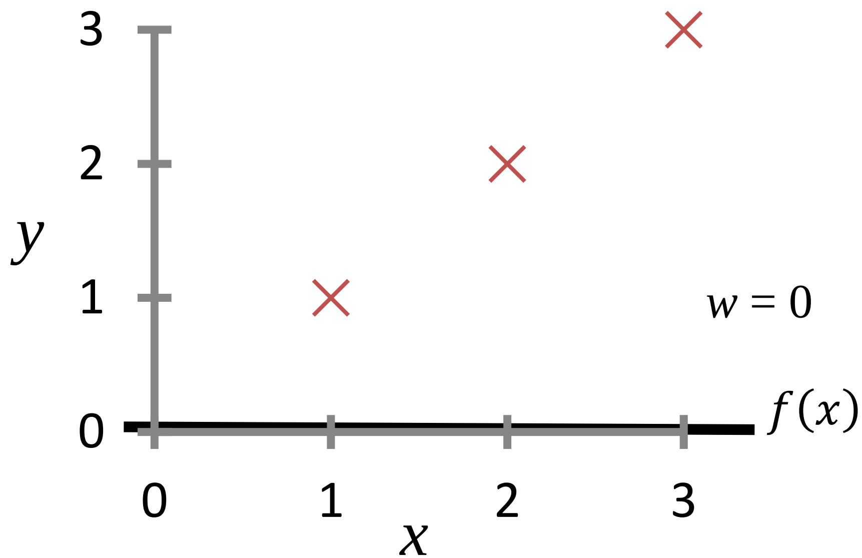
$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f\left(x^{(i)}\right) - y^{(i)})^2$$

$$= \frac{1}{2m}((0.5 - 1)^2 + (1\text{-}2)^2 + (1.5\text{-}3)^2)$$

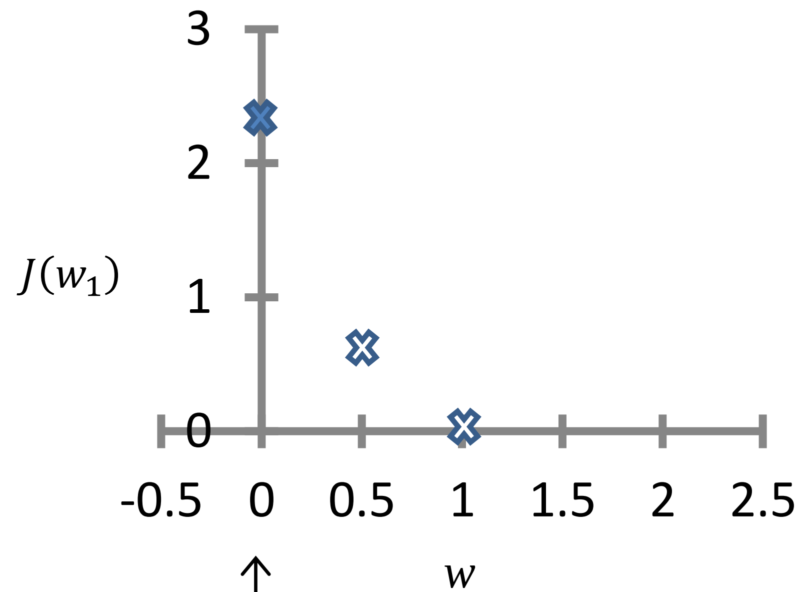
$$= \frac{1}{2\times 3}(3.5) = \frac{3.5}{6} = 0.58$$

Visualizing the Cost function

$$f(x) = wx$$



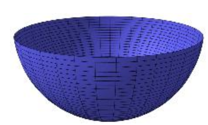
For simplicity, let us assume our optimization objective is to $\underset{w,b}{\text{minimize}}\, J(w)$, thus, b = 0



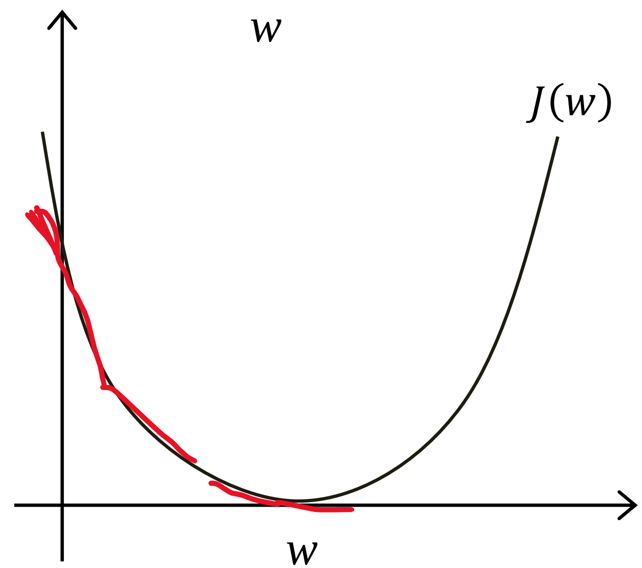
$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2m}(1^2 + 2^2 + 3^2)$$

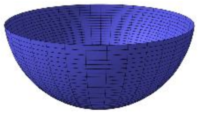
$$= \frac{1}{2\times 3}(14) = \frac{14}{6} = 2.3$$



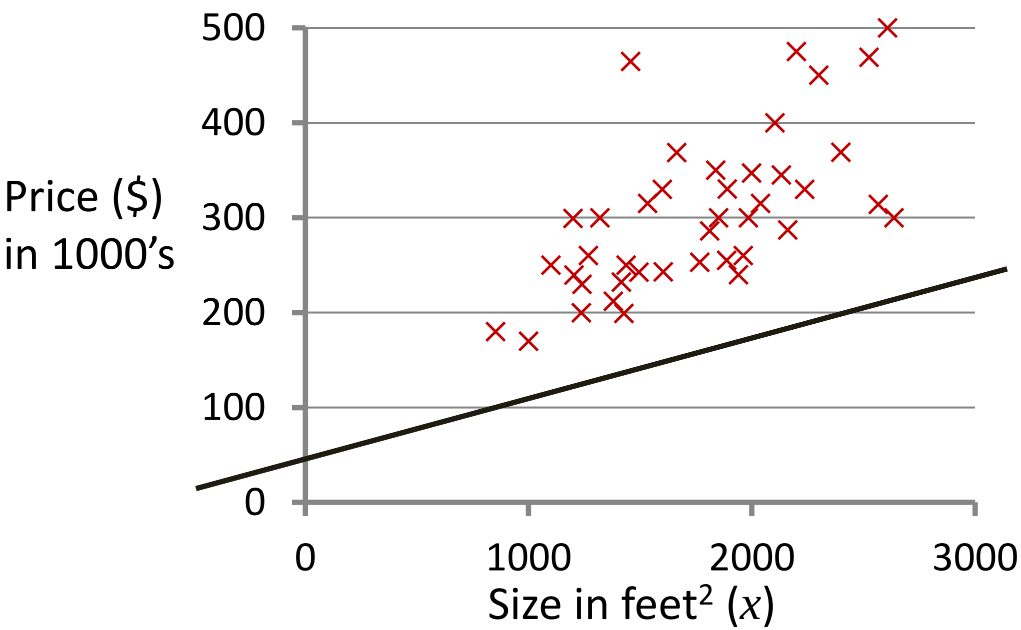
J(w) cost function
has bowl shape
Convex function



