Aprendizado de Máquina e Reconhecimento de Padrões 2021.2

Linear Regression and Gradient Descent

Prof. Dr. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas

Notations

Notations

These are the notations ˙ that we will use throughout this course:

housing median age	population	median income	housing price
10	546	76000.0	162500.0
7	6274	244970.0	90900.0
37	1095	432030.0	232800.0
32	1818	429690.0	145800.0

Notations

$m = 4$
$n = 3$

These are the notations that we will use throughout this course:

- m is the **number of instances** in the dataset or of given set if specified (*e.g.*, the training set);
- n is the **number of features** (excluding the label / dependent variable / outcome);

housing median age	population	median income	housing price
10	546	76000.0	162500.0
7	6274	244970.0	90900.0
37	1095	432030.0	232800.0
32	1818	429690.0	145800.0

Notations

$m = 4$
$n = 3$

These are the notations that we will use throughout this course:

- m is the **number of instances** in the dataset or of given set if specified (*e.g.*, the training set);
- n is the **number of features** (excluding the label / dependent variable / outcome);
- $\mathbf{x}^{(i)}$ is a **vector of all the feature values** / **feature vector** (excluding the label / dependent variable / outcome) , and
- $y^{(i)}$ is its **label** (the desired **output value** for that instance).

housing median age	population	median income	housing price
10	546	76000.0	162500.0
7	6274	244970.0	90900.0
37	1095	432030.0	232800.0
32	1818	429690.0	145800.0

$$\mathbf{x}^{(2)} = \begin{bmatrix} 7 \\ 6274 \\ 244970.0 \end{bmatrix} \qquad y^{(2)} = 90900.0$$

Notations

$m = 4$
$n = 3$

These are the notations · that we will use throughout this course:

- m is the **number of instances** in the dataset or of given set if specified (*e.g.*, the training set);
- n is the **number of features** (excluding the label / dependent variable / outcome);
- $\mathbf{x}^{(i)}$ is a **vector of all the feature values** / **feature vector** (excluding the label / dependent variable / outcome) , and
- $y^{(i)}$ is its **label** (the desired **output value** for that instance).

- $\mathbf{X}$ is a **matrix** containing **all the feature values** (excluding labels) of all **instances in the dataset** → **feature matrix**.
- y is a **vector** containing **all the labels** of **all instances in the dataset** → **true labels**.

housing median age	population	median income	housing price
10	546	76000.0	162500.0
7	6274	244970.0	90900.0
37	1095	432030.0	232800.0
32	1818	429690.0	145800.0

$$\mathbf{x}^{(2)} = \begin{bmatrix} 7 \\ 6274 \\ 244970.0 \end{bmatrix} \qquad y^{(2)} = 90900.0$$

$$\mathbf{X} = \begin{bmatrix} (\boldsymbol{x}^{(1)})^T \\ (\boldsymbol{x}^{(2)})^T \\ (\boldsymbol{x}^{(3)})^T \\ (\boldsymbol{x}^{(4)})^T \end{bmatrix} = \begin{bmatrix} 10 & 546 & 76000.0 \\ 7 & 6274 & 244970.0 \\ 37 & 1095 & 432030.0 \\ 32 & 1818 & 429690.0 \end{bmatrix}$$

$$y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & y^{(4)} \end{bmatrix}$$
$$= [162500.0 \quad 90900.0 \quad 232800.0 \quad 145800.0]$$

Notations

$m = 4$
$n = 3$

These are the notations that we will use throughout this course:

- m is the **number of instances** in the dataset or of given set if specified (*e.g.*, the training set);
- n is the **number of features** (excluding the label / dependent variable / outcome);
- $\mathbf{x}^{(i)}$ is a **vector of all the feature values** / **feature vector** (excluding the label / dependent variable / outcome) , and
- $y^{(i)}$ is its **label** (the desired **output value** for that instance).

- $\mathbf{X}$ is a **matrix** containing **all the feature values** (excluding labels) of all **instances in the dataset** → **feature matrix**.
- y is a **vector** containing **all the labels** of **all instances in the dataset** → **true labels**.

- h is the **hypothesis** (prediction function).
- $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$ is the **predicted value** for the given **instance's feature vector** $\mathbf{x}^{(i)}$.

housing median age	population	median income	housing price
10	546	76000.0	162500.0
7	6274	244970.0	90900.0
37	1095	432030.0	232800.0
32	1818	429690.0	145800.0

$$\mathbf{x}^{(2)} = \begin{bmatrix} 7 \\ 6274 \\ 244970.0 \end{bmatrix} \qquad y^{(2)} = 90900.0$$

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ (\mathbf{x}^{(3)})^T \\ (\mathbf{x}^{(4)})^T \end{bmatrix} = \begin{bmatrix} 10 & 546 & 76000.0 \\ 7 & 6274 & 244970.0 \\ 37 & 1095 & 432030.0 \\ 32 & 1818 & 429690.0 \end{bmatrix}$$

$$y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & y^{(4)} \end{bmatrix}$$
$$= [162500.0 \quad 90900.0 \quad 232800.0 \quad 145800.0]$$

Notations

$m = 4$
$n = 3$

These are the notations that we will use throughout this course:

- m is the **number of instances** in the dataset or of given set if specified (*e.g.*, the training set);
- n is the **number of features** (excluding the label / dependent variable / outcome);
- $\mathbf{x}^{(i)}$ is a **vector of all the feature values** / **feature vector** (excluding the label / dependent variable / outcome) , and
- $y^{(i)}$ is its **label** (the desired **output value** for that instance).

- $\mathbf{X}$ is a **matrix** containing **all the feature values** (excluding labels) of all **instances in the dataset** → **feature matrix**.
- y is a **vector** containing **all the labels** of **all instances in the dataset** → **true labels**.

- h is the **hypothesis** (prediction function).
- $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$ is the **predicted value** for the given **instance's feature vector** $\mathbf{x}^{(i)}$.

housing median age	population	median income	housing price
10	546	76000.0	162500.0
7	6274	244970.0	90900.0
37	1095	432030.0	232800.0
32	1818	429690.0	145800.0

$$\mathbf{x}^{(2)} = \begin{bmatrix} 7 \\ 6274 \\ 244970.0 \end{bmatrix} \qquad y^{(2)} = 90900.0$$

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ (\mathbf{x}^{(3)})^T \\ (\mathbf{x}^{(4)})^T \end{bmatrix} = \begin{bmatrix} 10 & 546 & 76000.0 \\ 7 & 6274 & 244970.0 \\ 37 & 1095 & 432030.0 \\ 32 & 1818 & 429690.0 \end{bmatrix}$$

$$y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & y^{(4)} \end{bmatrix}$$
$$= [162500.0 \quad 90900.0 \quad 232800.0 \quad 145800.0]$$

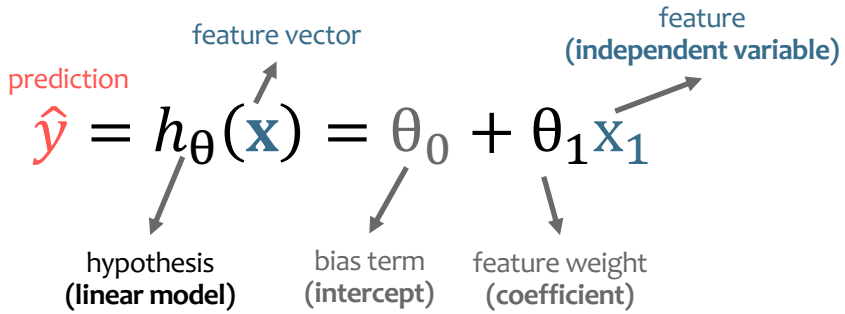
Remember that Python indexes **start at 0.**

Linear Regression

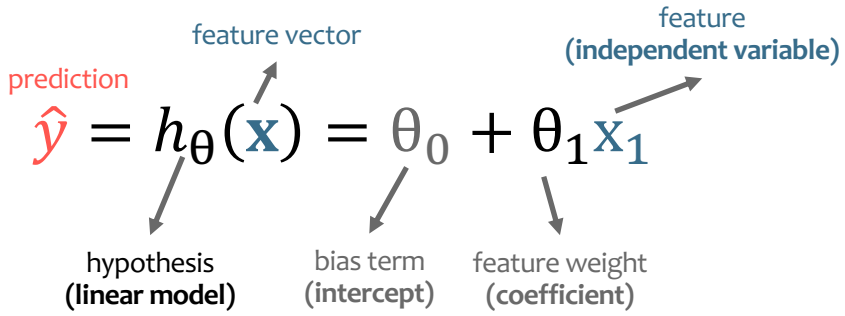
Linear Regression

- An **linear** approach for modeling the **relationship** between **independent (explanatory) variables** and a **numerical dependent variable** by fitting a **linear model** (*e.g.*,, a straight line) to the observations of a training set.
- Such a linear model is used **to predict** **numerical dependent variables** for new observations (unseen data).

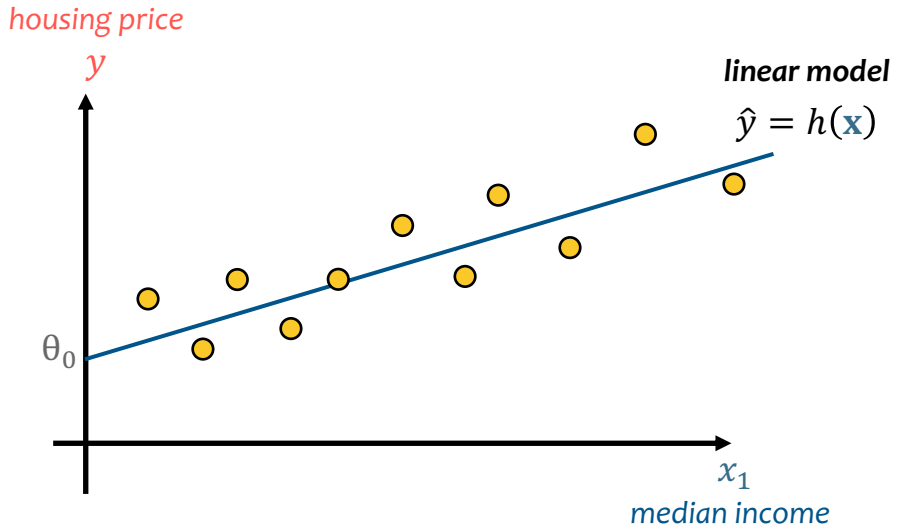
Simple Linear Regression



Simple Linear Regression



median income	housing price
76000.0	162500.0
...	...

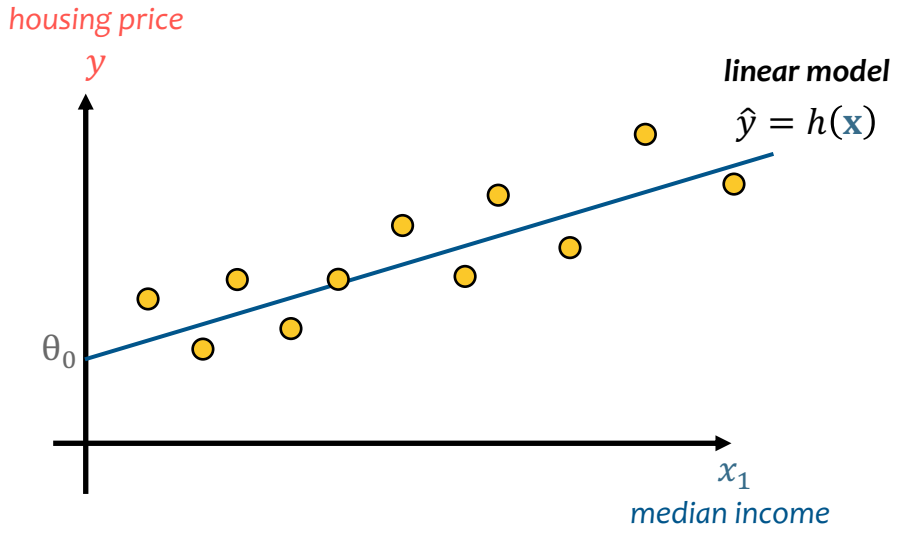


Simple Linear Regression

$$\hat{y} = h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 x_1$$

- $\hat{y}$: prediction
- $\mathbf{x}$: feature vector
- x_1: feature (independent variable)
- h_{θ}: hypothesis (linear model)
- θ_0: bias term (intercept)
- θ_1: feature weight (coefficient)

median income	housing price
76000.0	162500.0
...	...



Multiple Linear Regression

$$\hat{y} = h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

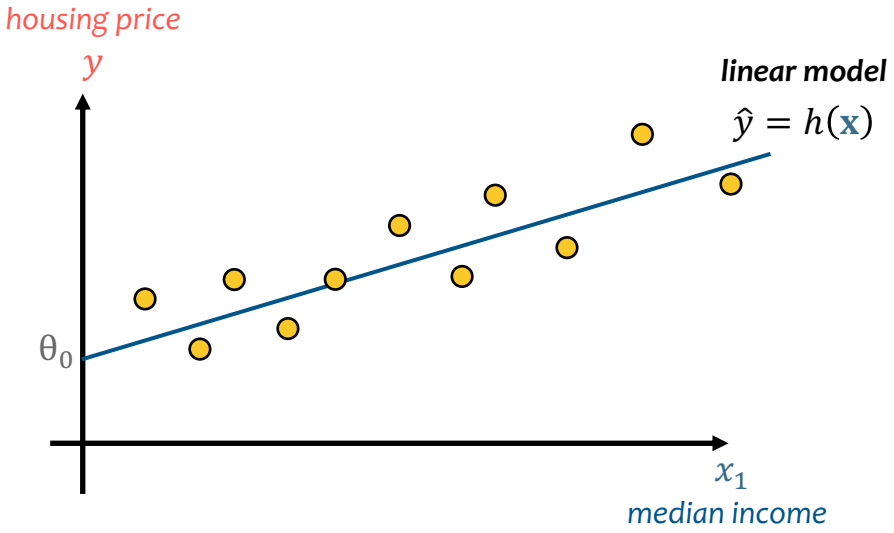
- $\hat{y}$: prediction
- $\mathbf{x}$: feature vector
- $x_1, x_2, \ldots, x_n$: features (independent variables)
- h_{θ}: hypothesis (linear model)
- θ_0: bias term (intercept)
- $\theta_1, \theta_2, \ldots, \theta_n$: feature weights (coefficients)

Simple Linear Regression

$$\text{prediction}: \hat{y} = h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 x_1$$

- $\mathbf{x}$: feature vector
- x_1: feature (independent variable)
- h_{θ}: hypothesis (linear model)
- θ_0: bias term (intercept)
- θ_1: feature weight (coefficient)

median income	housing price
76000.0	162500.0
…	…

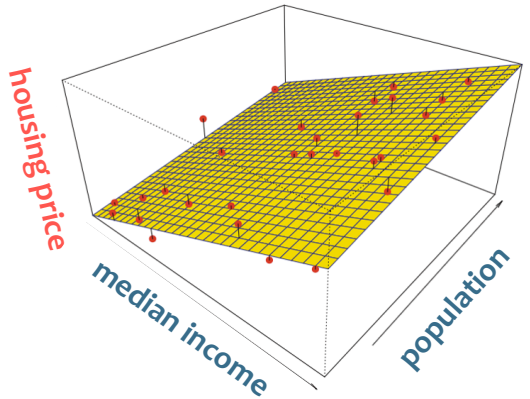


Multiple Linear Regression

$$\text{prediction}: \hat{y} = h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- $\mathbf{x}$: feature vector
- $x_1, x_2, \ldots, x_n$: features (independent variables)
- h_{θ}: hypothesis (linear model)
- θ_0: bias term (intercept)
- $\theta_1, \theta_2, \ldots, \theta_n$: feature weights (coefficients)

population	median income	housing price
546	76000.0	162500.0
…	…	…



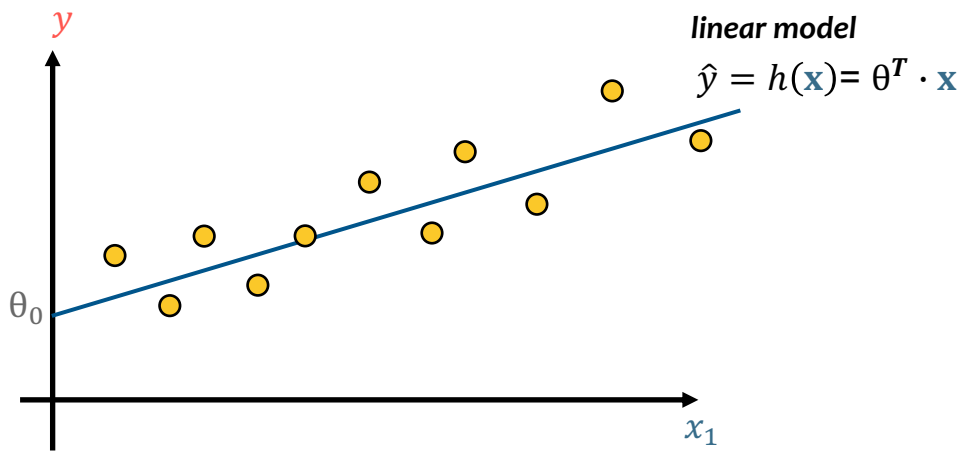
Linear Regression model (vectorized form)

$$\hat{y} = h_\theta(\mathbf{x}) = \theta^T \cdot \mathbf{x}$$

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0$$

$$\theta^T = [\theta_0 \ \theta_1 \cdots \theta_n]$$



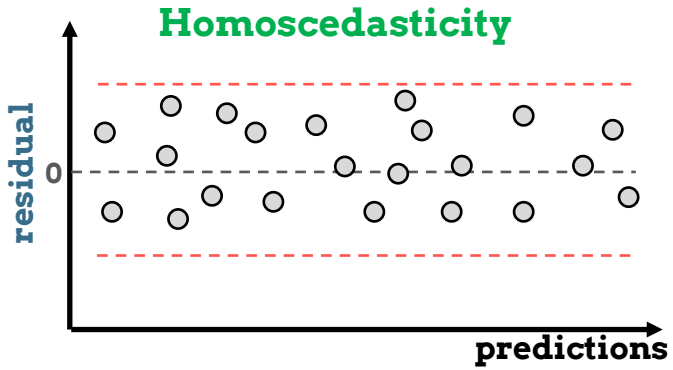
Assumptions of Linear Regression

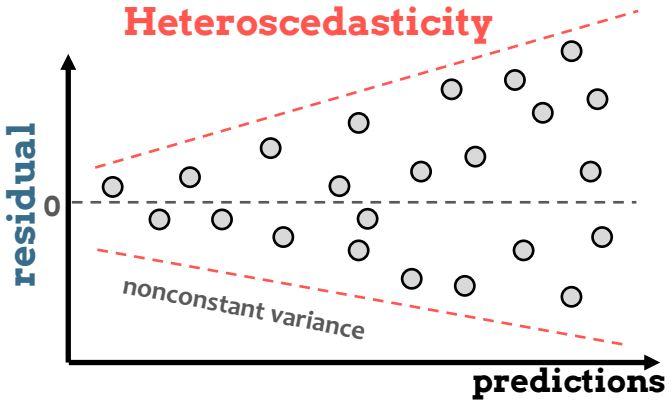
X = features (independent variables)

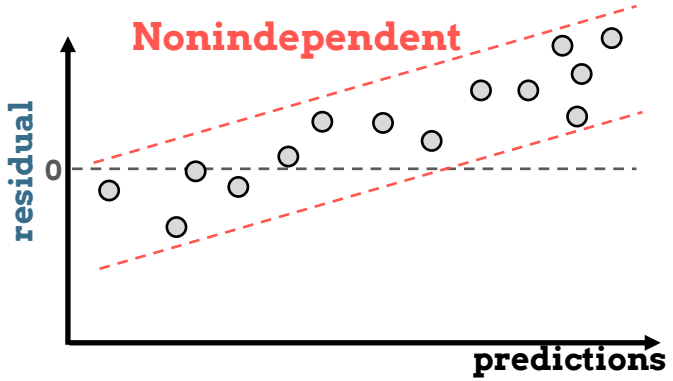
y = outcome (dependent variables)

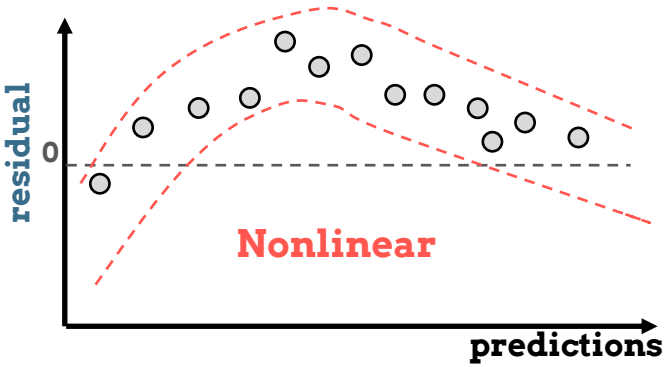
1. **Linearity:** The relationship between **X** and **y** is **linear**.
2. **Observations** are **independent** of each other.
3. No or little **multicollinearity** (the **independent variables** are **highly correlated** with each other).
4. **Residuals** **$(\hat{y} - y)$** are **normally** distributed.
5. **Homoscedasticity:** The **variance** of **residual** is the same for any value of **X**.

Checking Homoscedasticity Visually

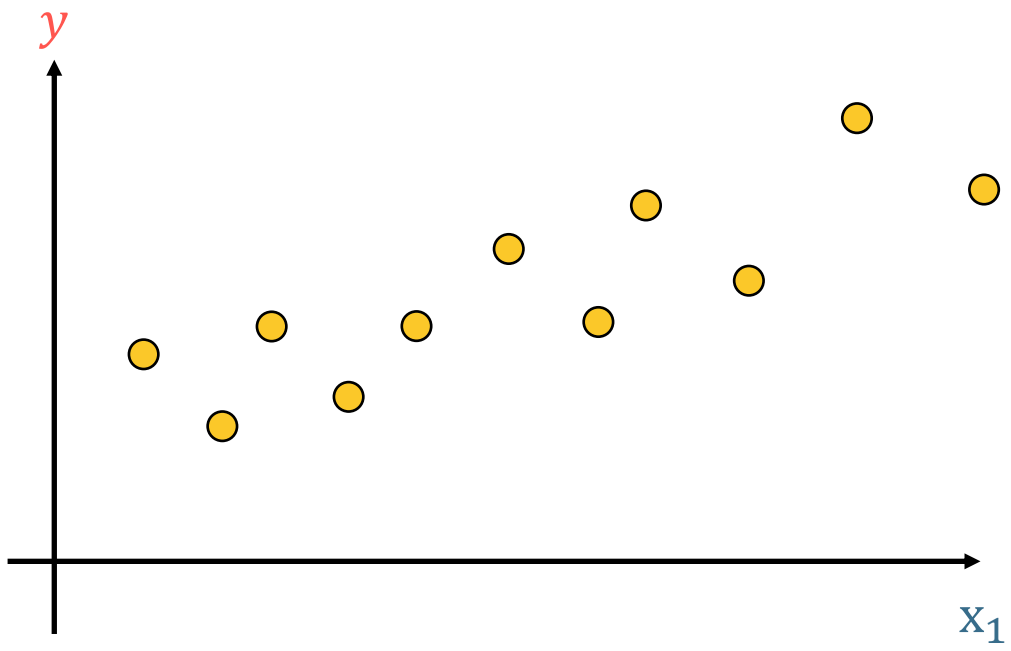




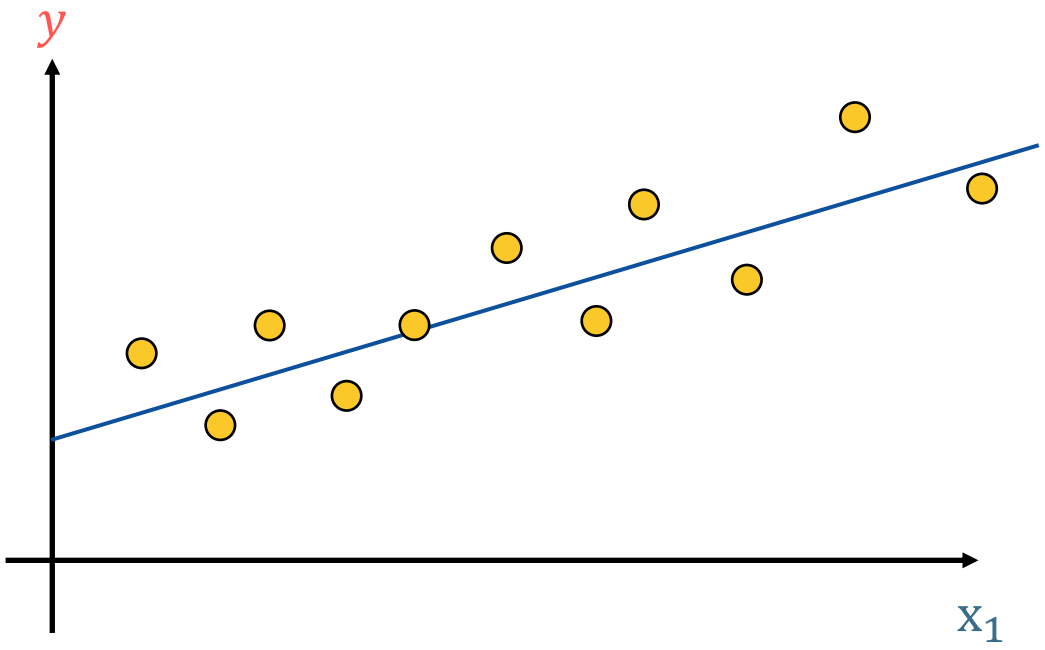




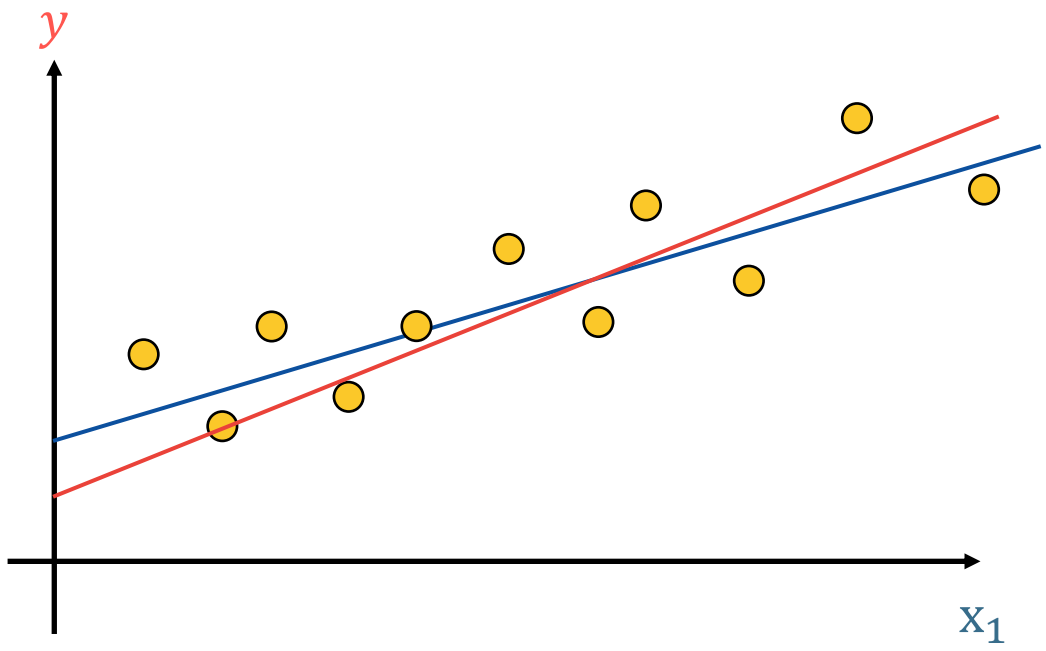
Which is the best linear model?



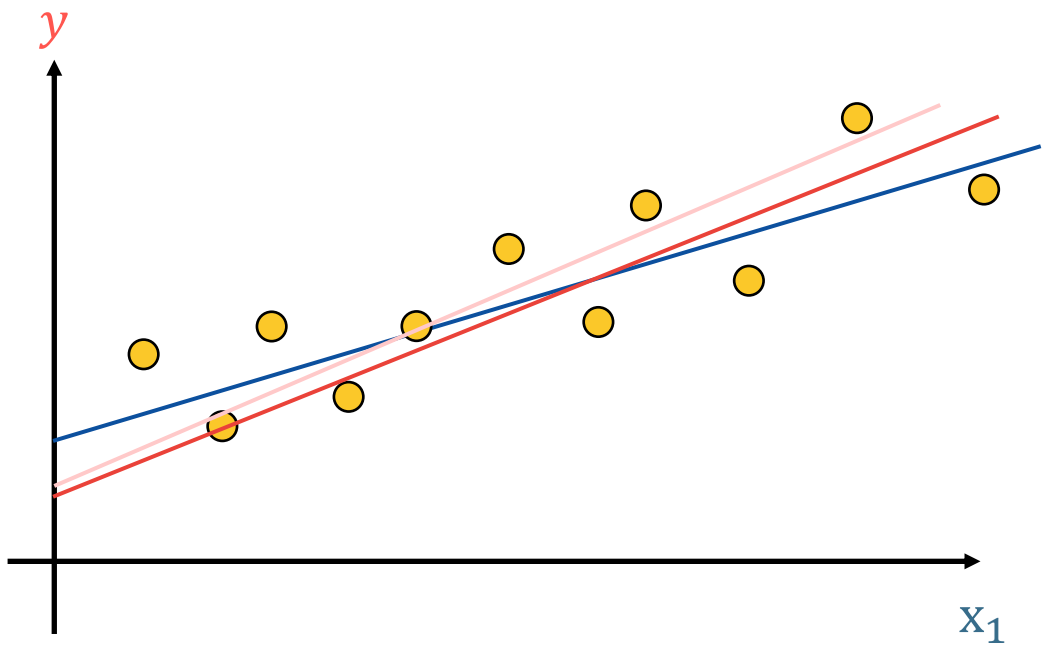
Which is the best linear model?



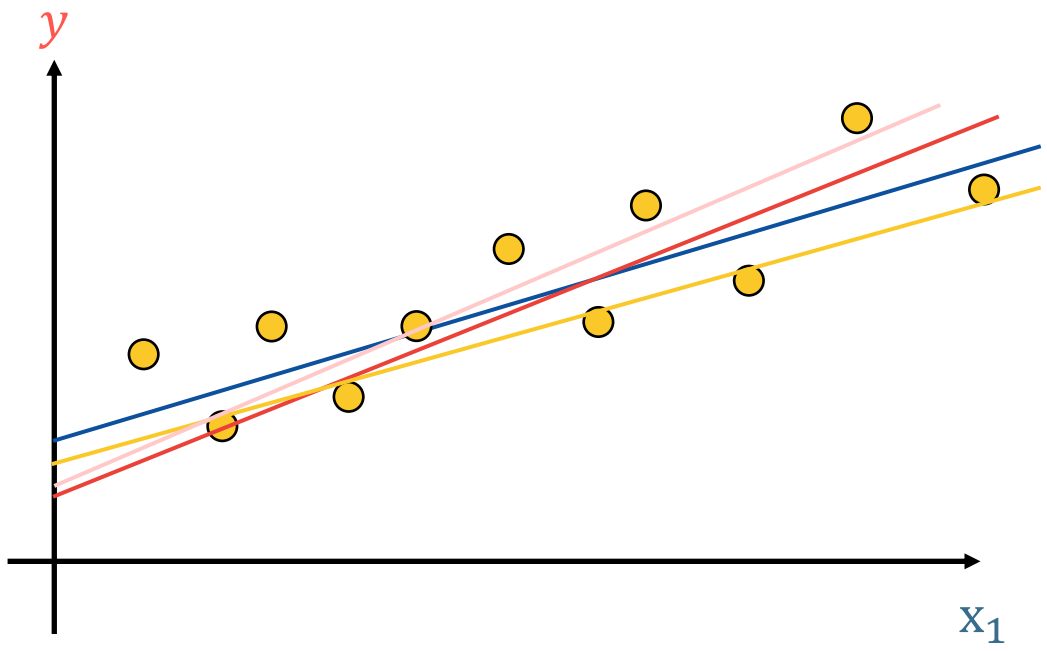
Which is the best linear model?



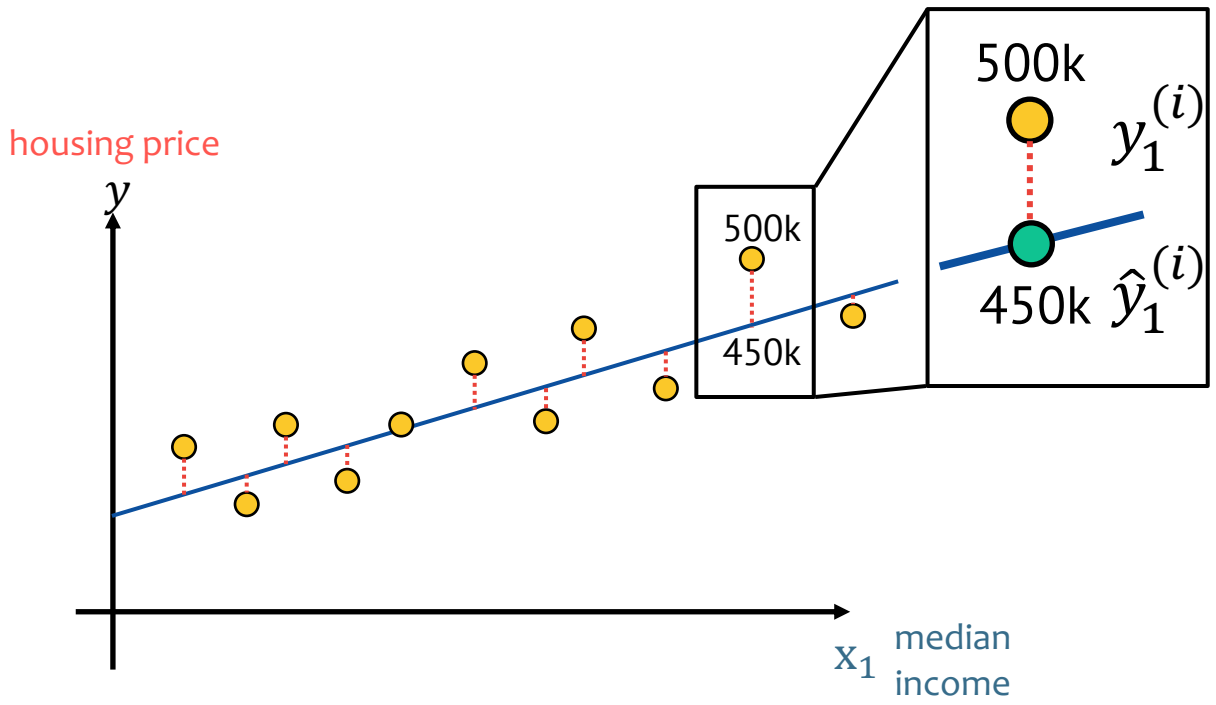
Which is the best linear model?



Which is the best linear model?