Visualizing the Cost function

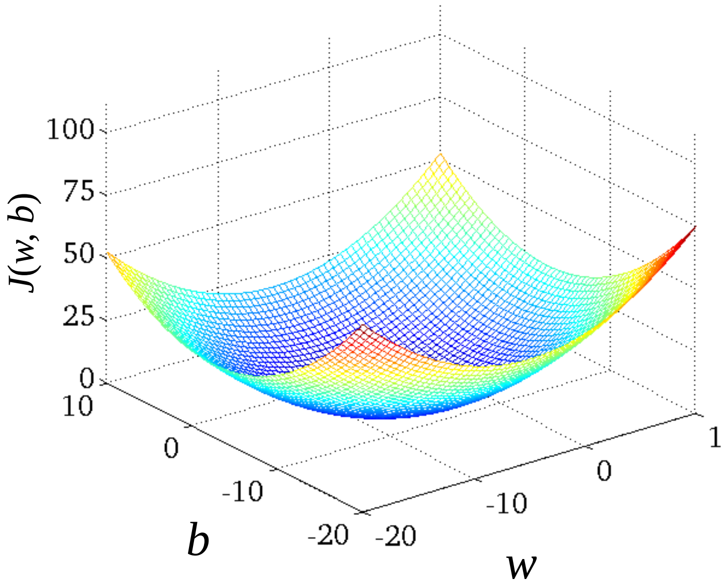


$$f(x) = wx + b$$



$f(x) = 0.06x + 50$

$J(w, b)$

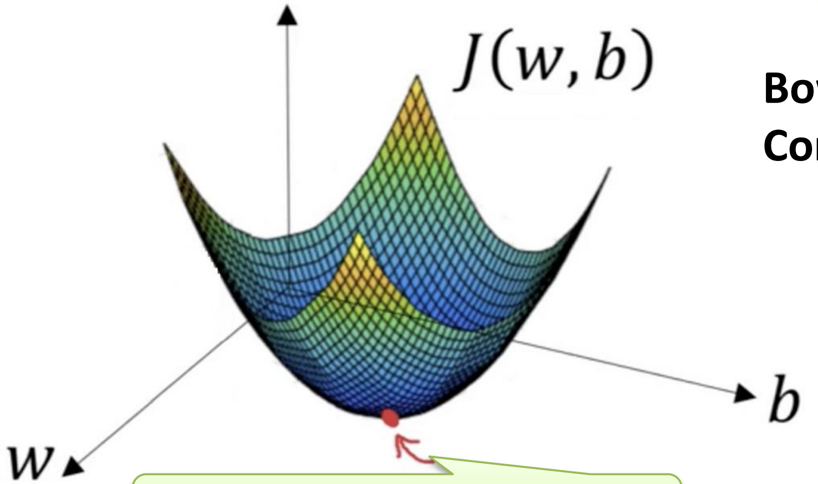


J(w,b) cost function has bowl shape

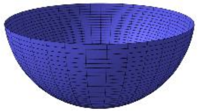
Convex function

- The fact that the cost function squares the error ensures that the 'error surface' is **convex** like a soup bowl.
- It will always have a minimum that can be reached by following the **gradient** (i.e., the slope)
- Minimizing the cost function yields optimal values of **w** and ***b***

`08.regression\02_cost_function.py`



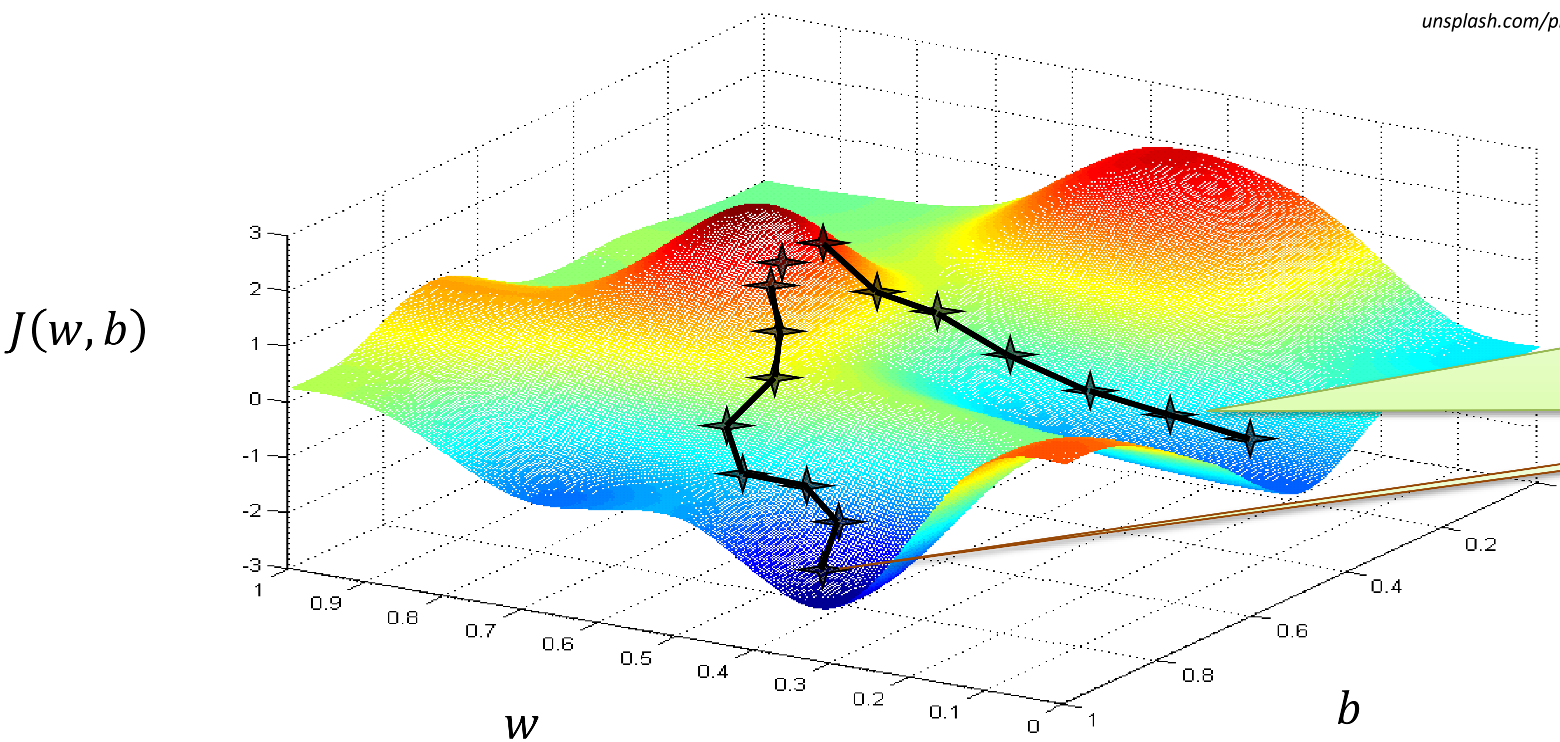
MSE has only 1 global minimum



Bowl shape
Convex function

Gradient Descent can be used to find the optimal w and b that minimizes that cost function

unsplash.com/photos/3m6vbzY69s4



Other cost functions may have multiple local minima

Gradient descent algorithm

Want to find **w** and ***b*** that minimize the cost function **J**

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

1. Initialize the values of **w** and ***b*** to some arbitrary values (say 0, 0)
2. Calculate the predicted values of ***y*** using the current values of **w** and ***b***
3. Calculate the gradients of the cost function with respect to **w** and ***b***
4. Update the values of **w** and ***b*** using the gradients and a learning rate

Learning Rate

Derivative of the Cost Function w.r.t w

$$w = w - \alpha \boxed{\frac{d}{dw} J(w,b)}$$

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

5. Repeat steps 2-4 until convergence (i.e., until the cost function converges to a minimum)

Gradient descent algorithm

Gradient descent utilizes the partial derivative of the cost function with respect to **w** and $\boldsymbol{b}$ to update **w** and $\boldsymbol{b}$ parameters

Repeat until convergence {

$$\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{\alpha} \underbrace{\frac{\mathbf{1}}{\boldsymbol{m}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{m}} \left(\hat{\mathbf{y}}^{(\boldsymbol{i})} - \boldsymbol{y}^{(\boldsymbol{i})} \right) . \ \boldsymbol{x}^{(\boldsymbol{i})}}_{\frac{\partial J}{\partial w}}$$

$$\boldsymbol{b} = \boldsymbol{b} - \boldsymbol{\alpha} \underbrace{\frac{\mathbf{1}}{\boldsymbol{m}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{m}} \left(\hat{\mathbf{y}}^{(\boldsymbol{i})} - \boldsymbol{y}^{(\boldsymbol{i})} \right)}_{\frac{\partial J}{\partial b}}$$

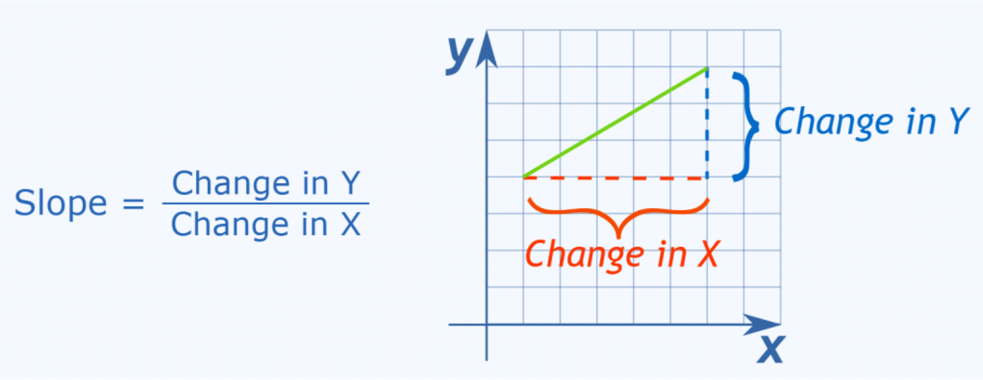
}

(simultaneously update $\boldsymbol{w}$ and $\boldsymbol{b}$)

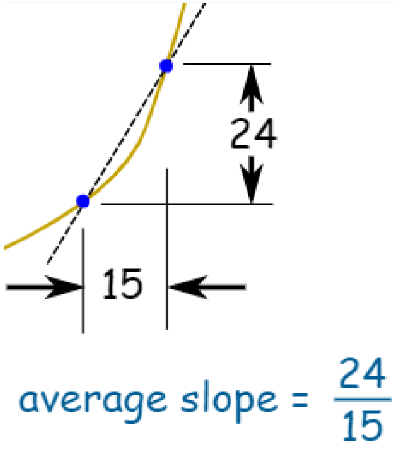
`08.regression\`
`04_gradient_descent_one_var.py`

Derivative 101

- Source https://www.mathsisfun.com/calculus/derivatives-introduction.html
- **Derivatives**: it is all about slope!