Cost Function / Error Function

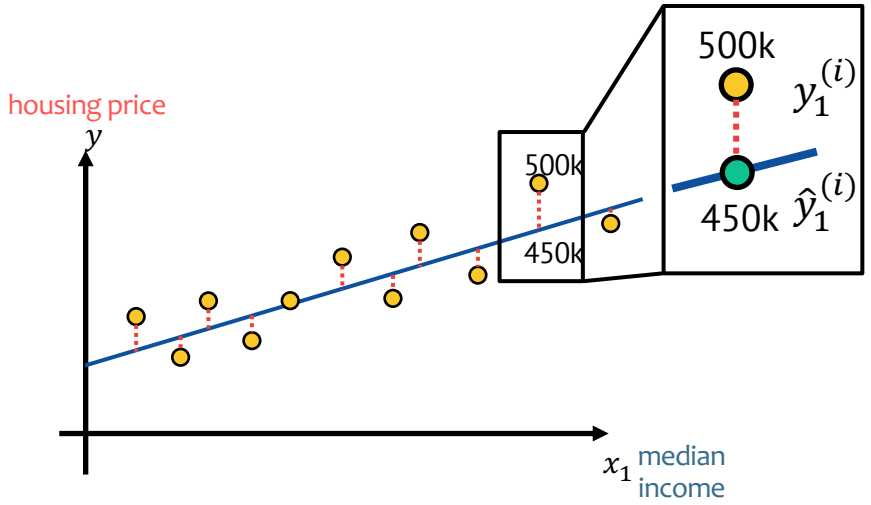


Cost Function / Error Function

Mean Square Error

$$J(\theta) = MSE(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^{T} \cdot \mathbf{x}^{(i)} - y^{(i)})^2$$

$$h_\theta(\mathbf{x}^{(i)}) = \hat{y}^{(i)}$$



Cost Function / Error Function

$$h_\theta(\mathbf{x}^{(i)}) = \hat{y}^{(i)}$$

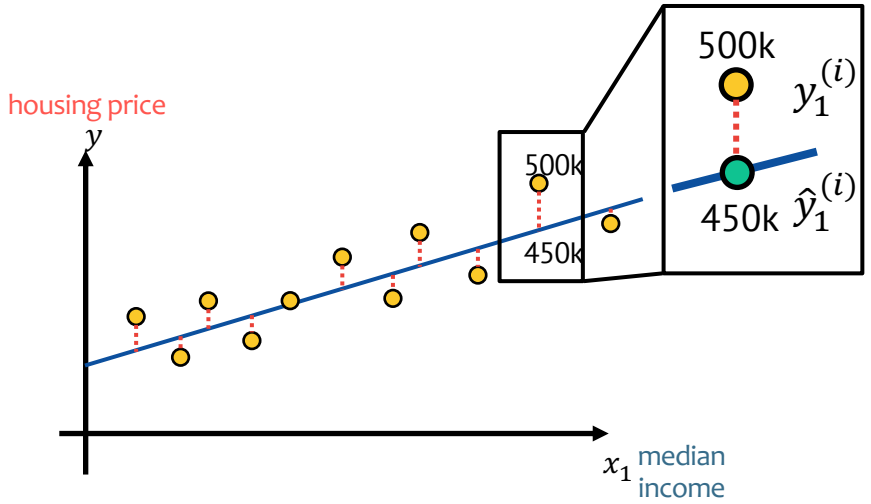
Mean Square Error

$$J(\theta) = MSE(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^{T} \cdot \mathbf{x}^{(i)} - y^{(i)})^2$$

training

Goal: Choose the parameters $\boldsymbol{\theta}$ (for simple regression, $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}$) that **minimizes** the **cost function** $J(\theta)$ for the training data.

Goal: Choose the parameters $\boldsymbol{\theta}$ (for simple regression, $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}$) so that $h_\theta(\mathbf{x}^{(i)})$ is **close** to y for the training examples.



Linear Model

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1$$

Simplified Linear Model

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_1 x_1$$

simplifying: $\theta_0 = 0$

Cost Function

$$J(\theta) = J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

Goal: $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

Cost Function

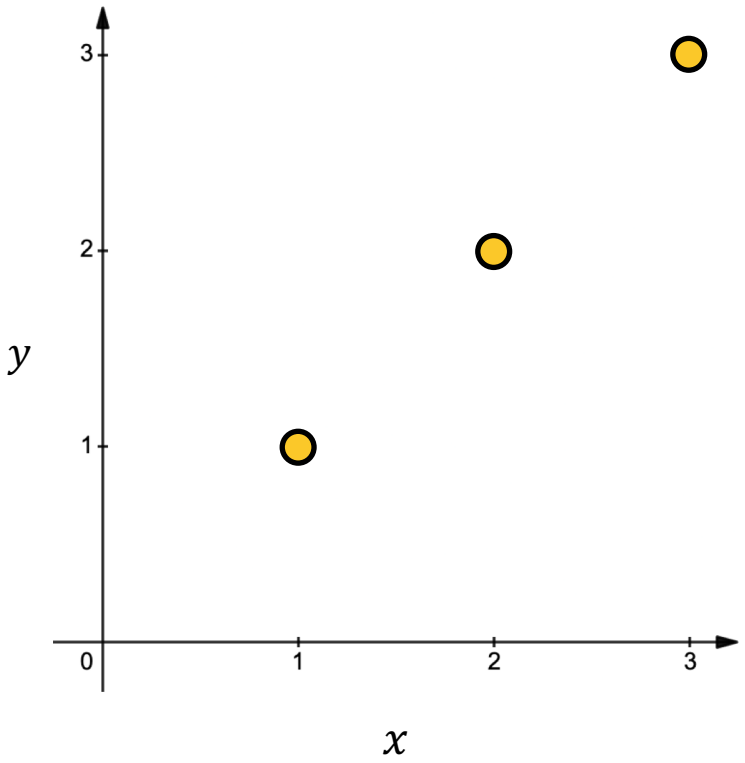
$$J(\theta) = J(\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

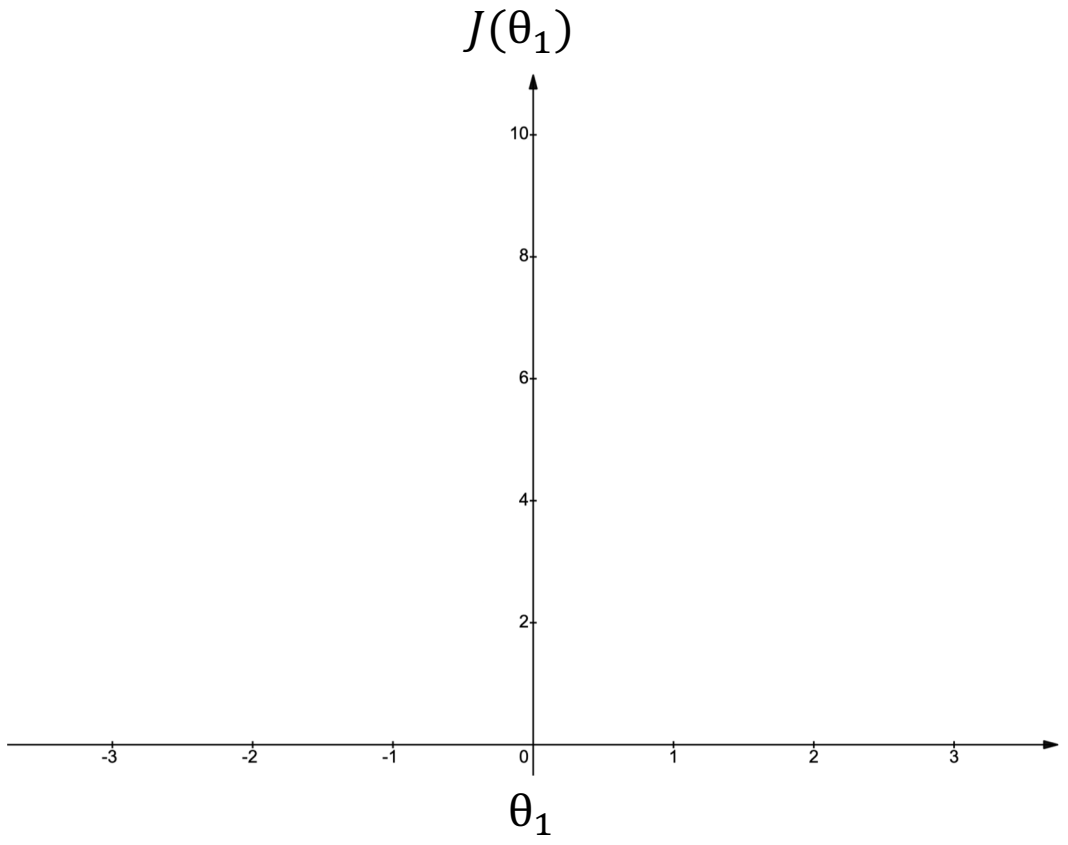
Goal: $\min_{\theta_1} J(\theta_1)$

Example

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_1 x_1$$

$$J(\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_1 x_1 - y^{(i)})^2$$

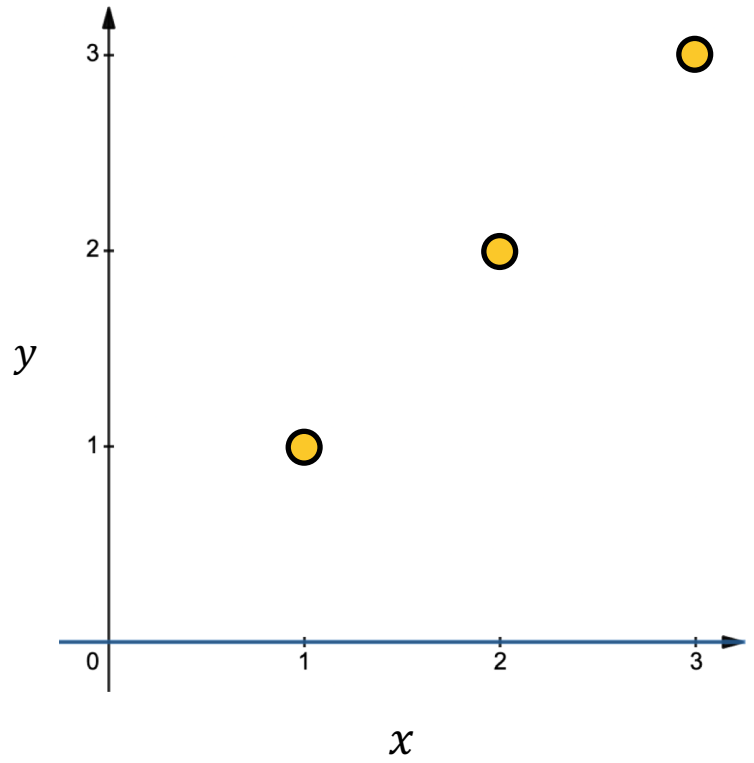




Example

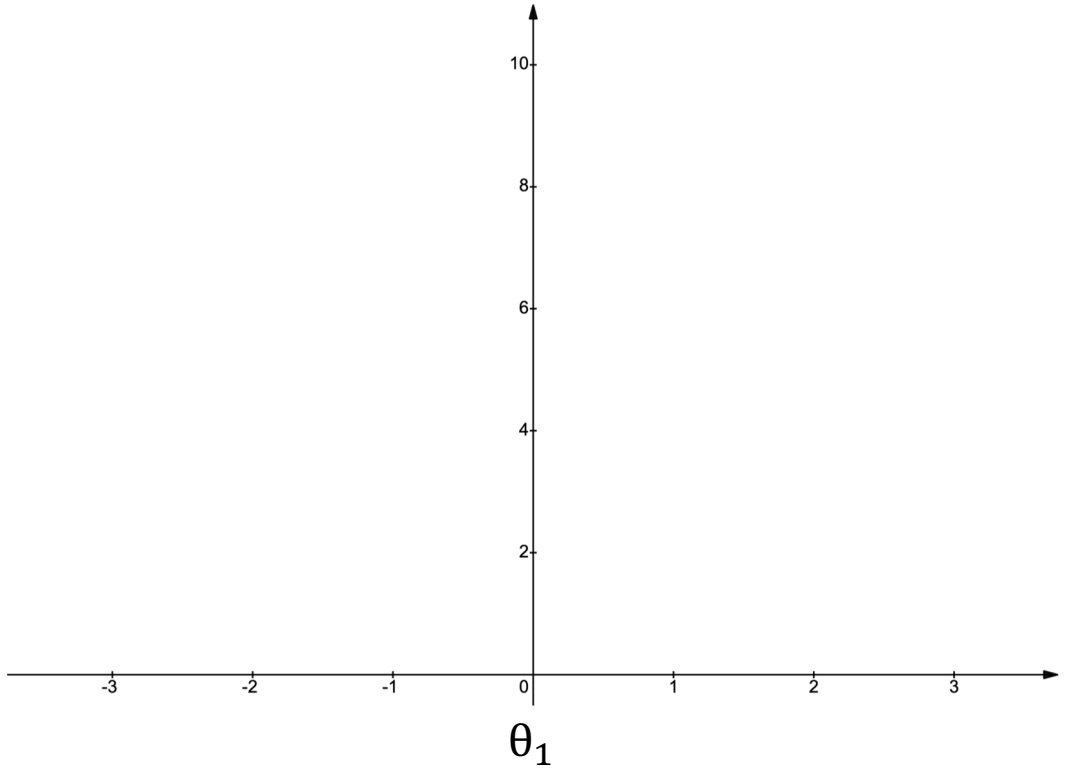
$$\hat{y} = h_\theta(\mathbf{x}) = \theta_1 x_1$$

$$\theta_1 = 0$$



$$J(\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_1 x_1 - y^{(i)})^2$$

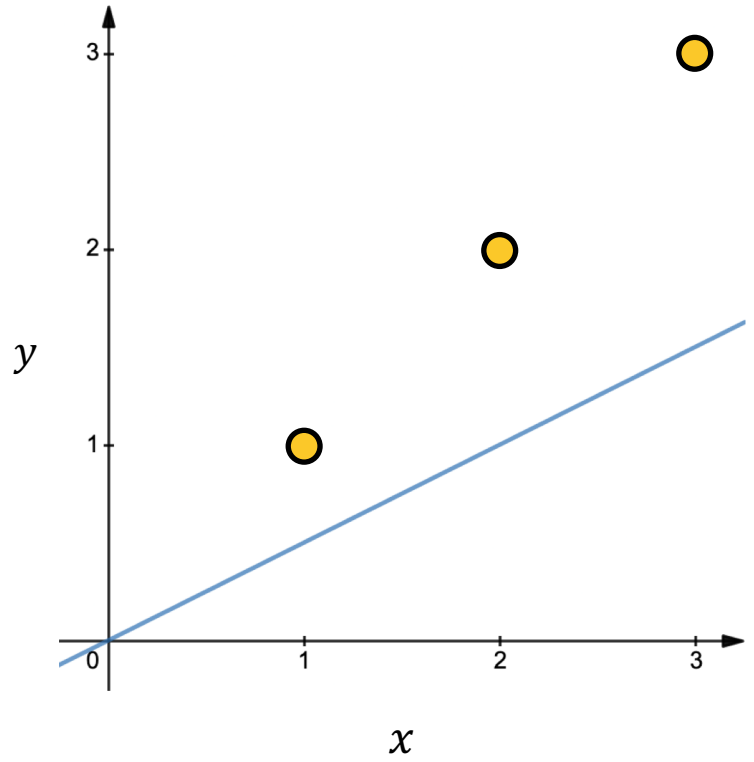
$J(\theta_1)$



Example

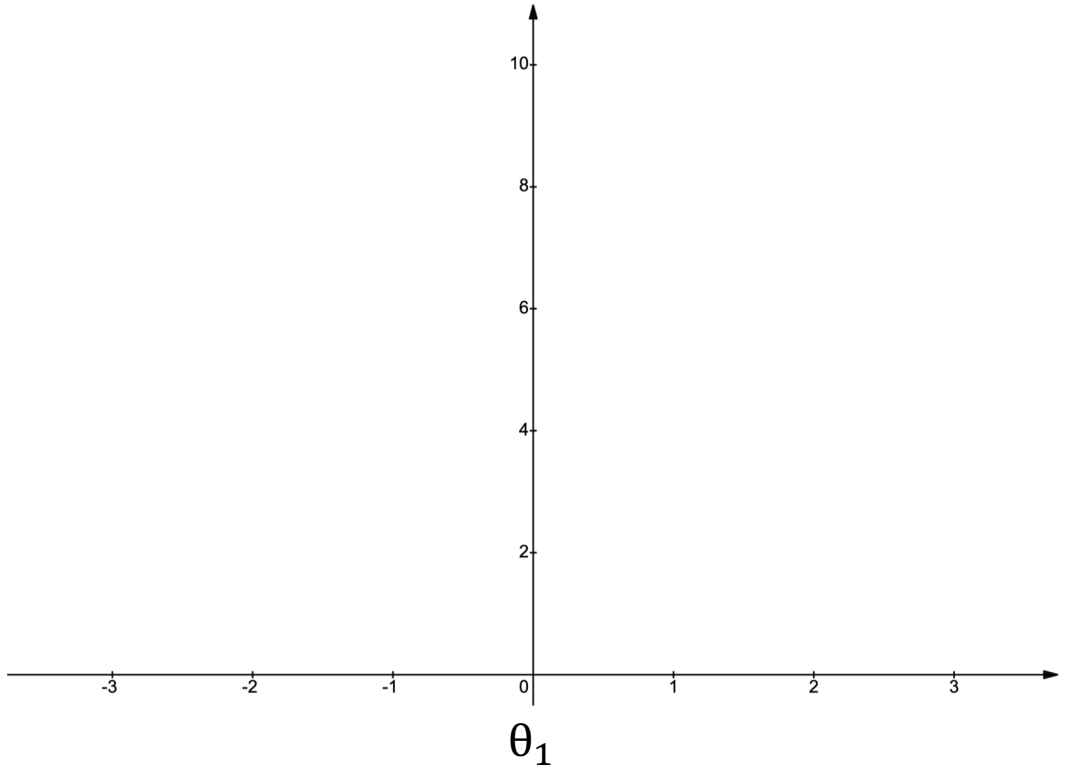
$$\hat{y} = h_\theta(\mathbf{x}) = \theta_1 x_1$$

$$\theta_1 = 0.5$$



$$J(\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_1 x_1 - y^{(i)})^2$$

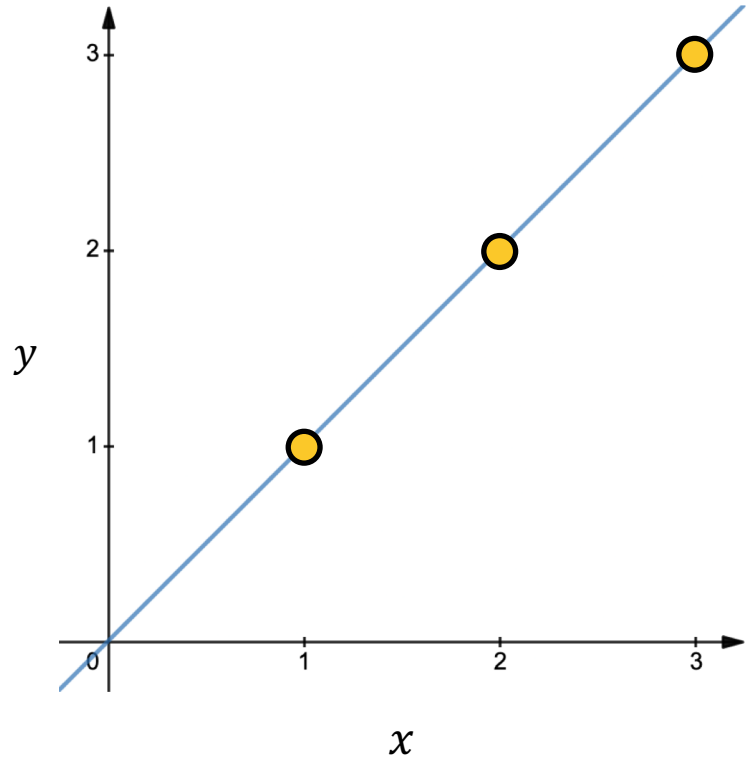
$J(\theta_1)$



Example

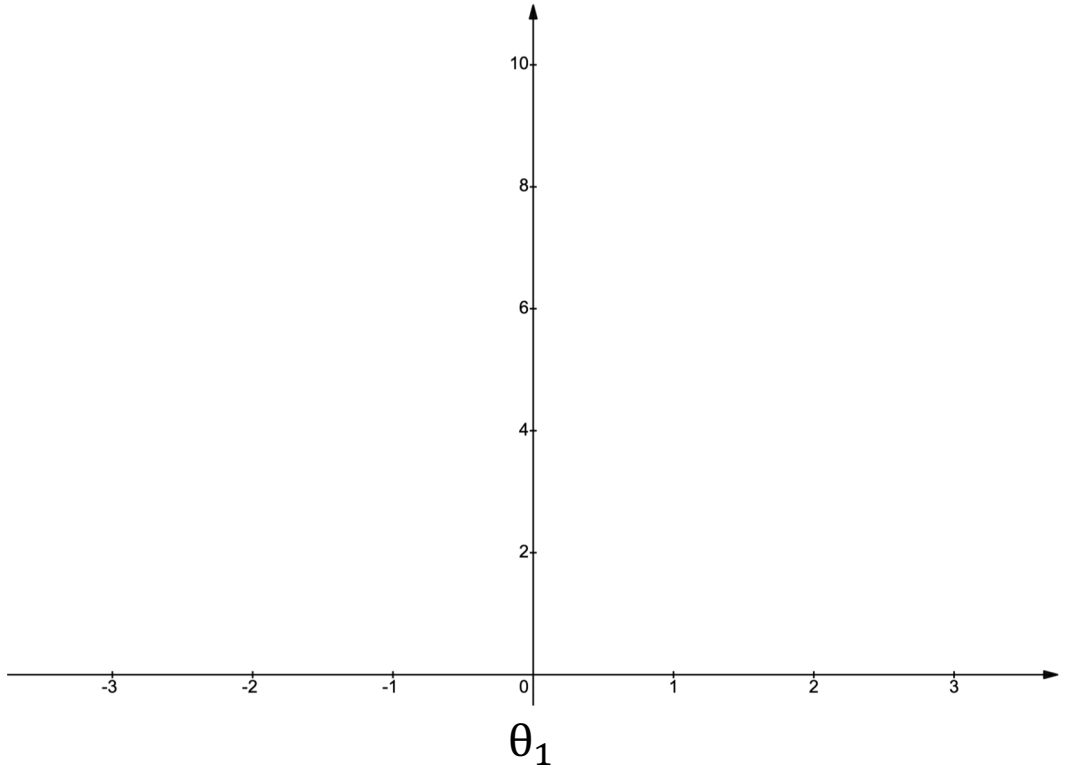
$$\hat{y} = h_\theta(\mathbf{x}) = \theta_1 x_1$$

$$\theta_1 = 1$$



$$J(\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_1 x_1 - y^{(i)})^2$$

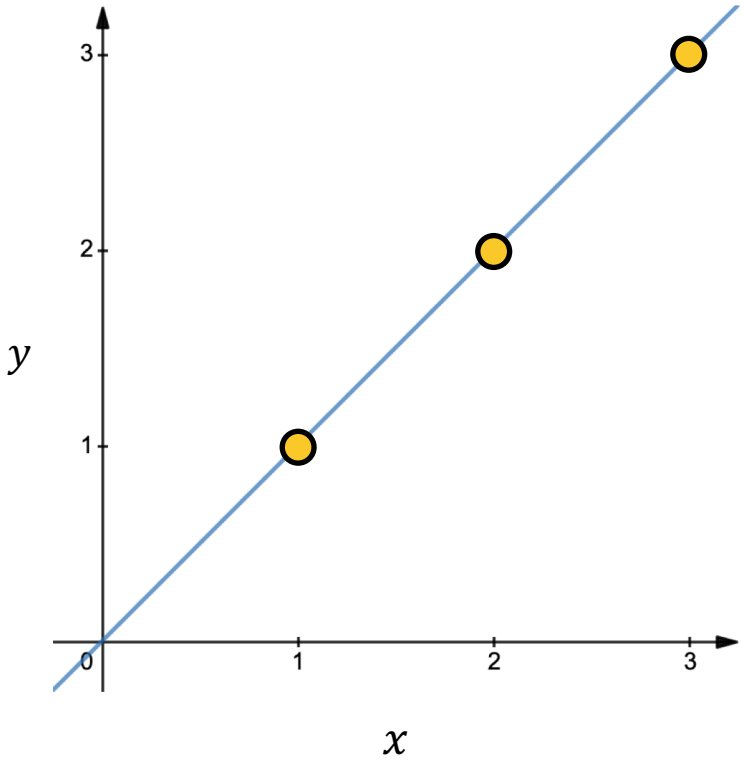
$$J(\theta_1)$$



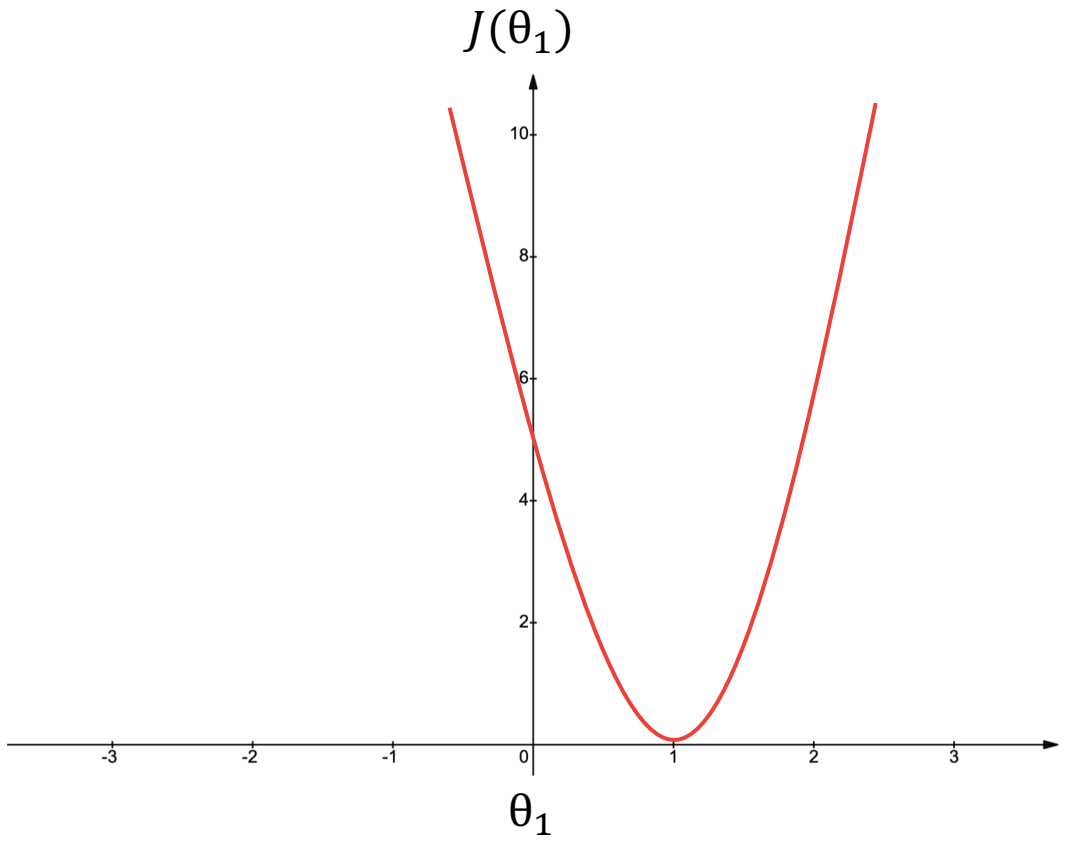
Example

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_1 x_1$$

$$\theta_1 = 1$$



$$J(\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_1 x_1 - y^{(i)})^2$$



Linear Model

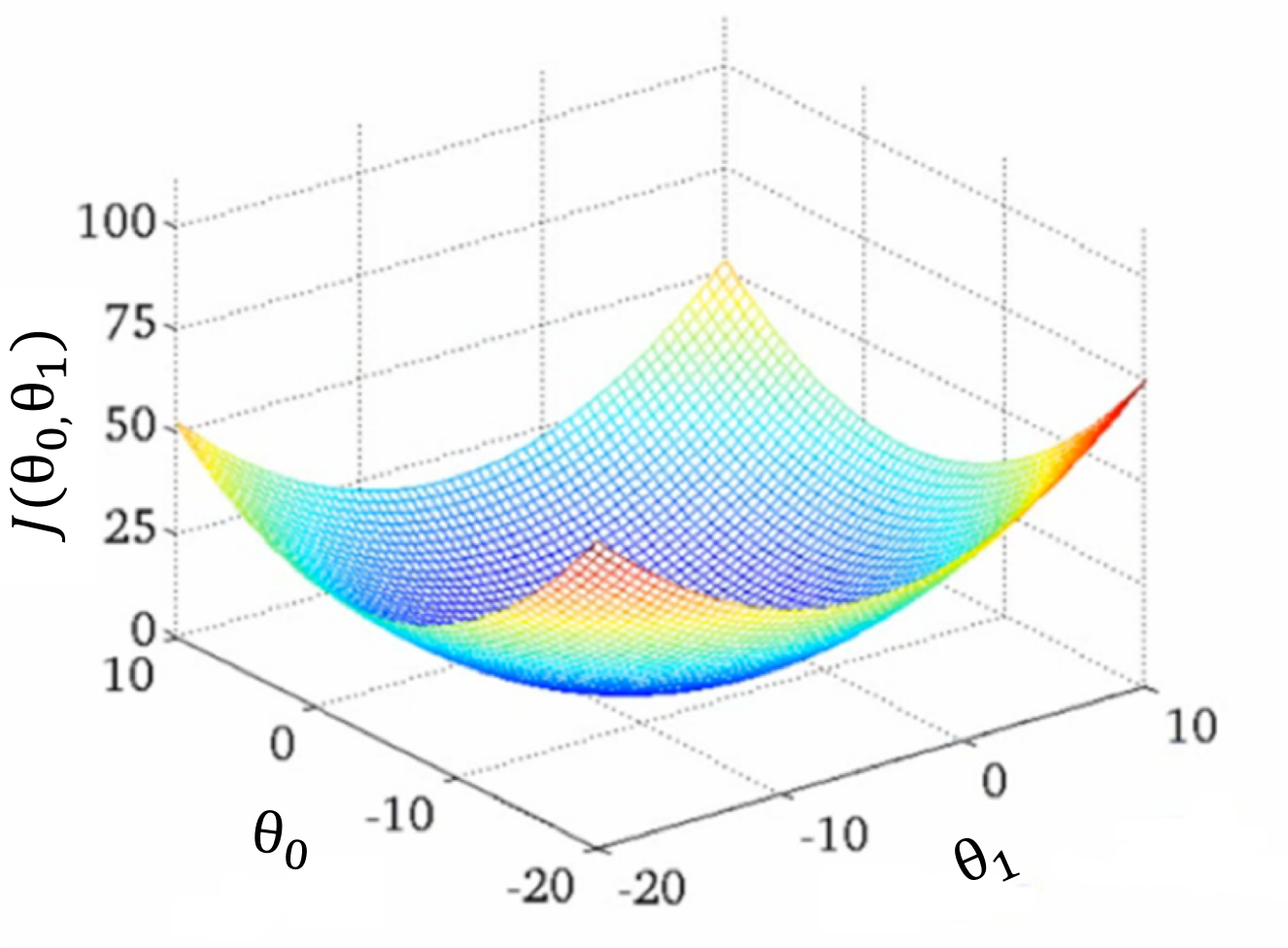
$$\hat{y} = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1$$

Cost Function

$$J(\theta) = J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

Goal: $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

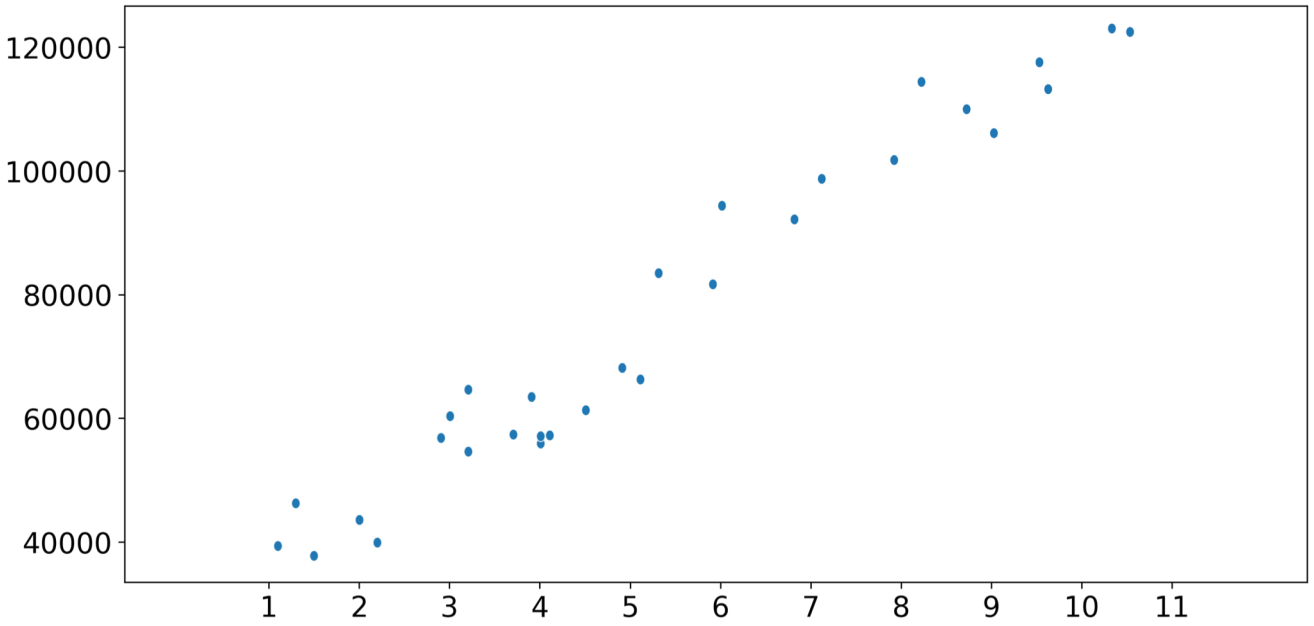
$$J(\theta_{0,}\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1 - y^{(i)})^2$$