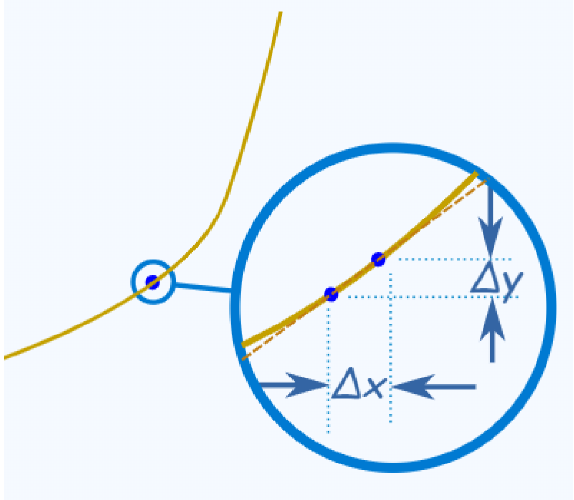


We can find an **average** slope between two points



Derivative = slope at a point

- Fill in this slope formula: $\dfrac{\Delta y}{\Delta x} = \dfrac{f(x+\Delta x) - f(x)}{\Delta x}$
- Simplify it as best we can
- Then make $\mathbf{\Delta x}$ shrink towards zero.



Example

The slope formula is: $\dfrac{f(x+\Delta x) - f(x)}{\Delta x}$

Use $f(x) = x^2$: $\dfrac{(x+\Delta x)^2 - x^2}{\Delta x}$

Expand $(x+\Delta x)^2$ to $x^2+2x\,\Delta x+(\Delta x)^2$: $\dfrac{x^2 + 2x\,\Delta x + (\Delta x)^2 - x^2}{\Delta x}$

Simplify (x^2 and $-x^2$ cancel): $\dfrac{2x\,\Delta x + (\Delta x)^2}{\Delta x}$

Simplify more (divide through by Δx): $2x + \Delta x$

Then, **as Δx heads towards 0** we get: $2x$

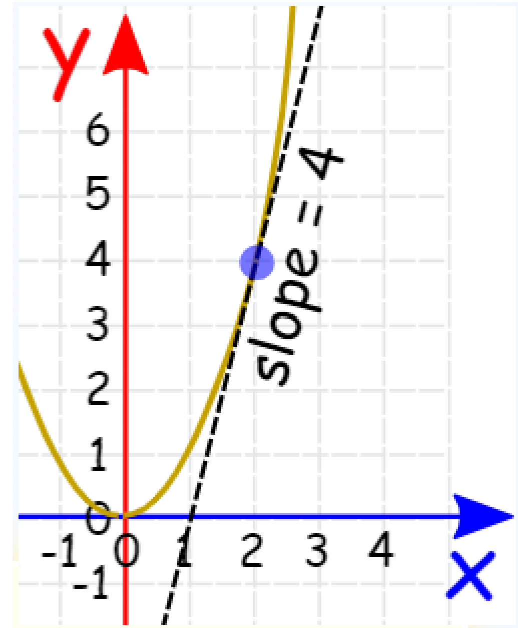
Result: the derivative of $\mathbf{x^2}$ is $\mathbf{2x}$

$$\frac{d}{dx}x^2 = 2x$$

In other words, the slope at x is 2x

Interpretation of Derivative

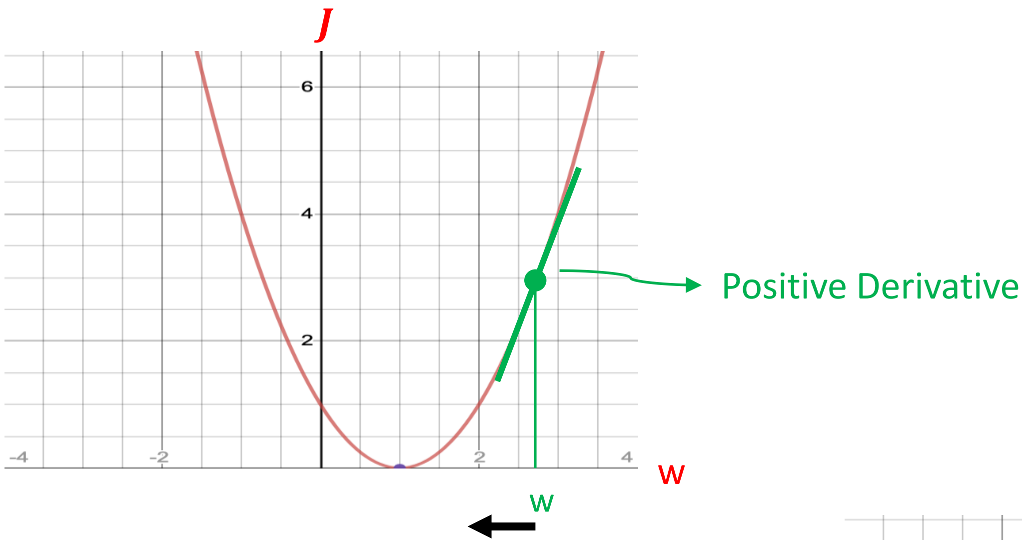
- So what does $\frac{d}{dx}x^2 = 2x$ mean?
- It means that, for the function x^2, the slope or "rate of change" at any point is **2x**
- So, when **x=2** the slope is **2x = 4**
- Or when **x=5** the slope is **2x = 10**, and so on



`08.regression\`
`03_gradient_decent_x_square_animation.py`

The Impact of Partial Derviative

- For simplicity, let us assume our optimization objective is to $\min\limits_{w,b} \boldsymbol{J(w)}$, thus, b = 0



Learing Rate

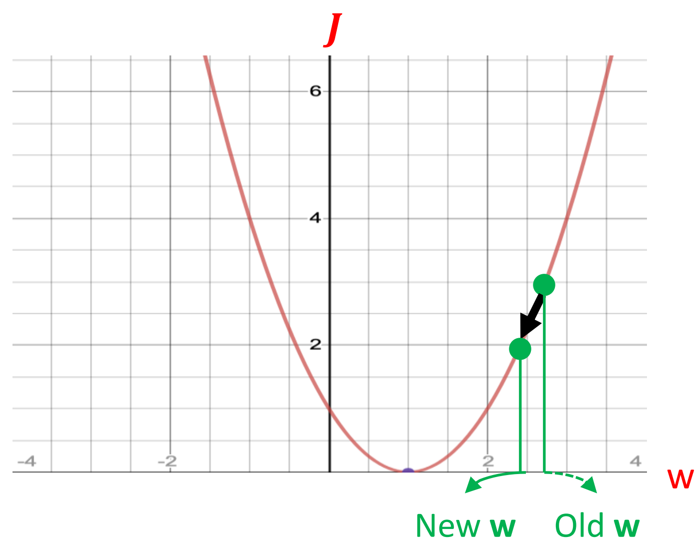
$$w = w - \alpha \frac{\boldsymbol{d}\,\boldsymbol{J(w)}}{\boldsymbol{d}w}$$

$$= \underbrace{w - \alpha\,(Positive\ Number)}_{\text{Decrease } \boldsymbol{w} \text{ by a certain value}}$$

repeat until convergence {

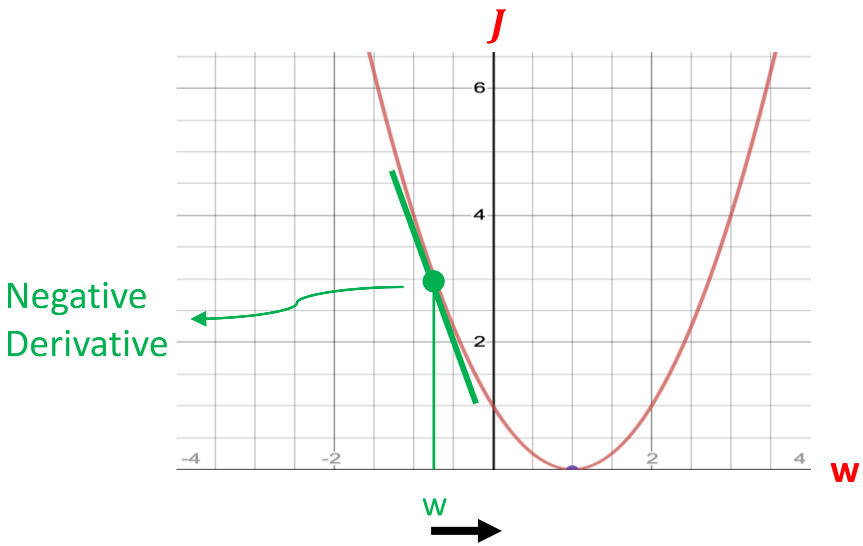
$$\text{w} = w - \alpha \frac{\boldsymbol{d}\,\boldsymbol{J(w)}}{\boldsymbol{d}w}$$