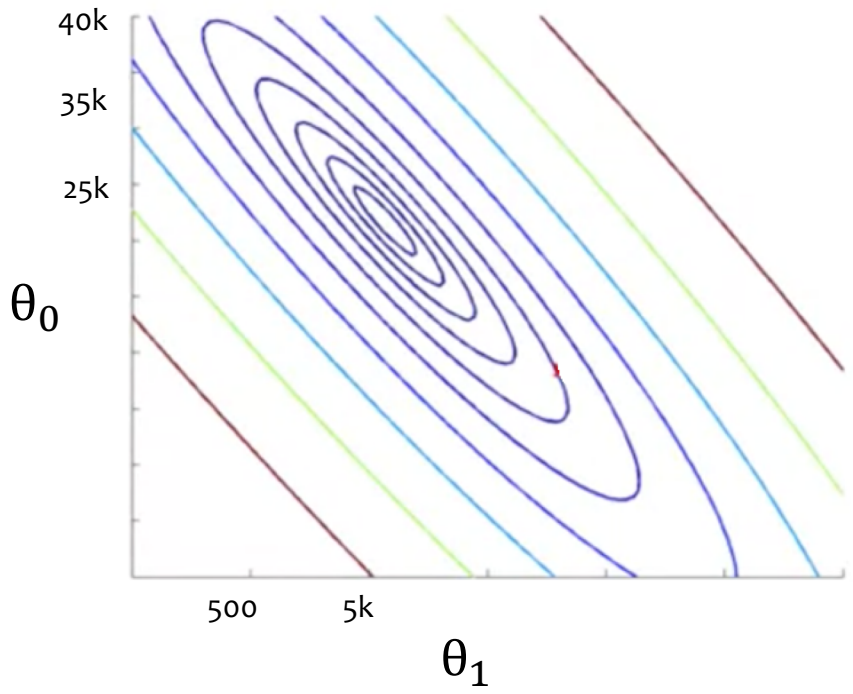


Example

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1$$

$$J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1 - y^{(i)})^2$$

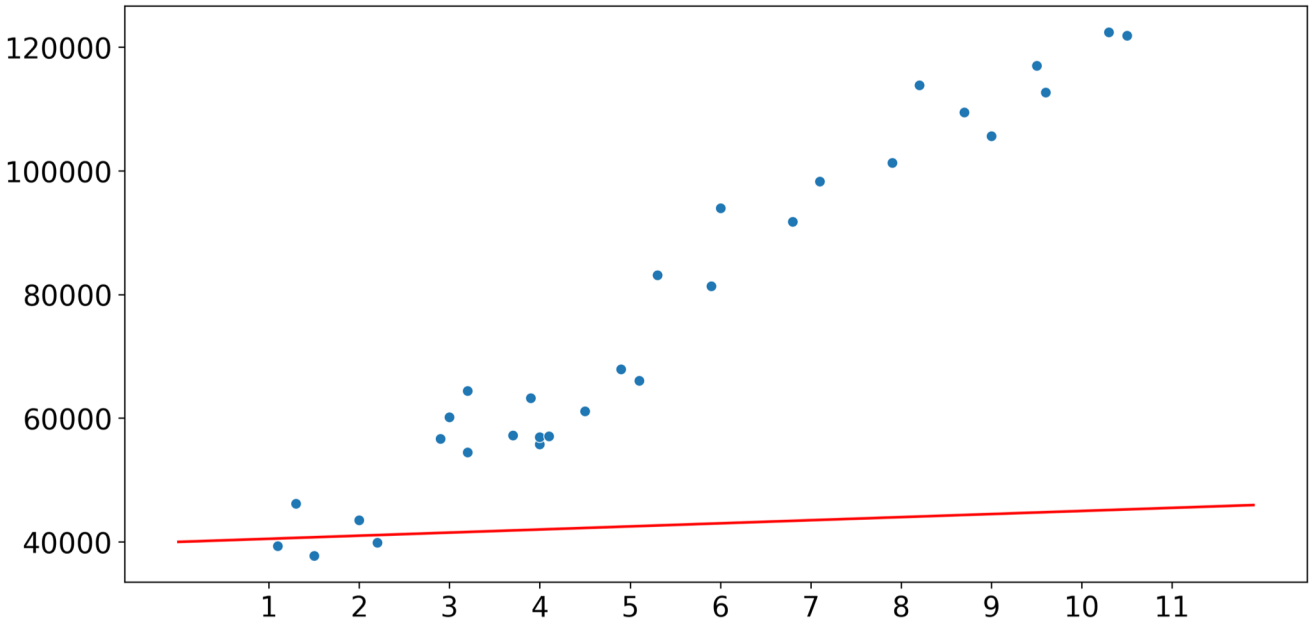


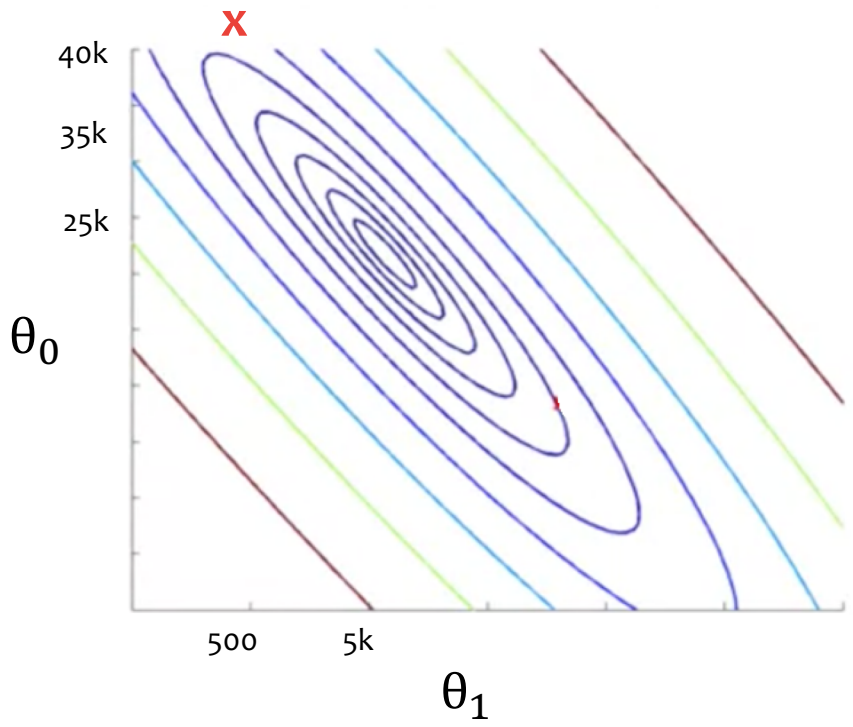


Example

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1$$

$$J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1 - y^{(i)})^2$$



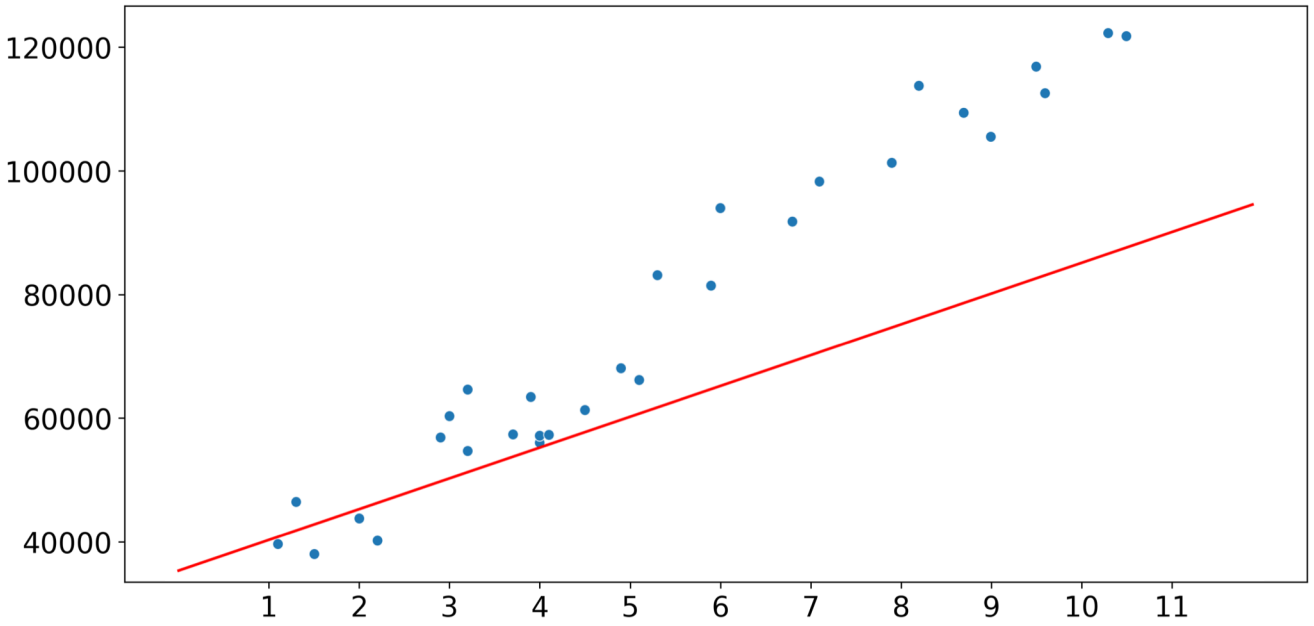


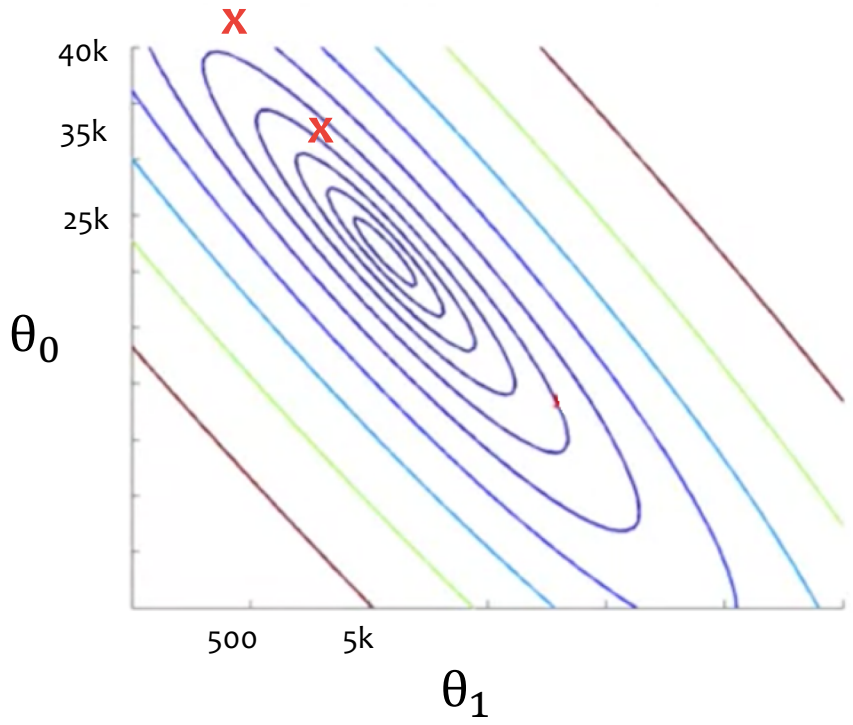
$\theta_0 = 40000$
$\theta_1 = 500$

Example

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1$$

$$J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1 - y^{(i)})^2$$



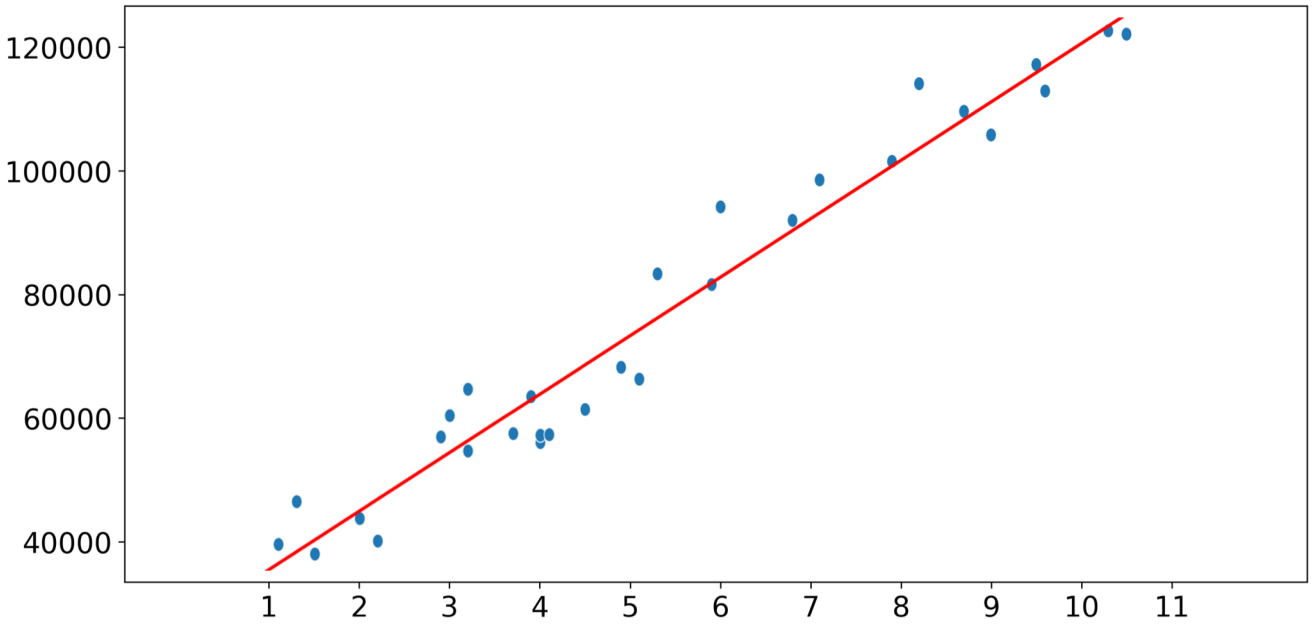


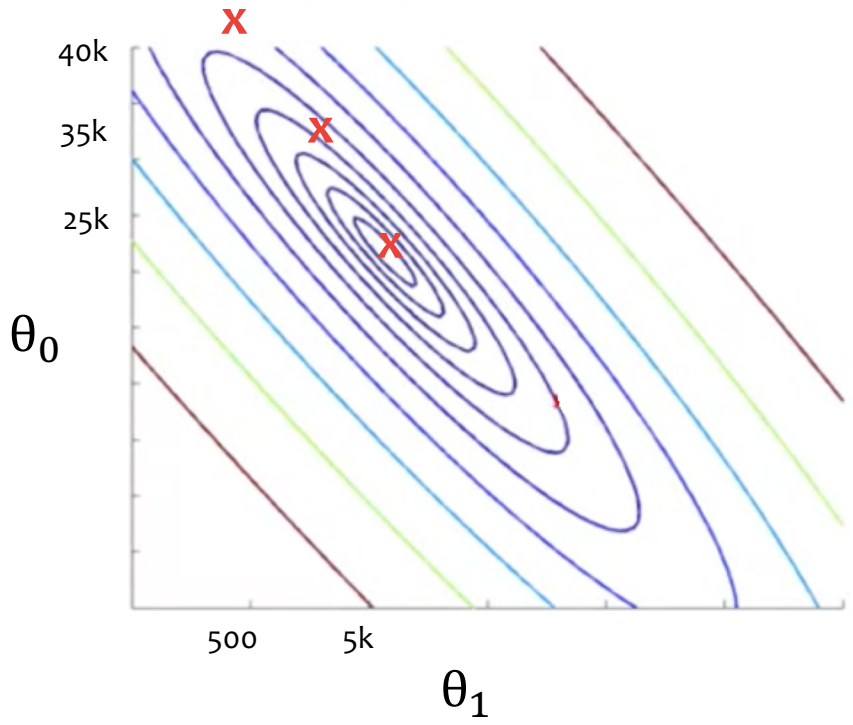
$\theta_0 = 35000$
$\theta_1 = 5000$

Example

$$\hat{y} = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1$$

$$J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1 - y^{(i)})^2$$





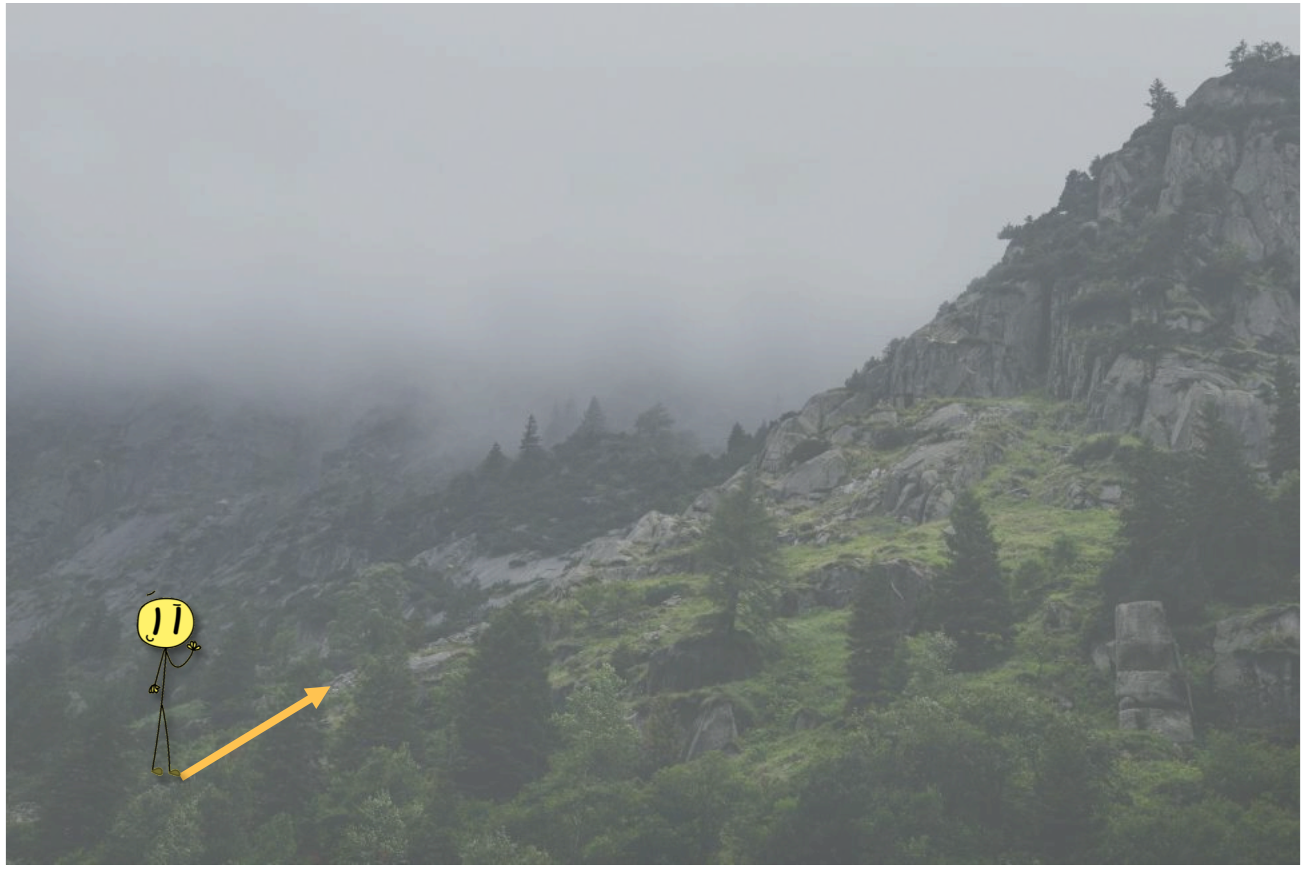
$$\theta_0 = 25000$$
$$\theta_1 = 9000$$

How to train a Linear Regression Model?