}



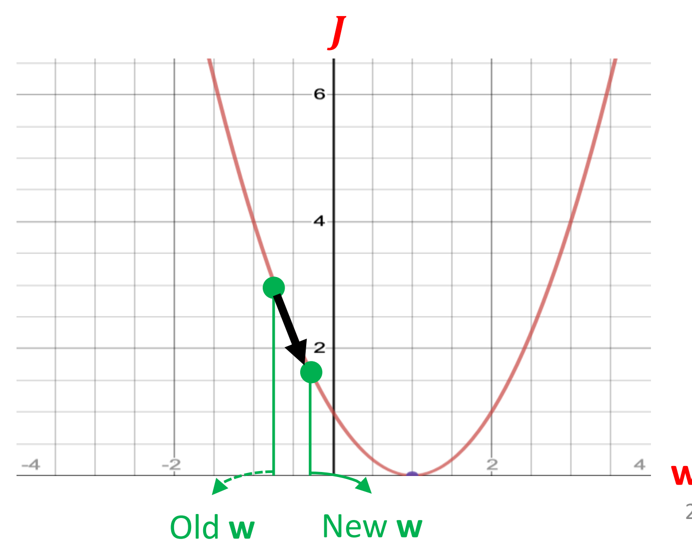
The Impact of Partial Derviative

- For simplicity, let us assume our optimization objective is to $\min\limits_{w,b} \boldsymbol{J}(\boldsymbol{w})$, thus, b = 0



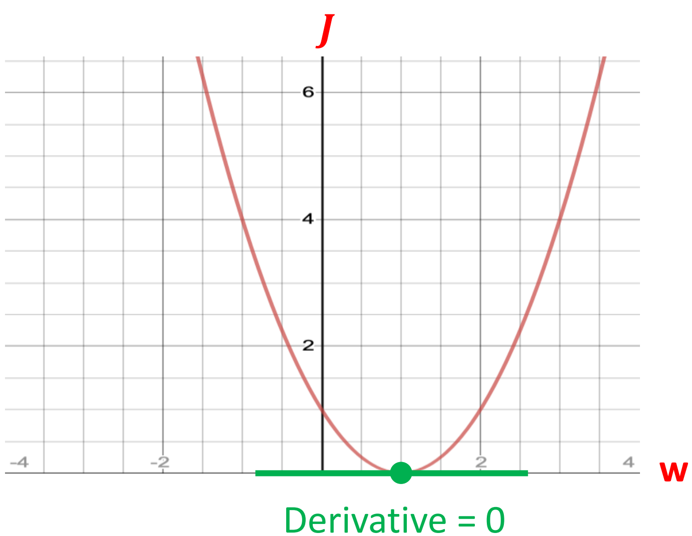
$$w = w - \alpha \frac{\boldsymbol{d}\, \boldsymbol{J}(\boldsymbol{w})}{\boldsymbol{d}w}$$

$$\underbrace{= \; w - \alpha \, (Negative\ Number)}_{\text{Increase } \boldsymbol{w} \text{ by a certain value}}$$



The Impact of Partial Derviative

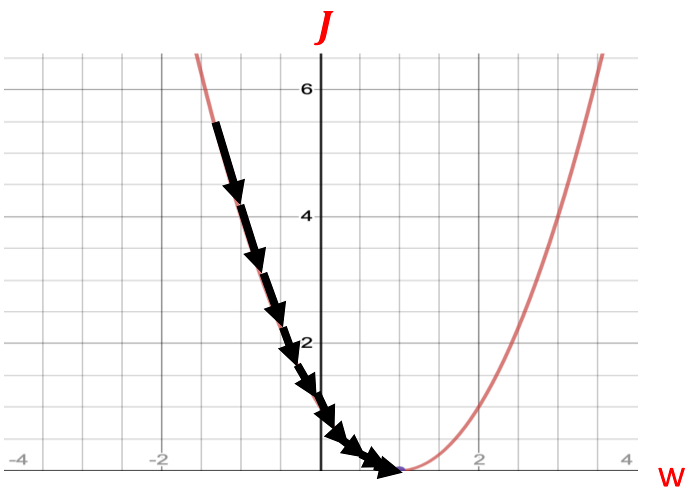
- For simplicity, let us assume our optimization objective is to $\underset{w,b}{\text{minimize}}\, \boldsymbol{J}(\boldsymbol{w})$, thus, b = 0



$$\underbrace{\begin{aligned} w &= w - \alpha \frac{\boldsymbol{d}\,\boldsymbol{J}(\boldsymbol{w})}{\boldsymbol{d}w} \\ &= w - \alpha\,(Zero) \end{aligned}}$$

w remains the same, hence, gradient descent *converges*

The Impact of Learning Rate

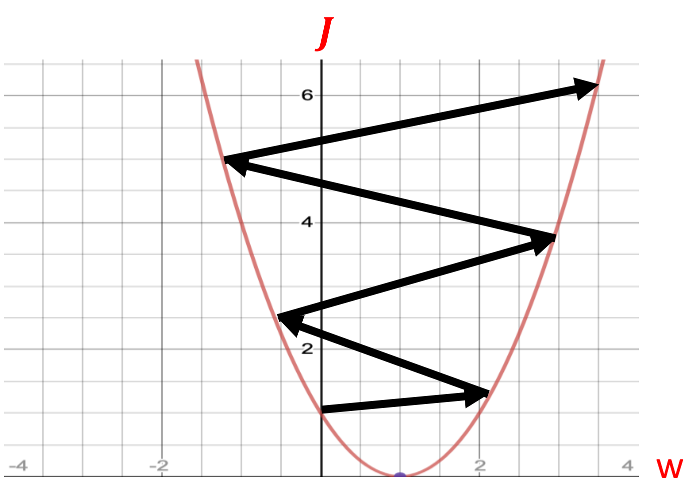


What happens if α is too small?

$$w = w - \alpha \frac{d\,J(w)}{dw}$$

$$\underbrace{= w - (Too\ Small\ Number)\ \frac{d\,J(w)}{dw}}$$

w changes only a tiny bit on each step, hence, gradient descent *will render slow (will take more time to converge)*



What happens if α is too large?

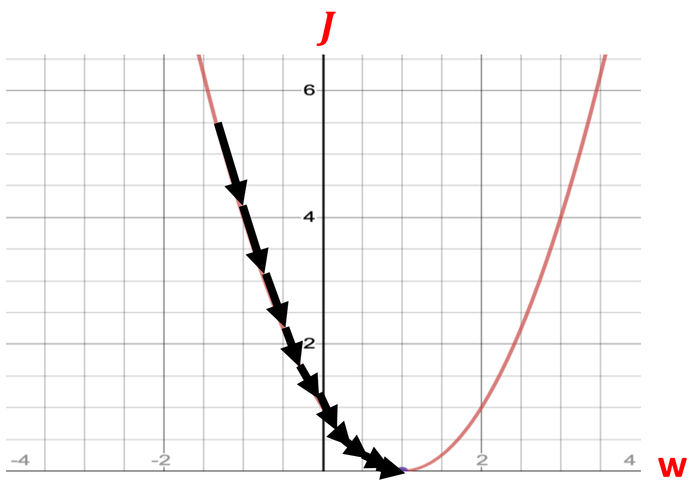
$$w = w - \alpha \frac{d\,J(w)}{dw}$$

$$\underbrace{= w - (Too\ Large\ Number)\ \frac{d\,J(w)}{dw}}$$

$\boldsymbol{w}$ changes a lot (and probably faster) on each step, hence, gradient descent *will potentially **overshoot the minimum** and, accordingly, fail to converge (or even diverge)*

The Impact of Learning Rate

We can set α between 0 and 1 (say, 0.1, or a little more or less, hence, not very small or very large)

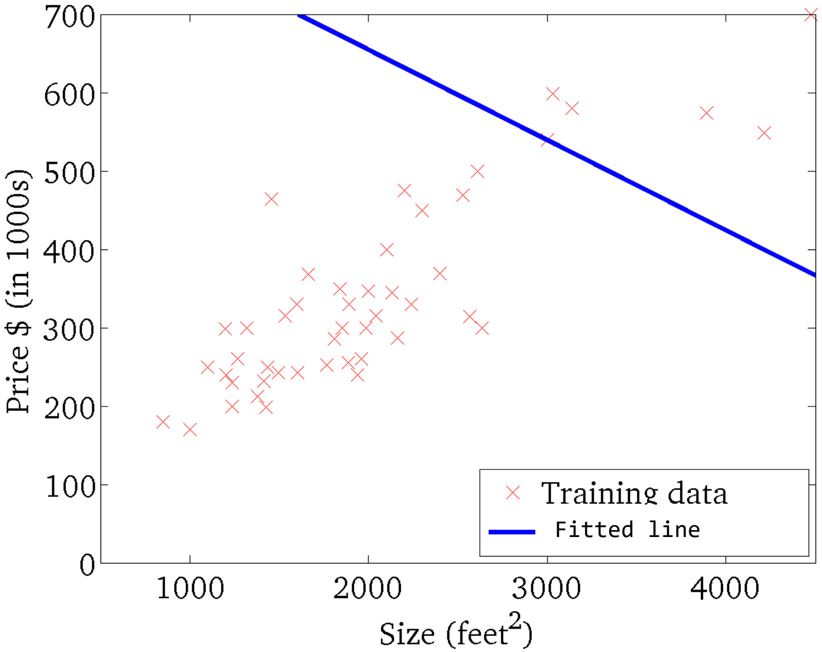


$$w = w - \alpha \frac{d\,J(w)}{dw}$$

$\boldsymbol{\alpha}$ remains fix. As we approach the minimum, gradient descent will automatically start taking smaller steps (i.e., $\boldsymbol{w}$ will start changing at a slower pace because the derivative will become less steep)