Gradient Descent

https://freerangestock.com/photos/123019/foggy-mountains-.html

Gradient Descent



https://freerangestock.com/photos/123019/foggy-mountains-.html

Gradient Descent

https://freerangestock.com/photos/123019/foggy-mountains-.html

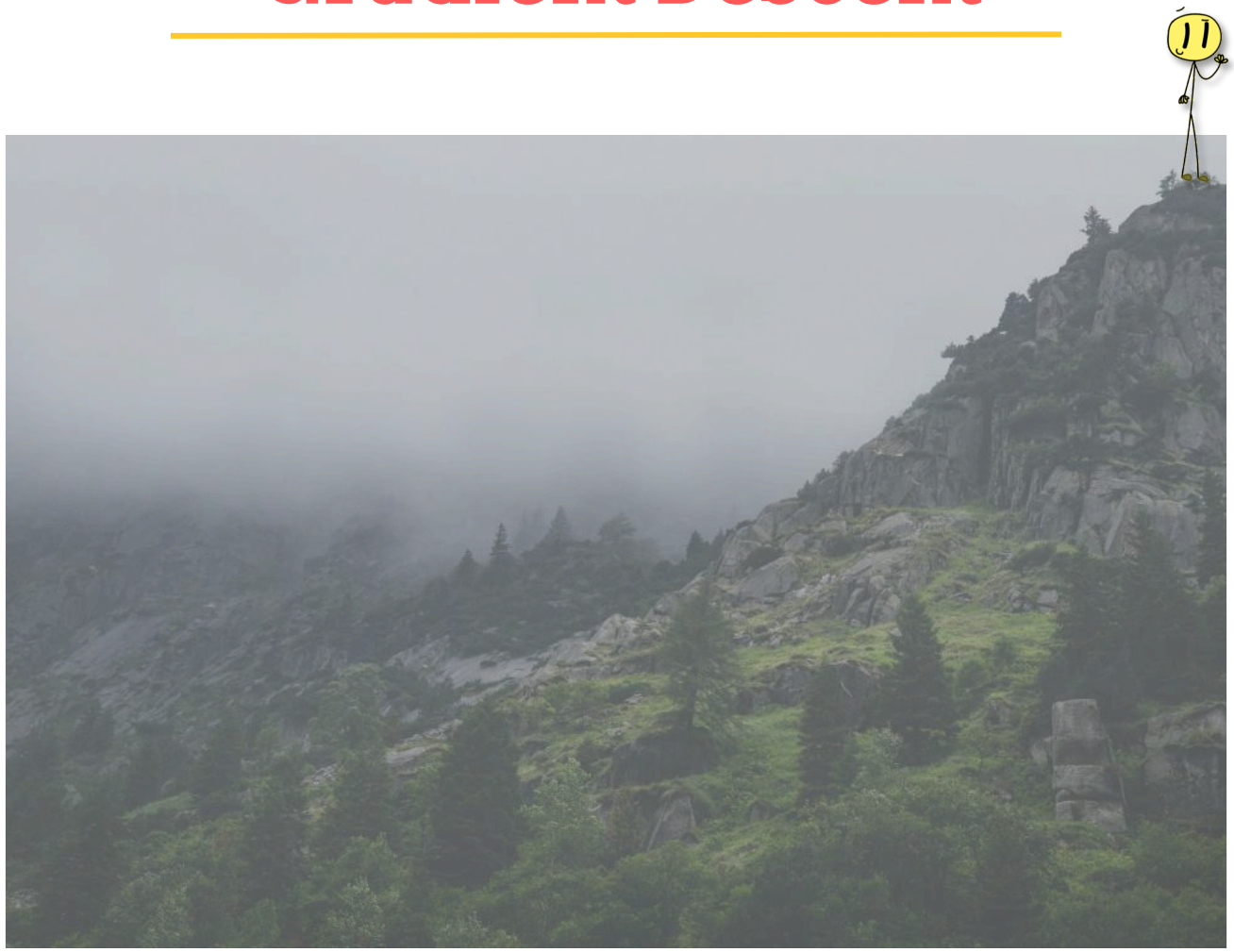
Gradient Descent

https://freerangestock.com/photos/123019/foggy-mountains-.html

Gradient Descent

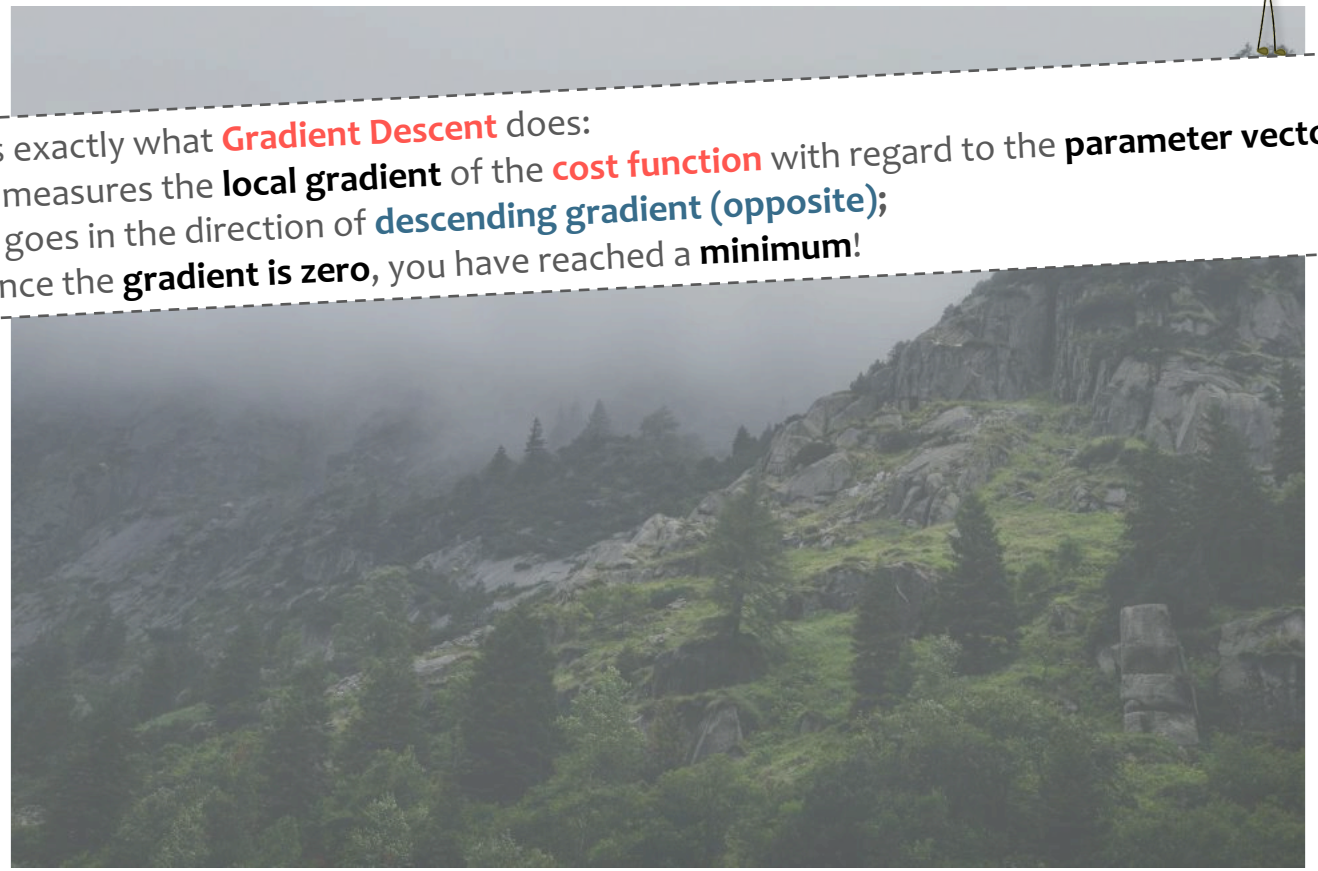
https://freerangestock.com/photos/123019/foggy-mountains-.html

Gradient Descent



https://freerangestock.com/photos/123019/foggy-mountains-.html

Gradient Descent

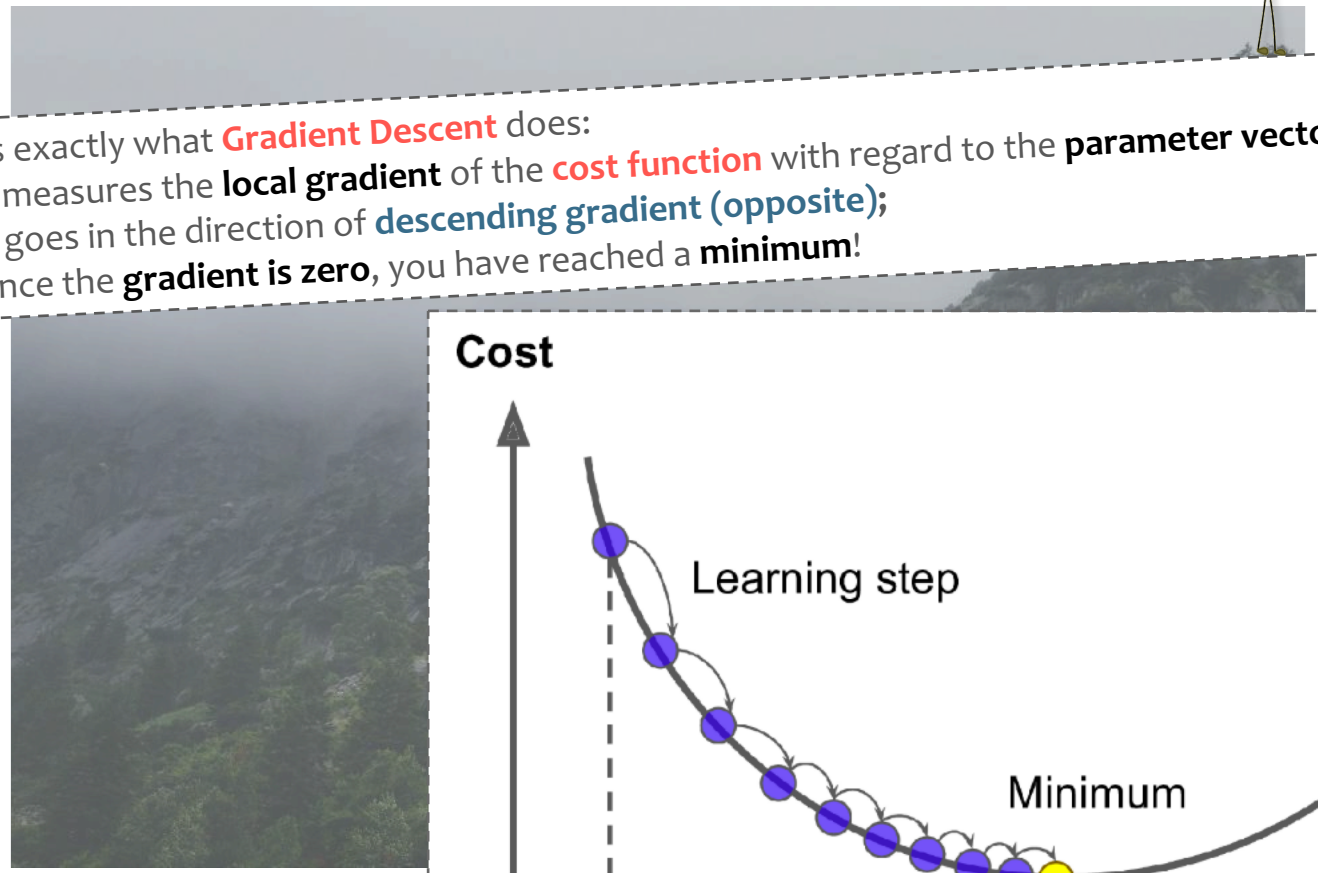


This is exactly what **Gradient Descent** does:

- it measures the **local gradient** of the **cost function** with regard to the **parameter vector θ**, and
- it goes in the direction of **descending gradient (opposite);**
- once the **gradient is zero**, you have reached a **minimum**!

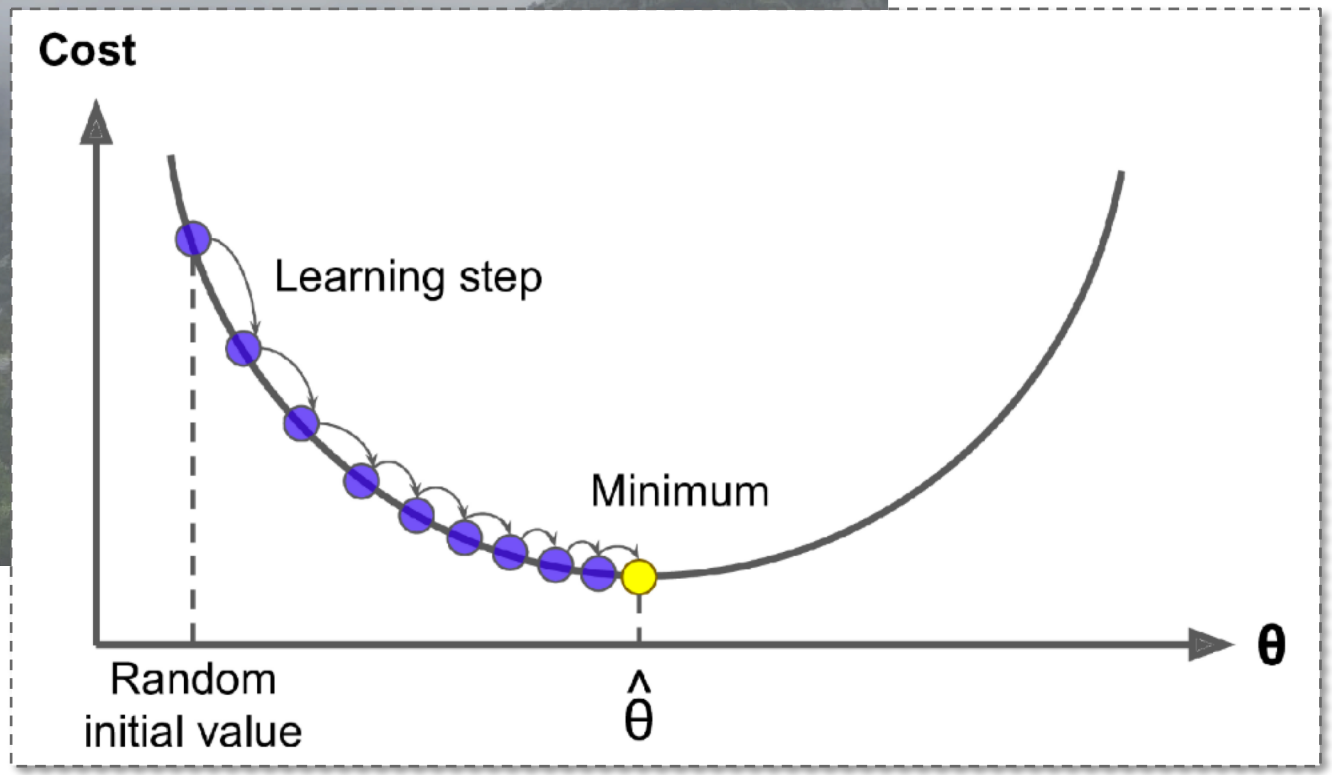
https://freerangestock.com/photos/123019/foggy-mountains-.html

Gradient Descent



This is exactly what **Gradient Descent** does:

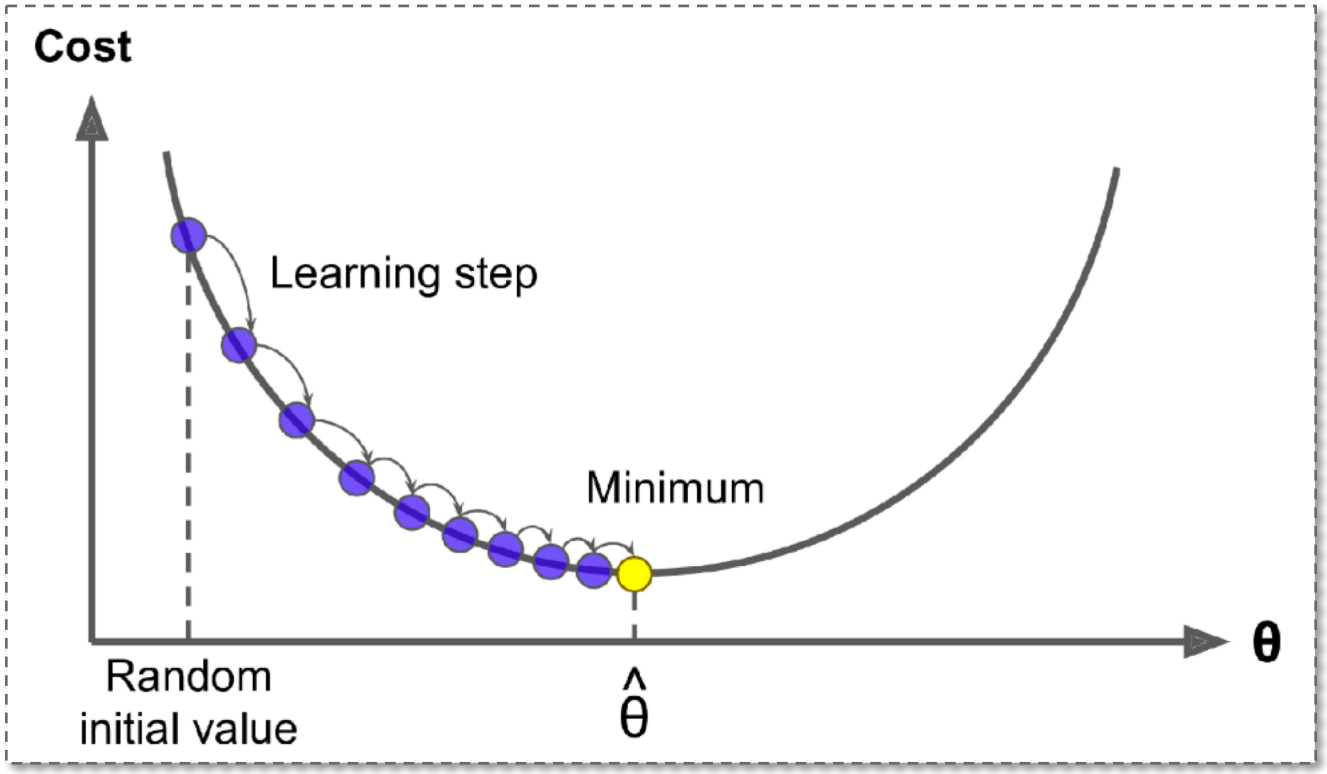
- it measures the **local gradient** of the **cost function** with regard to the **parameter vector θ**, and
- it goes in the direction of **descending gradient (opposite);**
- once the **gradient is zero**, you have reached a **minimum**!



https://freerangestock.com/photos/123019/foggy-mountains-.html

Gradient Descent

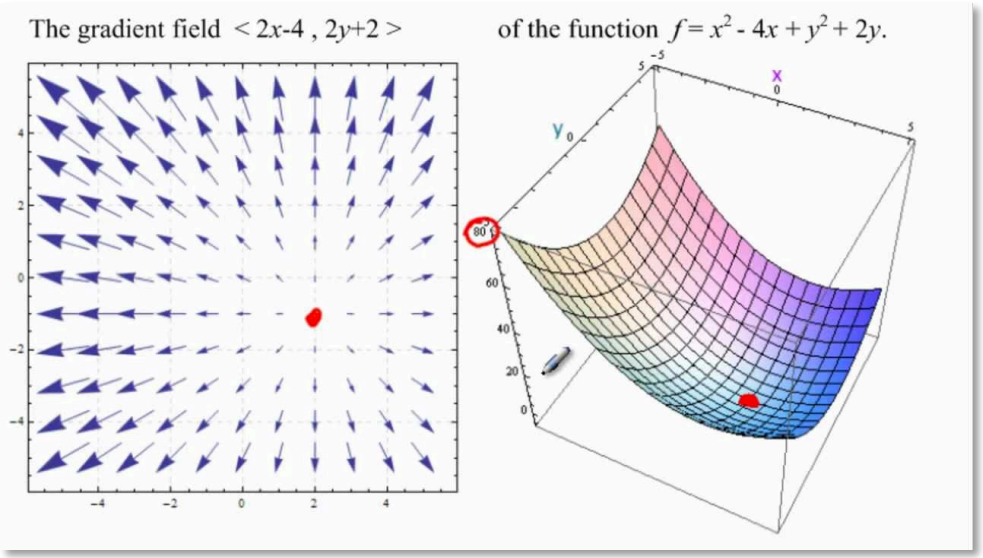
- A generic **optimization algorithm** capable of finding optimal solutions (e.g., **parameters**) to a wide range of problems.
- Its general idea is **to tweak parameters iteratively** in order **to minimize** a **cost function**.



Gradient Vector

- The **gradient vector**, or simply **gradient**, is a vector that points to **the direction of greatest increase** with a given **increase rate**, for any function points.
- It is formed by the **partial derivative** of each variable in the function.

$$\nabla f = \begin{bmatrix} \frac{\partial}{\partial x} f \\ \frac{\partial}{\partial y} f \\ \vdots \end{bmatrix} \qquad \nabla J(\theta) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial \theta_1} J(\theta) \\ \vdots \end{bmatrix}$$



Algorithm

Given some cost function: $J(\theta) = J(\theta_0, \theta_1, \cdots, \theta_n)$

Algorithm parameter: learning rate $\alpha \in (0, 1]$

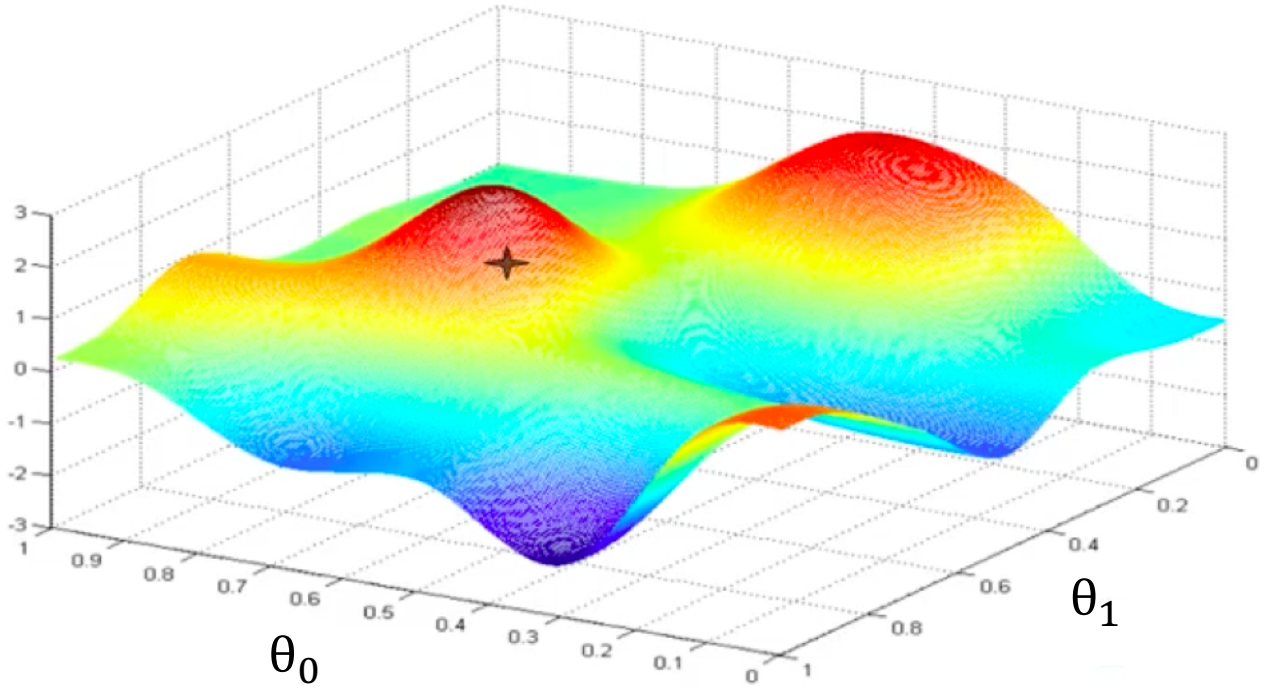
Goal: $\min_{\theta_0, \theta_1, \cdots, \theta_n} J(\theta_0, \theta_1, \cdots, \theta_n)$

Algorithm:

1. Initialize $\theta_0, \theta_1, \cdots, \theta_n$ arbitrarily (*e.g.*, $\theta_0 = 0, \theta_1 = 0, \cdots, \theta_n = 0$)
2. **repeat until convergence** (*e.g., for a given number of epochs/iterations):*
3. $\hat{y}$ = make predictions
4. Compute $J(\theta_0, \theta_1, \cdots, \theta_n)$
5. Change $\theta_0, \theta_1, \cdots, \theta_n$ **to reduce** $J(\theta_0, \theta_1, \cdots, \theta_n)$

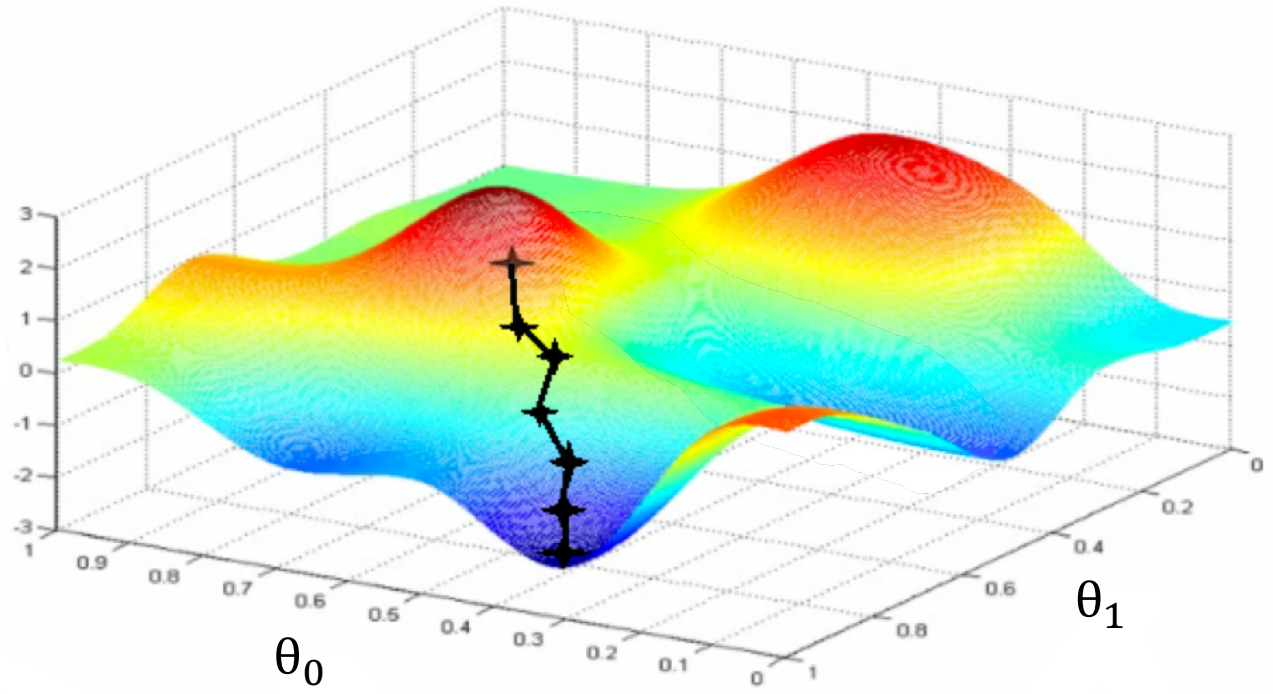
Algorithm

$J(\theta_0, \theta_1)$



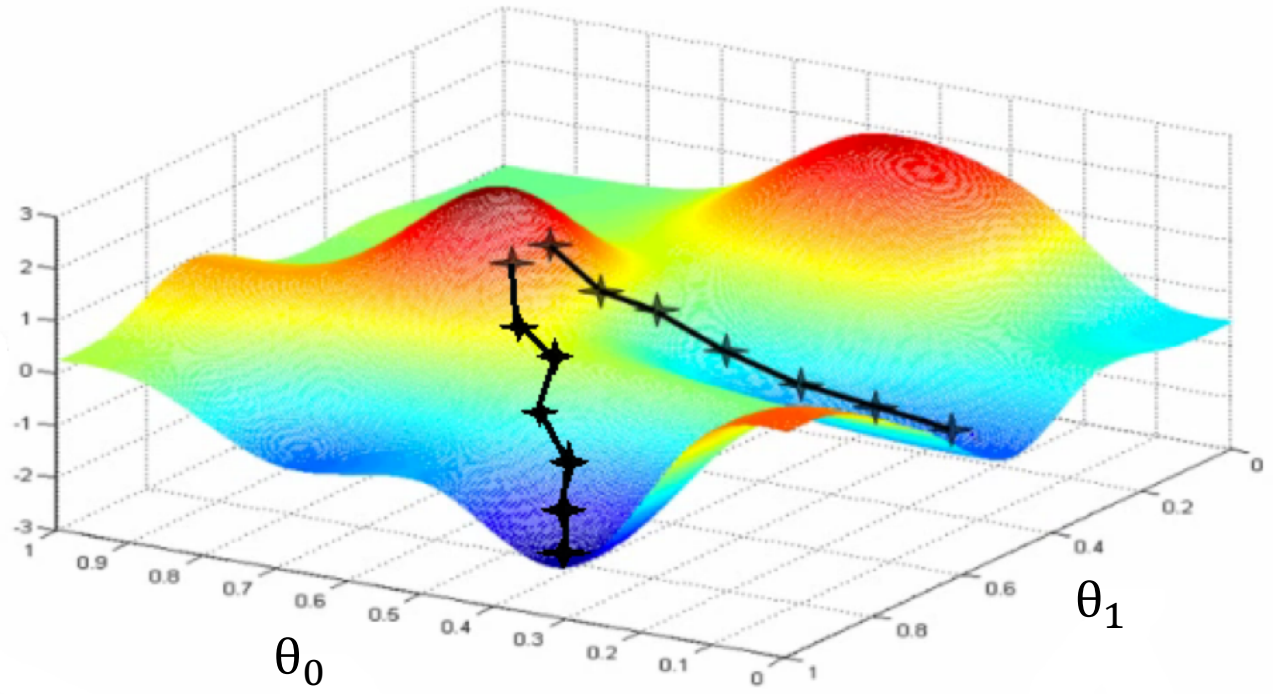
Algorithm

$J(\theta_0, \theta_1)$



Algorithm

$J(\theta_0, \theta_1)$



Algorithm

learning rate

for a given number of epochs/iterations

repeat until convergence {

$$temp_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$temp_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

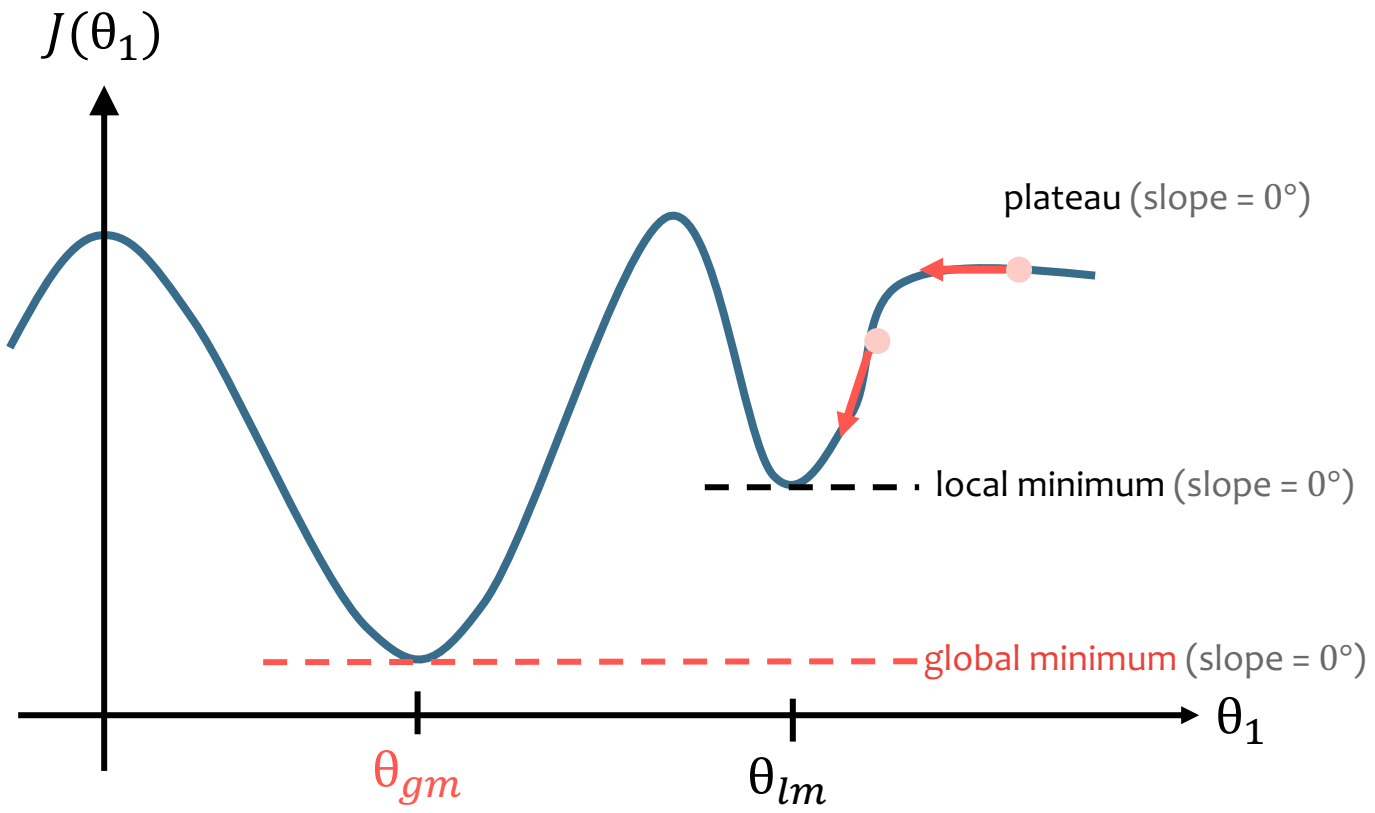
$$\theta_0 = temp_0$$

$$\theta_1 = temp_1$$

}

Local and global minima

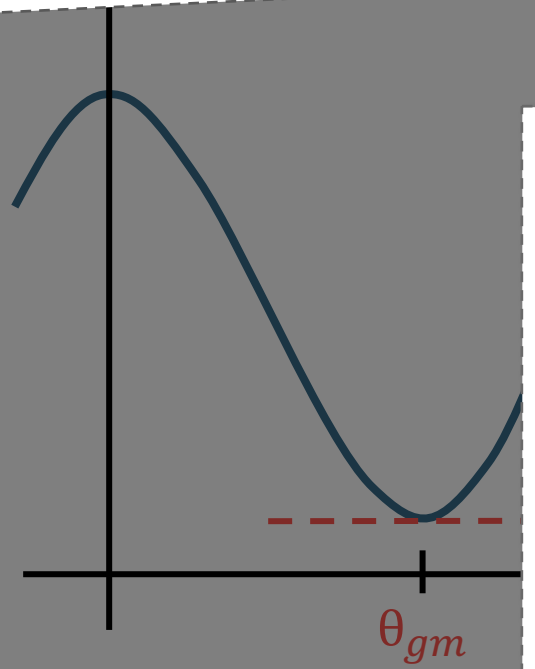
- **Gradient descent** approach usually presents some pittfals like local **minimum** or **plateaus**.