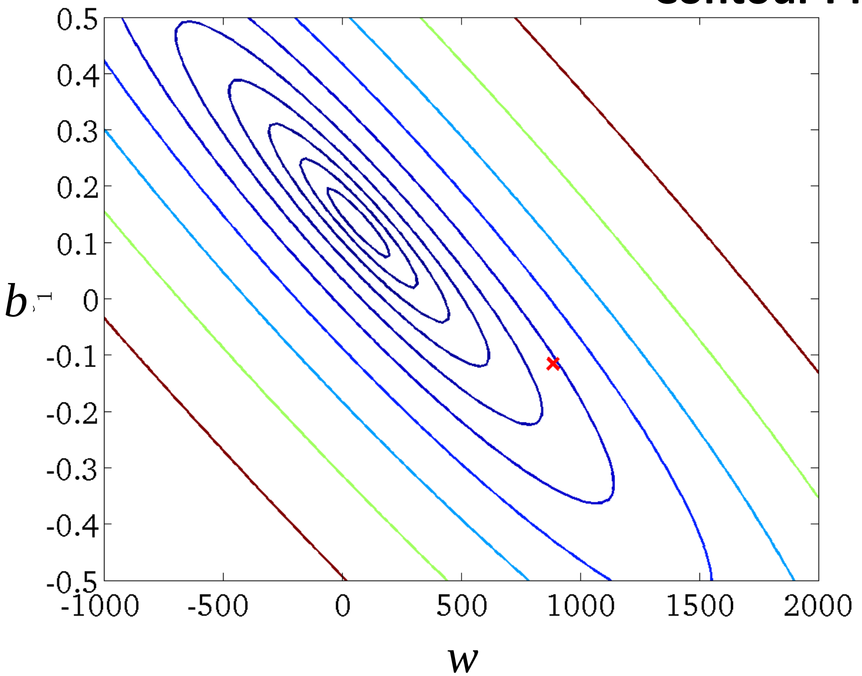
Visualizing gradient descent algorithm

$$f(x) = wx + b$$

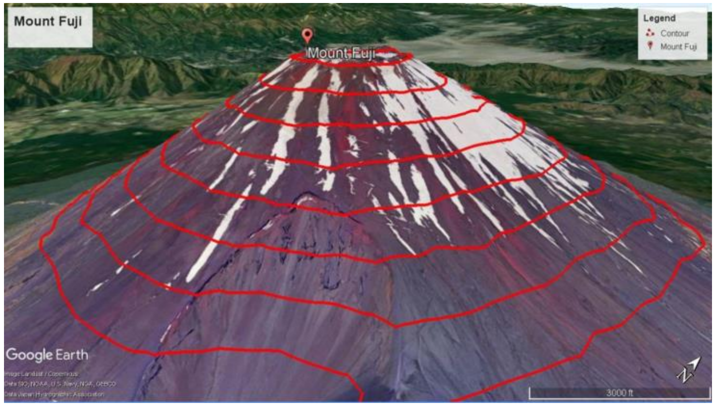


$$J(w, b)$$

Contour Plot



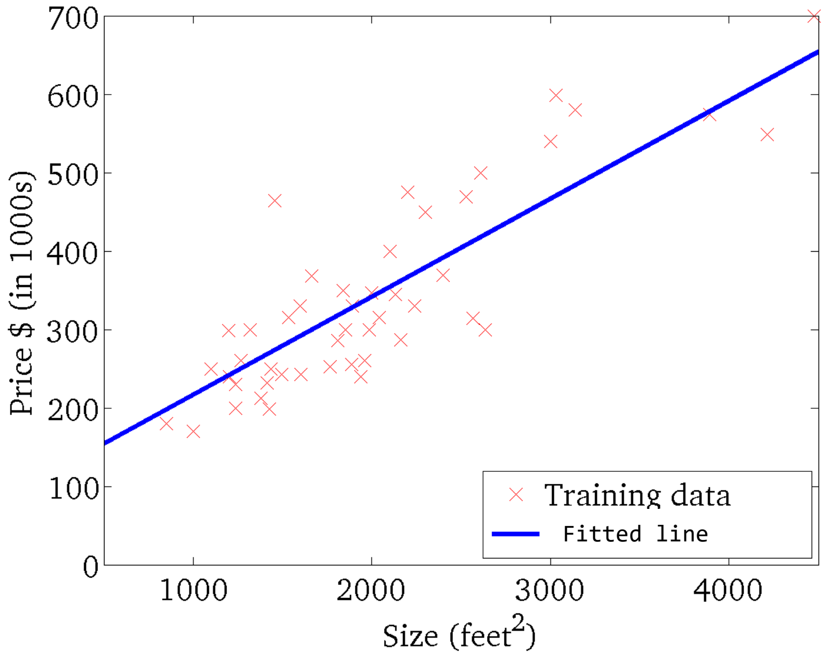
The optimal values of **w** and **b** are in the center of the inner most 'circle' of the Contour Plot



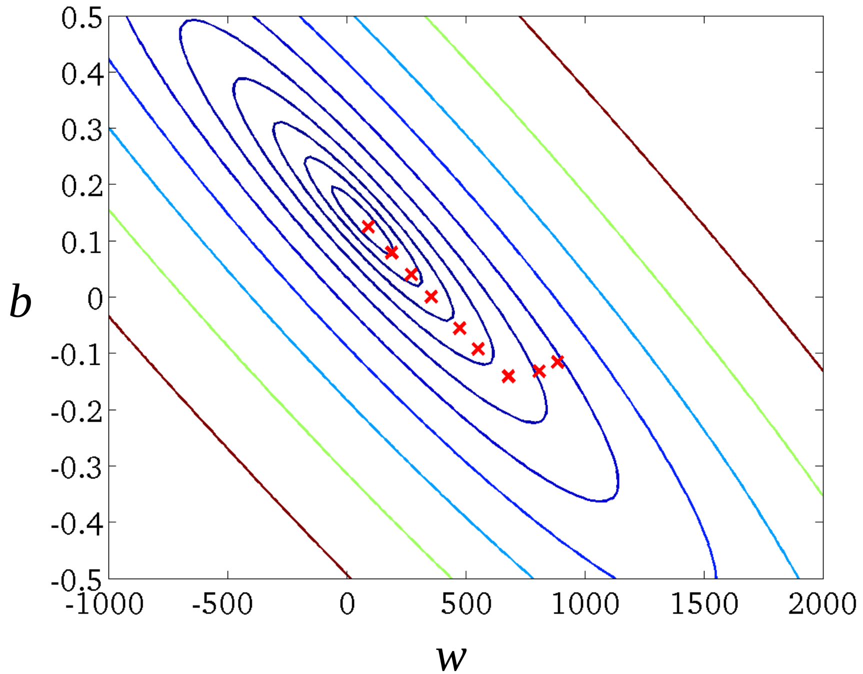
`08.regression\06_gradient_descent_visualization.py`

Visualizing gradient descent algorithm

$$f(x) = wx + b$$



$$J(w, b)$$

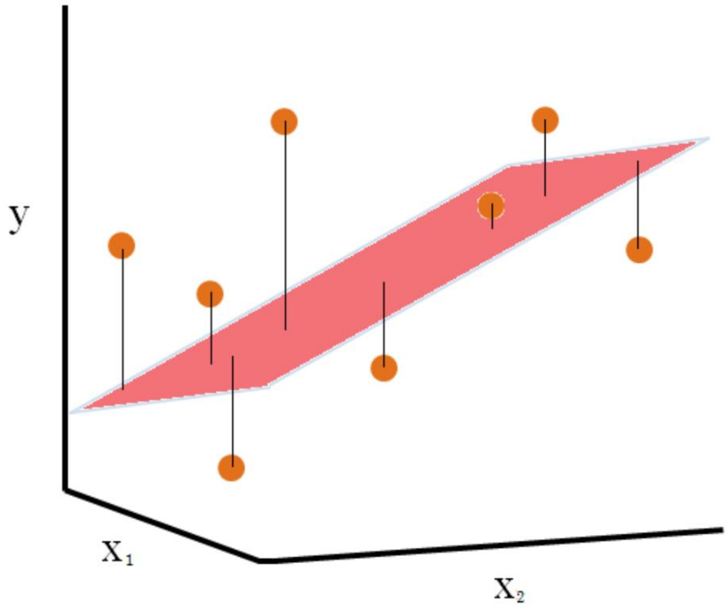


- The contour plot shows $J(w,b)$ over a range of w and b. The cost levels are represented by the rings
- The red crosses is the **path of gradient descent**. The path makes steady progress toward its goal. The initial steps are much larger than the steps near the goal.
- The optimal values of w and b are in the center of the inner most 'circle' of the Contour Plot

Different modes of gradient descent

- **Batch Gradient Descent (BGD)**: Each iteration of gradient descent uses all the training examples
  - It provides a precise estimate of the gradient but can be computationally expensive, especially for large datasets
- **Stochastic Gradient Descent (SGD):** only a random sample from the dataset is used to compute the gradient in each iteration
  - It is computationally more efficient
  - However, it introduces more noise in the parameter updates, leading to more oscillations in the convergence path
- **Mini-Batch Gradient Descent**: divides the dataset into small mini-batches and computes the gradient using a mini-batch in each iteration
  - This approach combines the efficiency of stochastic gradient descent with the stability of batch gradient descent

Multiple Linear Regression



Linear Regression with multiple variables

Multiple features (variables)

Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

m

$n=4$

w_1
w_2
w_3
w_4

Notation:

n = number of features

x_j = j^{th} feature

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

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

Multiple Linear Regression - Model Representation

Univariate Linear Regression: $f(x) = wx + b$

Multiple Linear Regression:

$$f(x) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

Example $f(x) = 0.1 x_1 + 4 x_2 + 10 x_3 + -2 x_4 + 80$

size, # bedrooms, #floors, years, base price

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

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

b is a number

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

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

dot product

Multiple Linear Regression - Model Representation

Parameters and features

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

b is a number

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

linear algebra: count from 1

NumPy

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

code: count from 0

Without vectorization

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

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

Without vectorization

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

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

Vectorization

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

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

Multiple Linear Regression - Model Representation

	Previous notation	Vector notation
Parameters	$w_1, \cdots, w_n$ b	$\vec{w} = [w_1 \quad \cdots \quad w_n]$ b
Model	$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + \cdots + w_n x_n + b$	$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$ dot product
Cost function	$J(w_1, \cdots, w_n, b)$	$J(\vec{w}, b)$

Gradient descent

Previous notation:

repeat {

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

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

}

Vector notation:

repeat {

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

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

}

Vector representation of X, y and w

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

$$\vec{w} \cdot \vec{x} = \begin{bmatrix} w_0 \\ w_1 \\ \cdots \\ w_n \end{bmatrix} \begin{bmatrix} x_0^{(0)} & x_1^{(0)} & \cdots & x_n^{(0)} \\ x_0^{(1)} & x_1^{(1)} & \cdots & x_n^{(1)} \\ \cdots & \cdots & \cdots & \cdots \\ x_0^{(m-1)} & x_1^{(m-1)} & \cdots & x_n^{(m-1)} \end{bmatrix} = \begin{bmatrix} w_0 x_0^{(0)} + w_1 x_1^{(0)} + \cdots + w_n x_n^{(0)} \\ w_0 x_0^{(1)} + w_1 x_1^{(1)} + \cdots + w_n x_n^{(1)} \\ \cdots \\ w_0 x_0^{(m-1)} + w_1 x_1^{(m-1)} + \cdots + w_n x_n^{(m-1)} \end{bmatrix}$$

Dot product operation

$$\begin{bmatrix} \boxed{1} & \boxed{2} \\ \boxed{3} & \boxed{4} \\ \boxed{5} & \boxed{6} \end{bmatrix} \begin{bmatrix} \boxed{2} \\ \boxed{3} \end{bmatrix} = \begin{bmatrix} \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \end{bmatrix} = \begin{bmatrix} \boxed{} \\ \boxed{} \\ \boxed{} \end{bmatrix}$$

Dot product operation

$$\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}^T\boldsymbol{b} = \sum_{i=0}^{n-1} a_i b_i$$

$$\boldsymbol{a} \cdot \boldsymbol{b} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = [a_0 b_0 + a_1 b_1 + a_2 b_2 + a_3 b_4]$$

$$\boldsymbol{a}^T\boldsymbol{b} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} [a_0 \quad a_1 \quad a_2 \quad a_3] \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = [a_0 b_0 + a_1 b_1 + a_2 b_2 + a_3 b_4]$$

transform

Gradient Descent

One feature

repeat {

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

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

simultaneously update w, b

}

n features ($n \geq 2$)

repeat {

$j=1$

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

$\vdots$

$j=n$

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

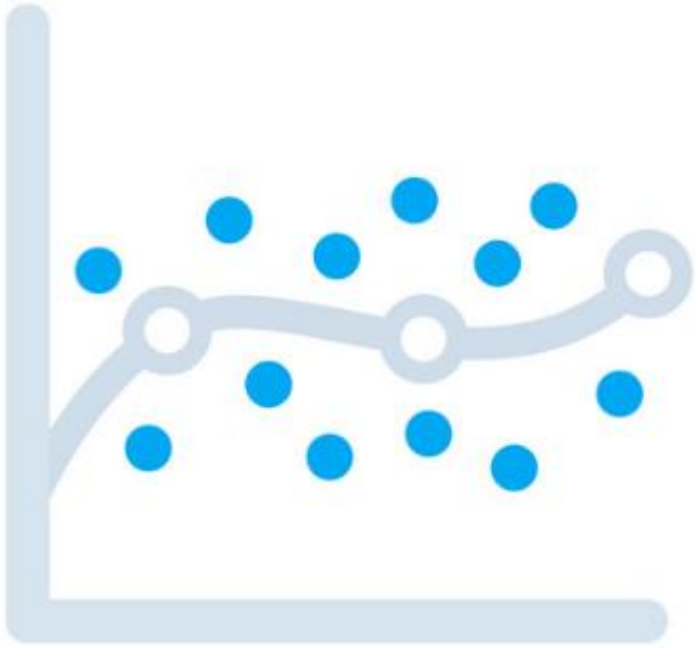
$$b = b - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)}\right)$$

simultaneously update

w_j (for $j = 1, \cdots, n$) and b

}

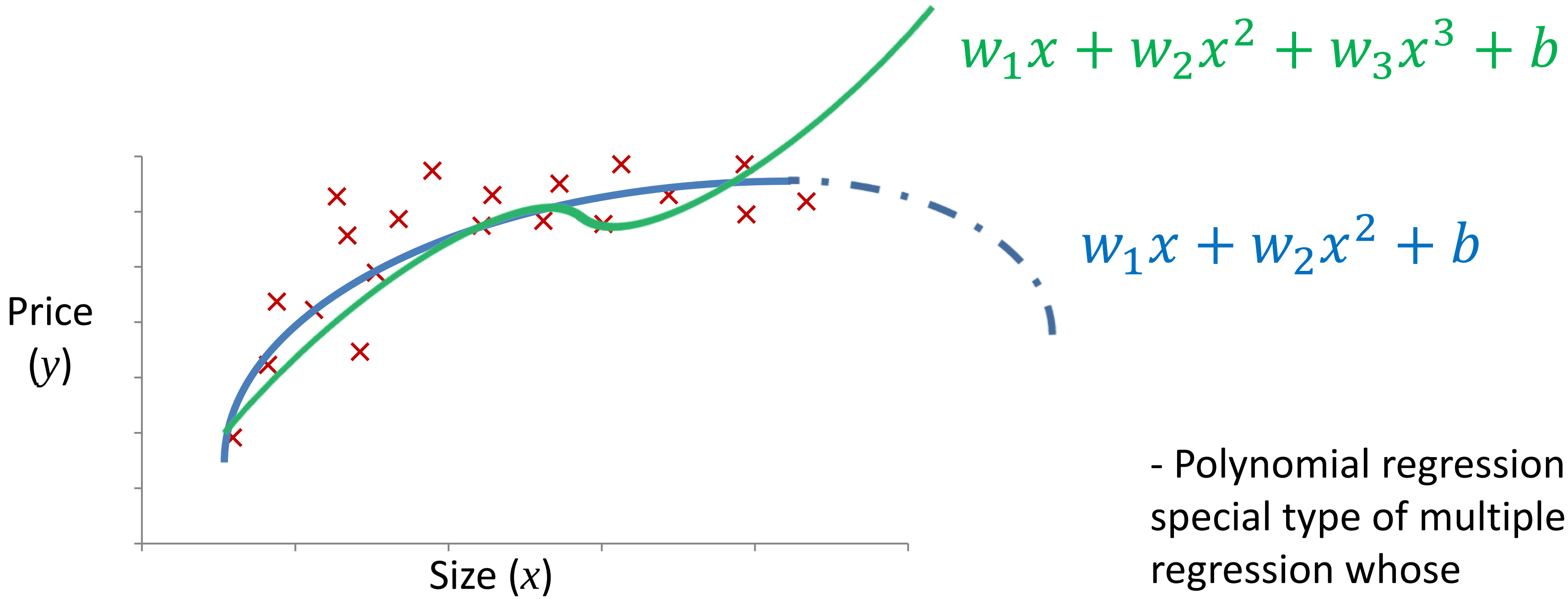
Polynomial Regression



Polynomial Regression

- When the relationship between the set of features and the target variable is not linear then we need to use polynomial regression of some degree
  - The degree of the polynomial is usually a hyperparameter of the model

Polynomial Regression

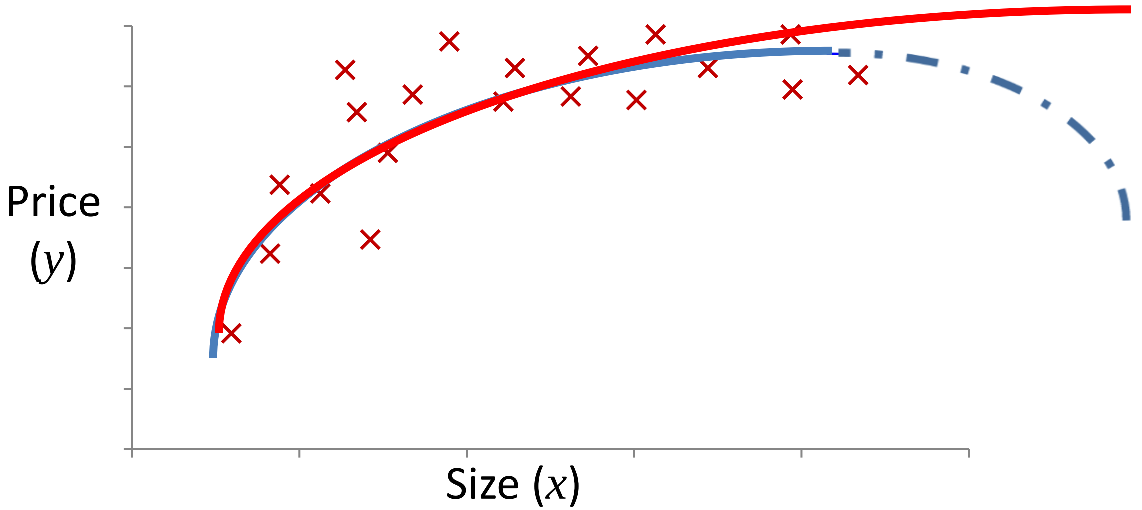


$$f(x) = w_1x_1 + w_2x_2 + w_3x_3 + b$$
$$= w_1(size) + w_2(size)^2 + w_3(size)^3 + b$$

$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$

- Polynomial regression is a special type of multiple regression whose independent variables are **powers** of variable X
- It is used to **approximate a curve** with unknown functional form
- For each additional power of X added to the model, the regression line will have one more bend

Polynomial Regression



$$f(x) = w_1(size) + w_2(size)^2 + \mathrm{b}$$

$$f(x) = w_1(size) + w_2\sqrt{(size)} + \mathrm{b}$$

Training Polynomial Regression Model

- Add **Polynomial Features** then train a Linear Regression Model
- For example, if we have two features then our new features will be: x_1, x_2, x_1x_2, x_1^2, x_2^2, and the features matrix will be:

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & x_{11}x_{12} & x_{11}^2 & x_{12}^2 \\ 1 & x_{21} & x_{22} & x_{21}x_{22} & x_{21}^2 & x_{22}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & x_{n1}x_{n2} & x_{n1}^2 & x_{n2}^2 \end{pmatrix}$$

- Scikit-learn provides the transformer PolynomialFeatures that creates the features matrix consisting of all the polynomial combinations of the features up to a **specified degree**

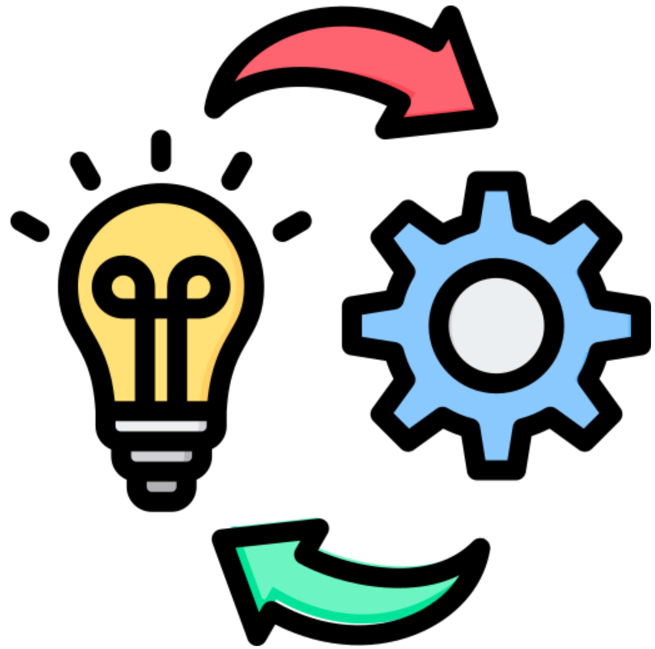
- Then we can use the created features matrix to train a linear regression model

Polynomial Regression

- We can treat the powers of x: x, x^2, ..., x^d, as distinct independent variables
  - Then, polynomial regression becomes a special case of multiple linear regression, since the model is still linear in the parameters that need to be estimated

- Therefore, we can find the optimal parameters $\vec{w}$ using gradient descent

Regression Practical Considerations

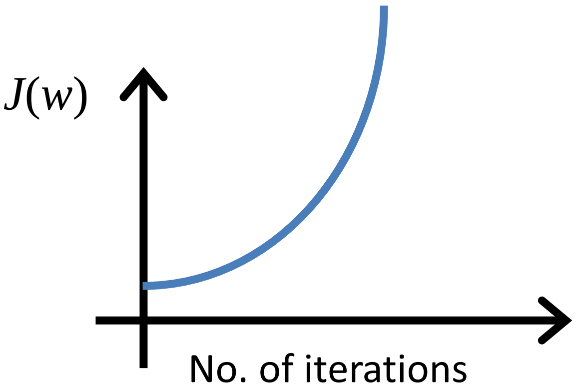


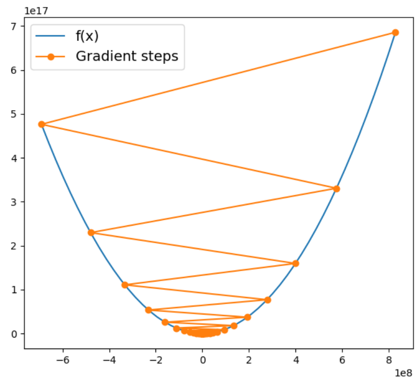
Gradient descent in practice:

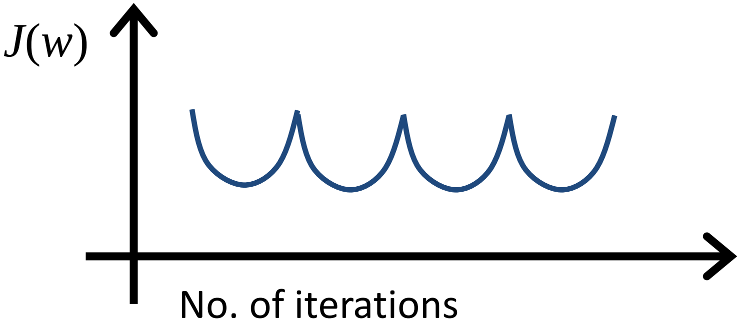
- Debugging Learning rate
- Feature Scaling
- Regularization
- Regression Evaluation

Learning rate: Gradient descent not working

Use smaller α







- For sufficiently small α, $J(w)$ should decrease on every iteration
- If α is too small, gradient descent can be slow to converge
- If α is too large: $J(w)$ may not decrease on every iteration; may not converge

To choose α, try

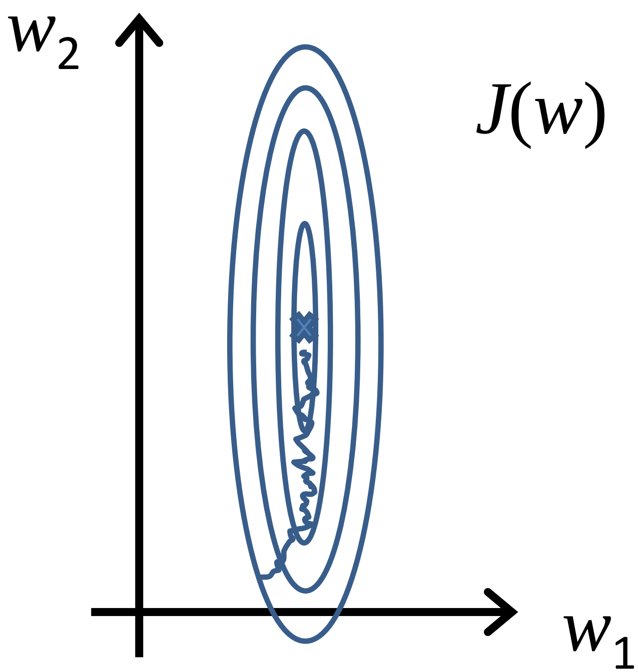
…, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, …

Feature Scaling: divide the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1.

The idea: Make sure features are on a similar scale. So that the gradient descent converges faster.

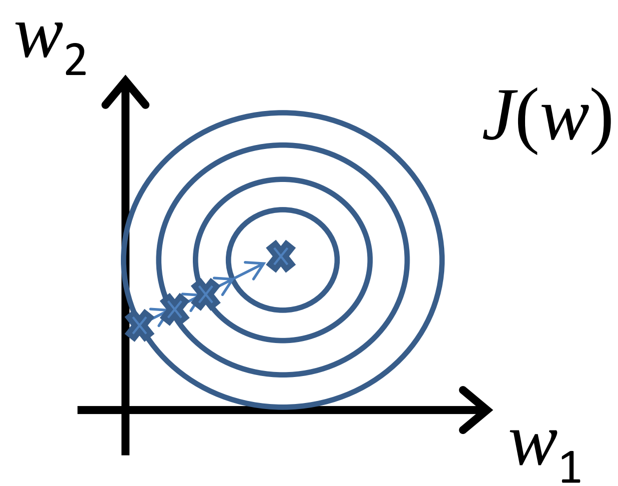
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2)}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Rule-of-thumb: Get every feature into approximately $-1 \leq x_i \leq 1$ range, $-0.5 \leq x_i \leq 0.5$, or other similar small ranges.

Mean normalization

- Replace x_i to make features have approximately zero mean: $$x_i = \frac{x_i - \mu_i}{s_i}$$

  Where μ_i is the **average** of all the values for feature (i) (**<u>in the training set</u>**) and s_i is the standard deviation.

  – e.g.,

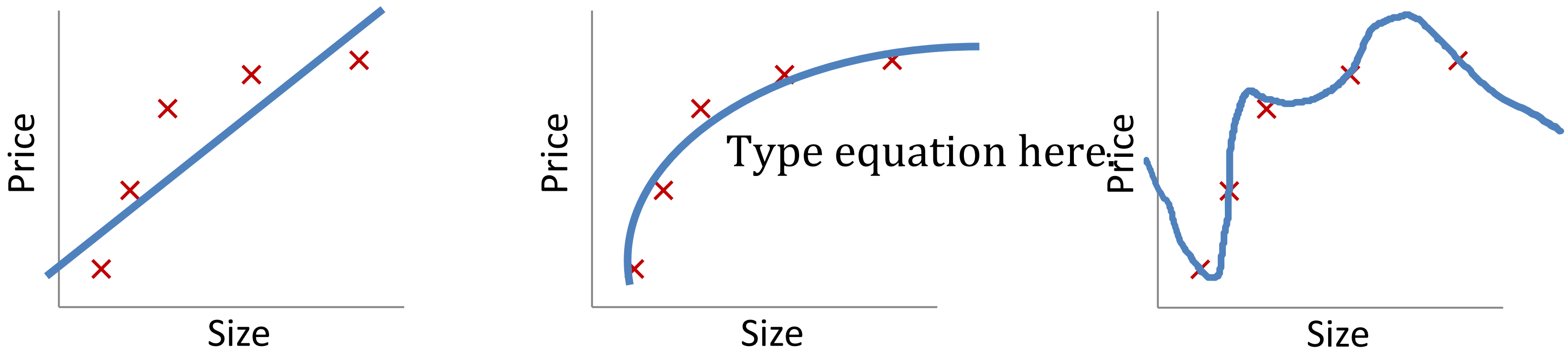
$x_1 = \frac{size - 1000}{400}$ (average size of the houses is 1000, and standard deviation is 400)

$x_2 = \frac{\#bedrooms - 2}{3}$ (average # of bedrooms is 2, and the standard deviation 3)

Regularization

- The problem of overfitting

Example: Linear regression (housing prices)



$w_0 + w_1 x$ | $w_0 + w_1 x + w_2 x^2$ | $w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$

"underfit/high bias" | "just right" | "overfit/high variance"

Overfitting: If we have too many features, the learned function may fit the training set very well ($J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2 \approx 0$) but fail to generalize to new examples (predict prices on new examples)

Addressing overfitting:

$x_1 =$ size of house

$x_2 =$ no. of bedrooms

$x_3 =$ no. of floors

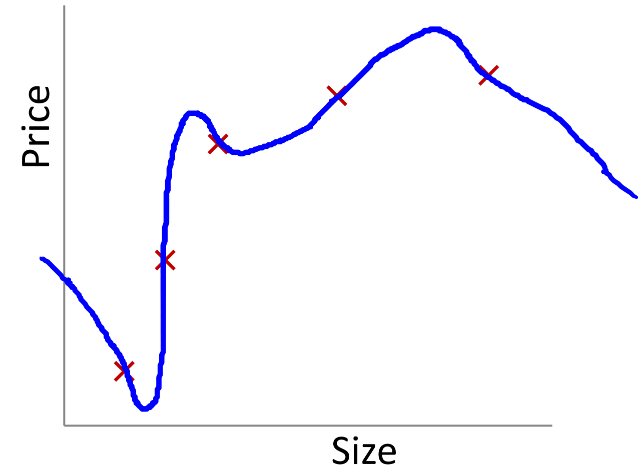
$x_4 =$ age of house

$x_5 =$ average income in neighborhood

$x_6 =$ kitchen size

$\vdots$

x_{100}



Addressing overfitting:

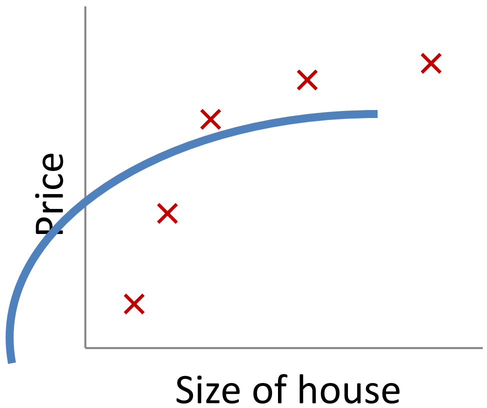
Options:

1. Reduce number of features.
   - Manually select which features to keep
   - Use feature selection algorithm
2. Regularization.
   - Keep all the features, but reduce magnitude/values of parameters w_j
   - Works well when we have a lot of features, each of which contributes a bit to predicting y

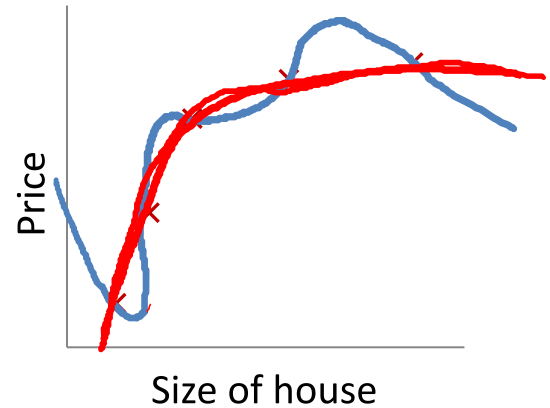
Regularization

- Cost function

Intuition



$$w_0 + w_1 x + w_2 x^2$$



$$w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$$

Suppose we penalize and make w_3, w_4 really small

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (f_w(x^{(i)}) - y^{(i)})^2 + 1000\, w_3^2 + 1000\, w_4^2$$

$$w_3 \approx 0, \quad w_4 \approx 0$$

Regularization

Small values for parameters $w_1, w_2, \ldots, w_n$

- "Simpler/smoother" function
- Less prone to overfitting

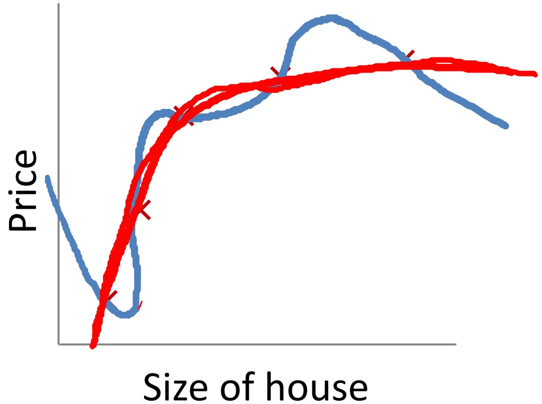
Housing:

- Features: $x_1, x_2, \ldots, x_{100}$
- Parameters: $w_1, w_2, \ldots, w_{100}$

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

$$J(w) = \frac{1}{2m} \left[\sum_{i=1}^{m} \left(f_w\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} w_j^2 \right]$$

$$\min_w J(w)$$



Lasso (L1) Regularization

$$\frac{1}{2m}\sum_{i=1}^{m}(y-Xw)^2+alpha\sum_{j=1}^{p}\left|w_j\right|$$

Ridge (L2) Regularization

$$\sum_{i=1}^{n}(y-Xw)^2+alpha\sum_{j=1}^{p}w_j^{\,2}$$

Regularized linear regression

$$J(w) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} w_j^2\right]$$

$$\min_{w} J(w)$$

Gradient descent

Repeat $\{$

$$b = b - \alpha \overbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)}}^{\frac{\partial}{\partial w_0}J(w)}$$

$$w_j = w_j - \alpha \underbrace{\left[\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} - \frac{\lambda}{m} w_j\right]}_{\frac{\partial}{\partial w_j}J(w) \text{ "Regularized"}}$$

$\}$ $\quad (j = 1, 2, 3, \ldots, n)$

$$w_j = w_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

$$1 - \alpha\frac{\lambda}{m} < 1$$

Regression Evaluation

- Performance measured by
  - Mean Squared Error (MSE) $MSE = \frac{1}{n}\sum(y - \hat{y})^2$
  - Root-Mean-Squared-Error (RMSE) $RMSE = \sqrt{\frac{(y-\hat{y})^2}{n}}$
  - Mean-Absolute-Error (MAE) $MAE = \frac{1}{n}\sum|y - \widehat{y}|$
  - ...others