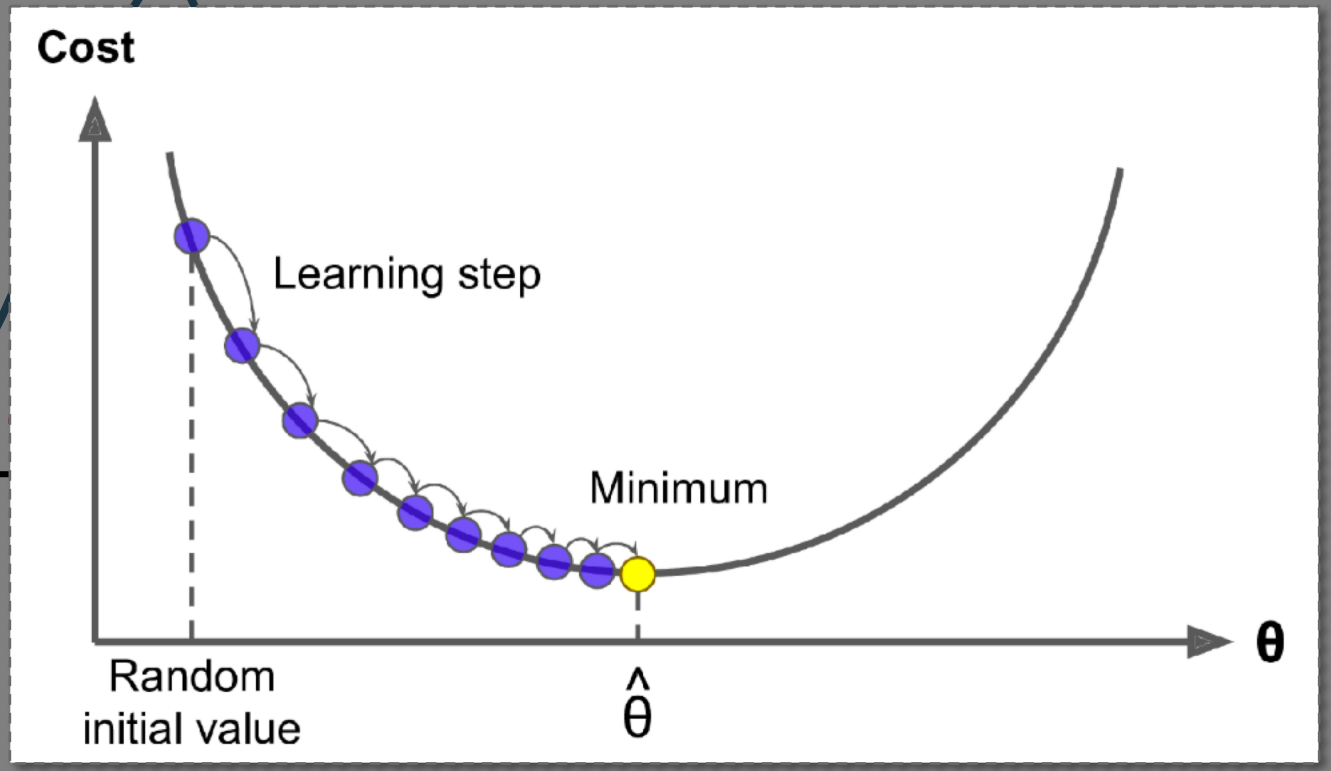
Local and global minima

- **Gradient descent** approach usually presents some pitfals like local **minima** and **plateaus**.

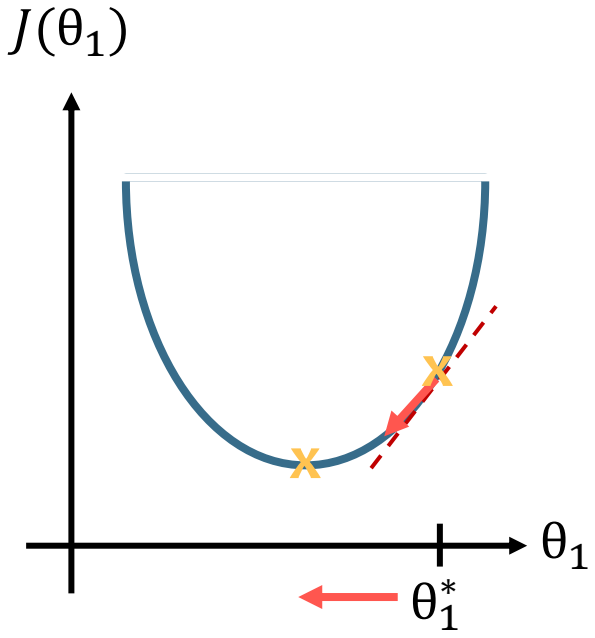
Fortunately, the **MSE** cost function for a Linear Regression is a **convex functions.**
This means that:
- There are no local minima, **just the global minimum**;
- Function never changes abruptly.



plateau (slope = 0°)

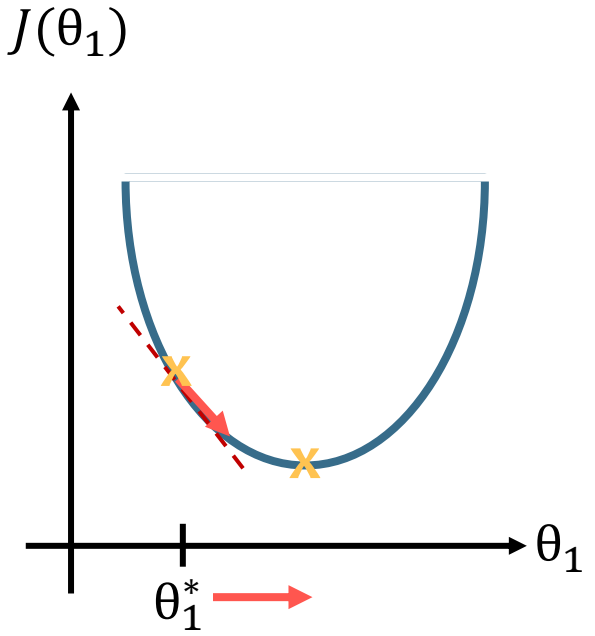


Updates



$$\theta_1 = \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)}$$

≥ 0



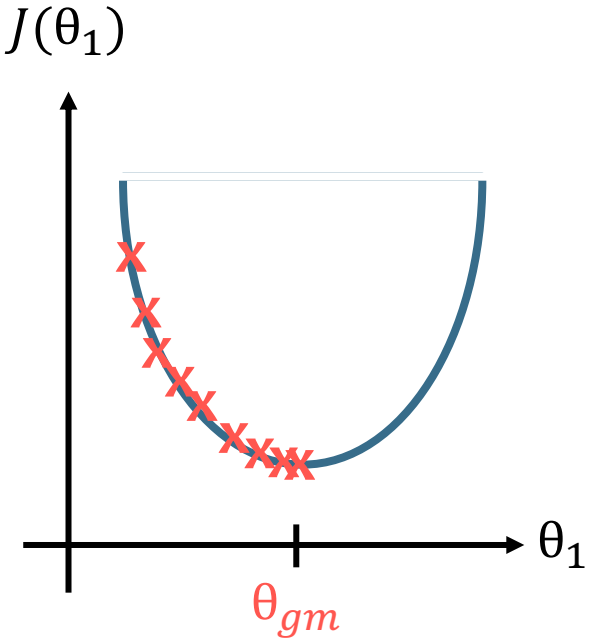
$$\theta_1 = \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)}$$

≤ 0

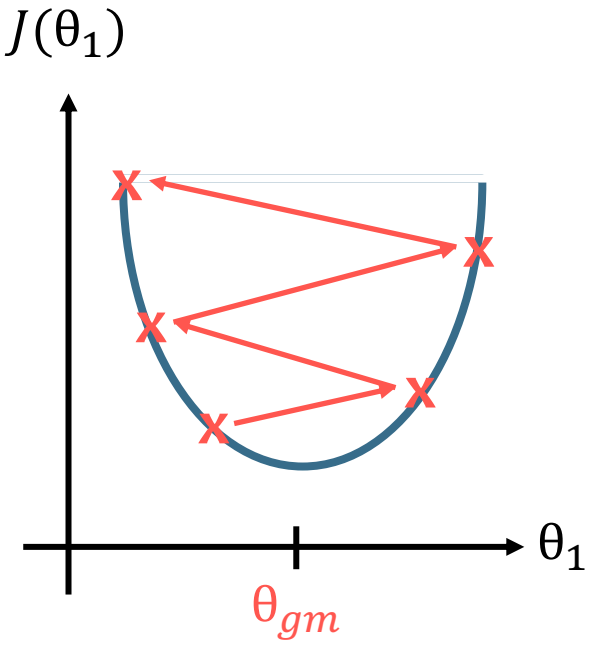
Learning rate

$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is **too small**, gradient descent can be slow.



If α is **too large**, gradient descent can be overshoot the minimum, fail to converge, or even diverge.



Implementation Strategies for GD

Batch Gradient Descent

- Each iteration of the gradient descent (GD) training uses **all samples** of the **training set X**:

Algorithm:

1. Initialize $\theta_0, \theta_1, \cdots, \theta_n$ arbitrarily (*e.g.*, $\theta_0 = 0, \theta_1 = 0, \cdots, \theta_n = 0$)
2. **repeat until convergence**: *// (e.g., for a given number of epochs/iterations)*
3. $\hat{y} = h_\theta(\mathbf{X})$ *// predict all samples in* $\boldsymbol{X}$
4. Compute $J(\theta_0, \theta_1, \cdots, \theta_n)$ *// based on* y *and* $\hat{y}$
5. Change $\theta_0, \theta_1, \cdots, \theta_n$ **to reduce** $J(\theta_0, \theta_1, \cdots, \theta_n)$

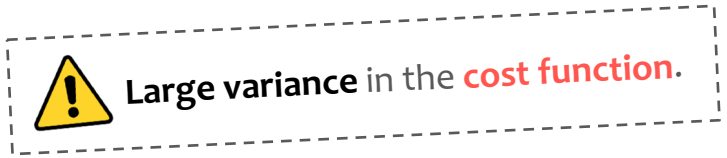
Impractical for **large training set.**

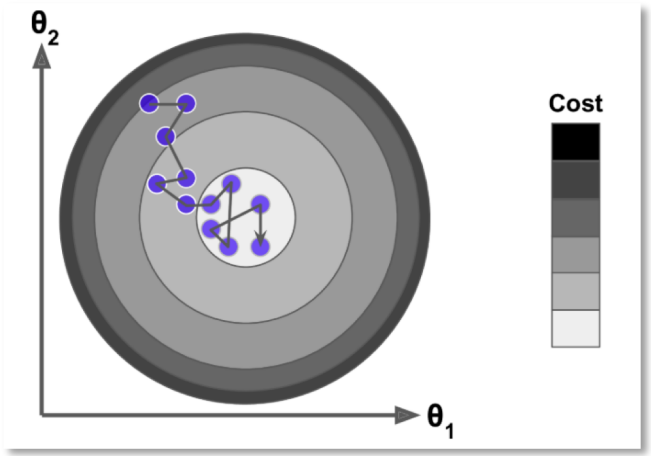
Stochastic Gradient Descent (SGD)

- Each iteration of the GD training uses **only one sample** of the **training set** chosen **randomly**.

Algorithm:

1. Initialize $\theta_0, \theta_1, \cdots, \theta_n$ arbitrarily (*e.g.*, $\theta_0 = 0, \theta_1 = 0, \cdots, \theta_n = 0$)
2. **repeat until convergence**: *// (e.g., for a given number of epochs/iterations)*
3. Shuffle $\langle \mathbf{X}, \boldsymbol{y} \rangle$
4. for each sample $\boldsymbol{x}^{(i)}$ in shuffled $\mathbf{X}$:
5. $\hat{y}^{(i)} = h_\theta(\boldsymbol{x}^{(i)})$ *// predict* $\boldsymbol{x}^{(i)}$
6. Compute $J(\theta_0, \theta_1, \cdots, \theta_n)$ *// based on shuffled* y *and* $\hat{y}$
7. Change $\theta_0, \theta_1, \cdots, \theta_n$ **to reduce** $J(\theta_0, \theta_1, \cdots, \theta_n)$





Mini-batch Gradient Descent

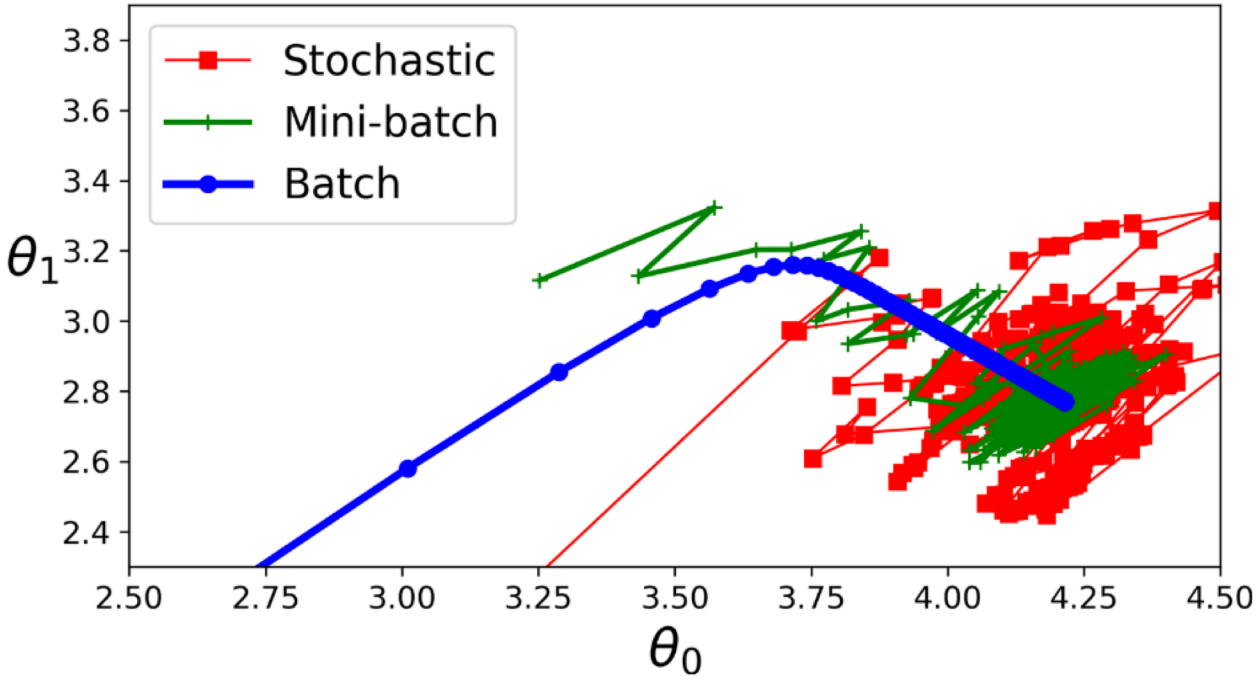
- Each iteration of the GD training uses k (min-batch size) samples of the training set chosen **randomly**.

Algorithm:

1. Initialize $\theta_0, \theta_1, \cdots, \theta_n$ arbitrarily (*e.g.*, $\theta_0 = 0, \theta_1 = 0, \cdots, \theta_n = 0$)
2. **repeat until convergence**: *// (e.g., for a given number of epochs/iterations)*
3. for each batch iteration: *// based on the batch size and training set size*
4. $\mathbf{X_b}, y_b$ = draw **k random samples** with replacement in $\langle \mathbf{X}, \boldsymbol{y} \rangle$
5. $\hat{y}_b = h_\theta(\mathbf{X_b})$ *// predict the mini batch* $\mathbf{X_b}$
6. Compute $J(\theta_0, \theta_1, \cdots, \theta_n)$ *// based on* y_b *and* $\hat{y}_b$
7. Change $\theta_0, \theta_1, \cdots, \theta_n$ **to reduce** $J(\theta_0, \theta_1, \cdots, \theta_n)$

It requires an additional **"mini-batch size"** hyperparameter for the learning algorithm.

Convergence



Closed-form solutions to train a Linear Regressor Model

Normal Equation

- It is an analytical approach to Linear Regression to find out the parameters $\boldsymbol{\theta}$.
- In other words, it is a mathematical equation that gives the result directly.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j}J(\theta) = \cdots = \mathbf{0}, \text{ solve for } \theta_0, \theta_1, \ldots, \theta_n$$

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

not always *invertible*

SVD strategy

- $\hat{\theta} = X^{+} y$
- Strategy that uses the Singular Value Decomposition (SVD) to compute a pseudoinverse matrix to find the parameters $\boldsymbol{\theta}$.

Normal Equation

- It is an analytical approach to Linear Regression to find out the parameters $\boldsymbol{\theta}$.
- In other words, it is a mathematical equation that gives the result directly.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = \mathbf{0}, \text{ so } \theta_0, \theta_1, \ldots, \theta_n$$

The Normal Equation and the SVD approach get **very slow** when **the number of features grows large** (e.g., 100,000).

$$\hat{\theta} = (\mathrm{X}^T \mathrm{X})^{-1}\ \mathrm{X}^T y$$

not always *invertible*

On the positive side, both are **linear** with regard to the number of **training instances** (they are O(m)), so **they handle large training sets efficiently**, provided they can fit in memory

SVD strategy

- $\hat{\theta} = \mathrm{X}^{+} y$
- Strategy that uses the Singular Value Decomposition (SVD) to compute a pseudoinverse matrix to find the parameters $\boldsymbol{\theta}$.

There is almost **no difference** after **training**:
all these algorithms end up with very **similar models**
and **make predictions** in exactly **the same way**.

Table 4-1. Comparison of algorithms for Linear Regression

Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor` ????
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor` ????

To check manual implementations of those algorithms, we refer to this GitHub repository.

Aprendizado de Máquina e Reconhecimento de Padrões 2021.2

Linear Regression and Gradient Descent

Prof. Dr. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas