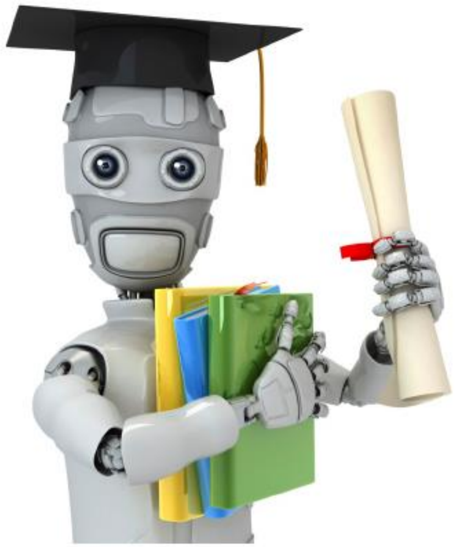
Introduction to ML

Week 2

“aimed at getting you to kickass in AI”

Agenda

1. Welcome
2. What is ML?
3. Supervised Learning
4. Model Representation
5. Cost function
6. Gradient Descent



Machine Learning

Buzz Words

Features

1st order polynomial

Straight-line

Discrete value

Regression problem

Machine Learning

2nd order polynomial

Continuous values

What is Machine Learning

According to Arthur Samuel in 1959, Machine Learning gives “**computers the ability to learn without being explicitly programmed.**”

- **Arthur Samuel**

Learning is any process by which a system improves performance from experience - “ML is concerned with computer programs that automatically improve their performance through experience”- **Herbert Simon**

What is Machine Learning

A computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. - **Tom Mitchell**

Learning = Improving with experience at some task.

- Improve over task, T(**classification**)
- With performance measure i.e. accuracy, P(**correctly classify orange/apple**)
- Based on experience , E(**watching you label fruits into orange/apple**).

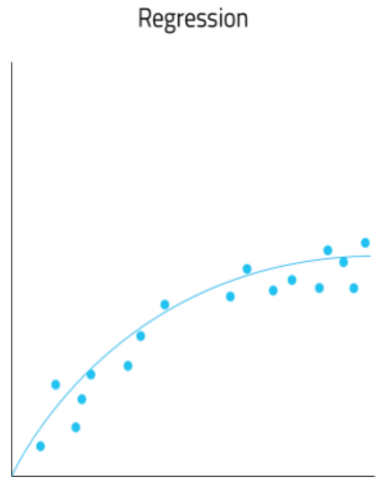
Classification of Machine Learning

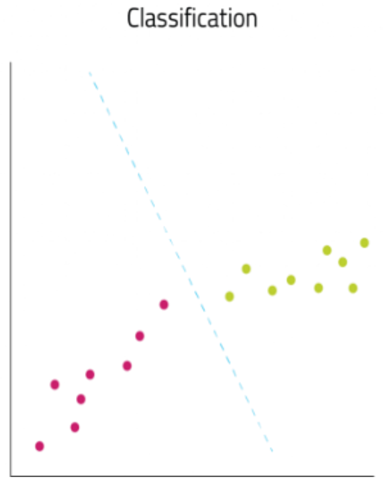
Supervised Learning:

Predicting values. **Known** targets.
User inputs correct answers to learn from. Machine uses the information to guess new answers.

REGRESSION:
Estimate continuous values
(Real-valued output)

CLASSIFICATION:
Identify a unique class
(Discrete values, Boolean, Categories)





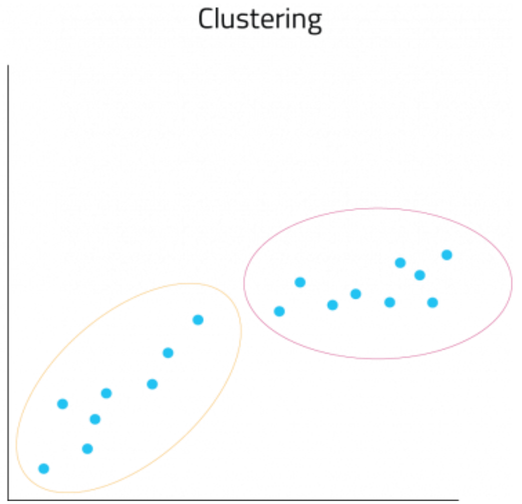
Unsupervised Learning:

Search for structure in data. **Unknown** targets.
User inputs data with undefined answers. Machine finds useful information hidden in data.

Cluster Analysis
Group into sets

Density Estimation
Approximate distributions

Dimension Reduction
Select relevant variables



Others: RL

Linear Regression Basic

- **Linear regression** allows us to understand relationship between two continuous variables.
- X: independent variable e.g. room size, security, road network(Lekki, Ipaja)
- Y: dependent variable e.g. cost of housing

$$Y = mX + b$$

X1: room size

Regression: predict continuous valued output (price)

Linear Regression Basic - Supervised Learning (Housing Price)

Label	X(Housing state, security, light)	Y(Housing cost)
1.		N5,000,000
2.		**???**
3.		N25,000,000

Fun Illustration

Linear Regression Basic – Supervised Learning (Housing Price)

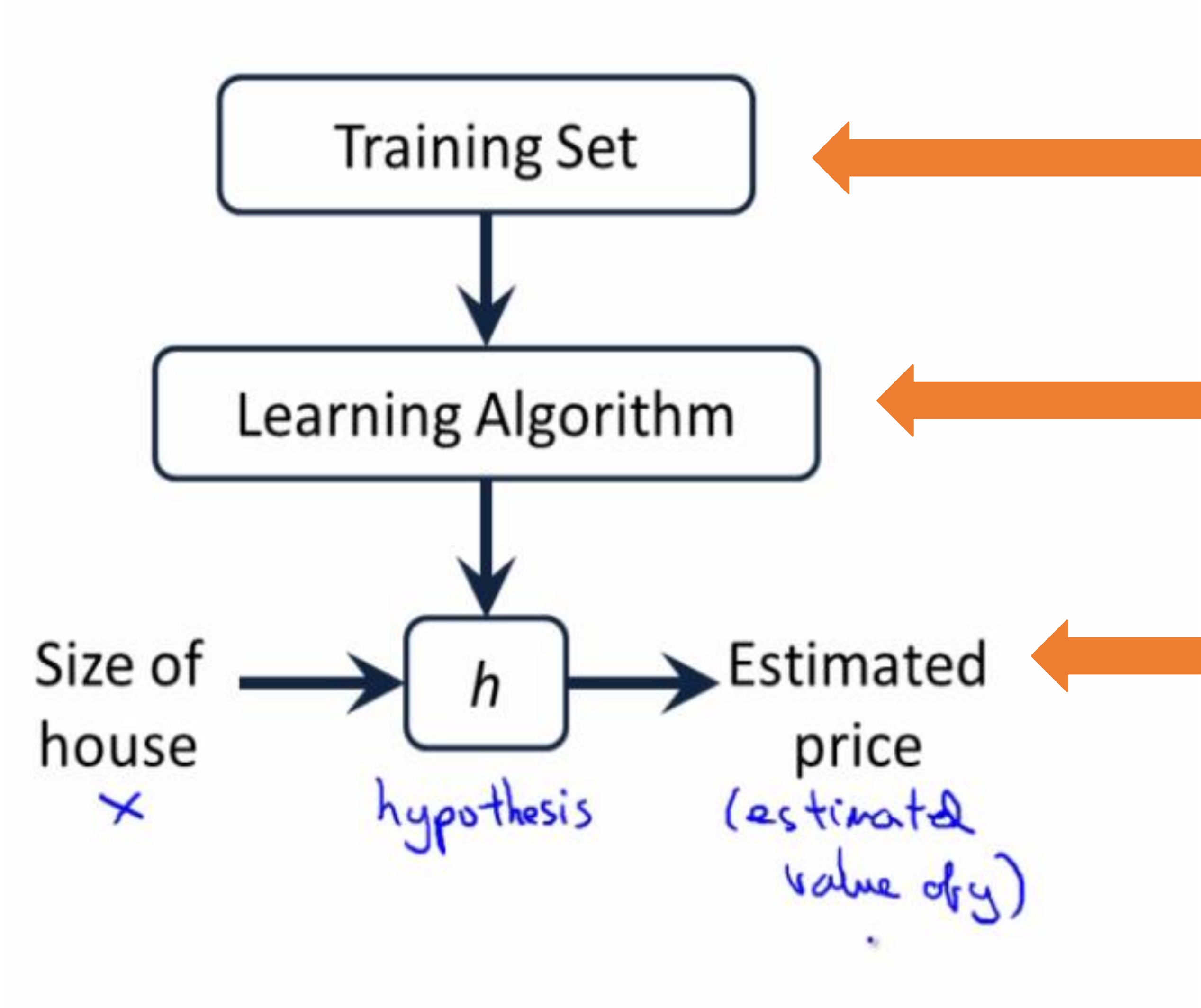
Training set of housing prices (Portland, OR)

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's = "input" variable / features
y's = "output" variable / "target" variable

Linear Regression Basic – Model Representation

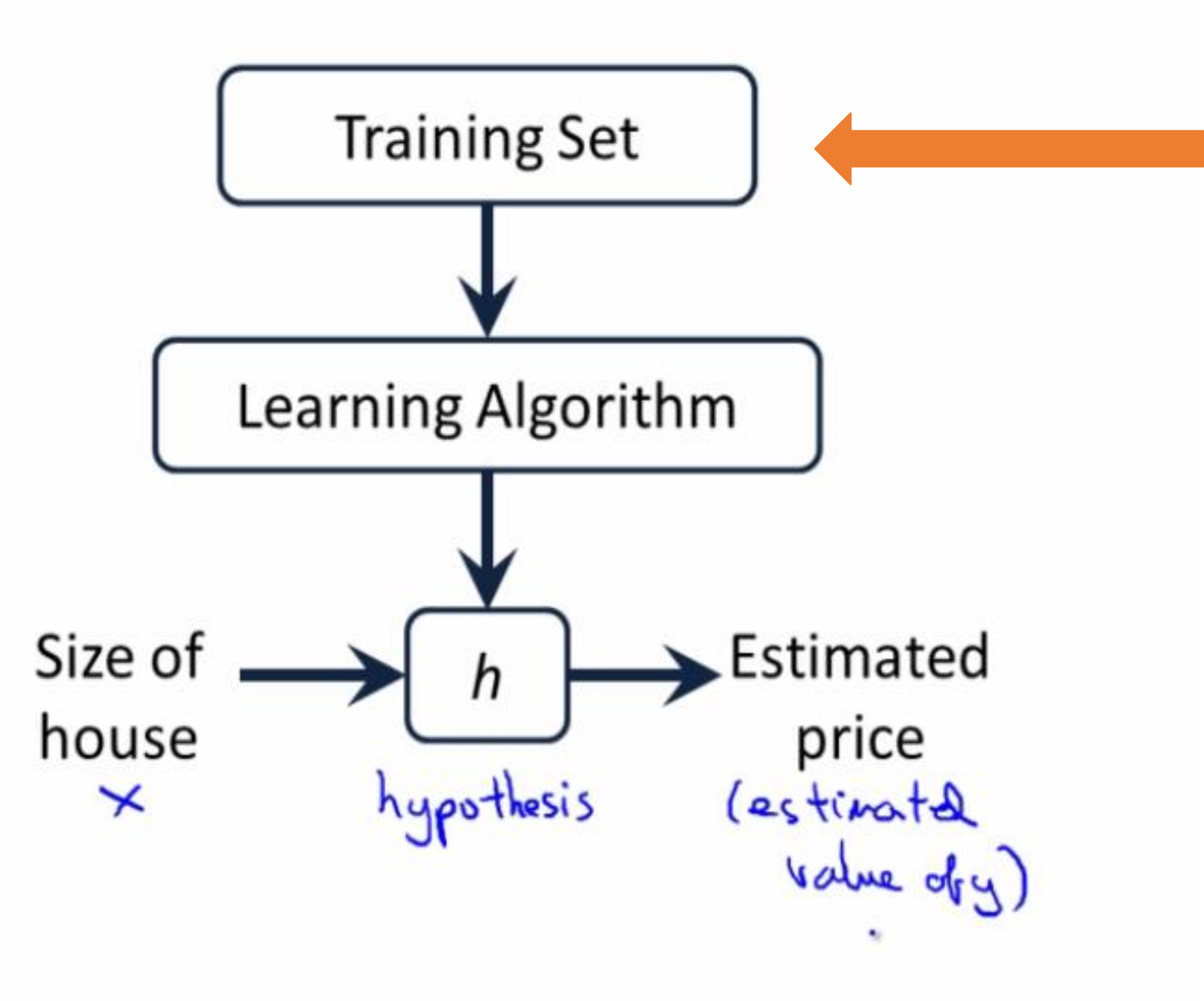


PART 1

PART 3

PART 2

Linear Regression Basic – Model Representation



PART 1

Linear Regression Basic – Model Representation

Training set of housing prices (Portland, OR)

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

$m = 47$

Notation:

- **m** = Number of training examples
- **x**'s = "input" variable / features
- **y**'s = "output" variable / "target" variable

(x, y) – one training example

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

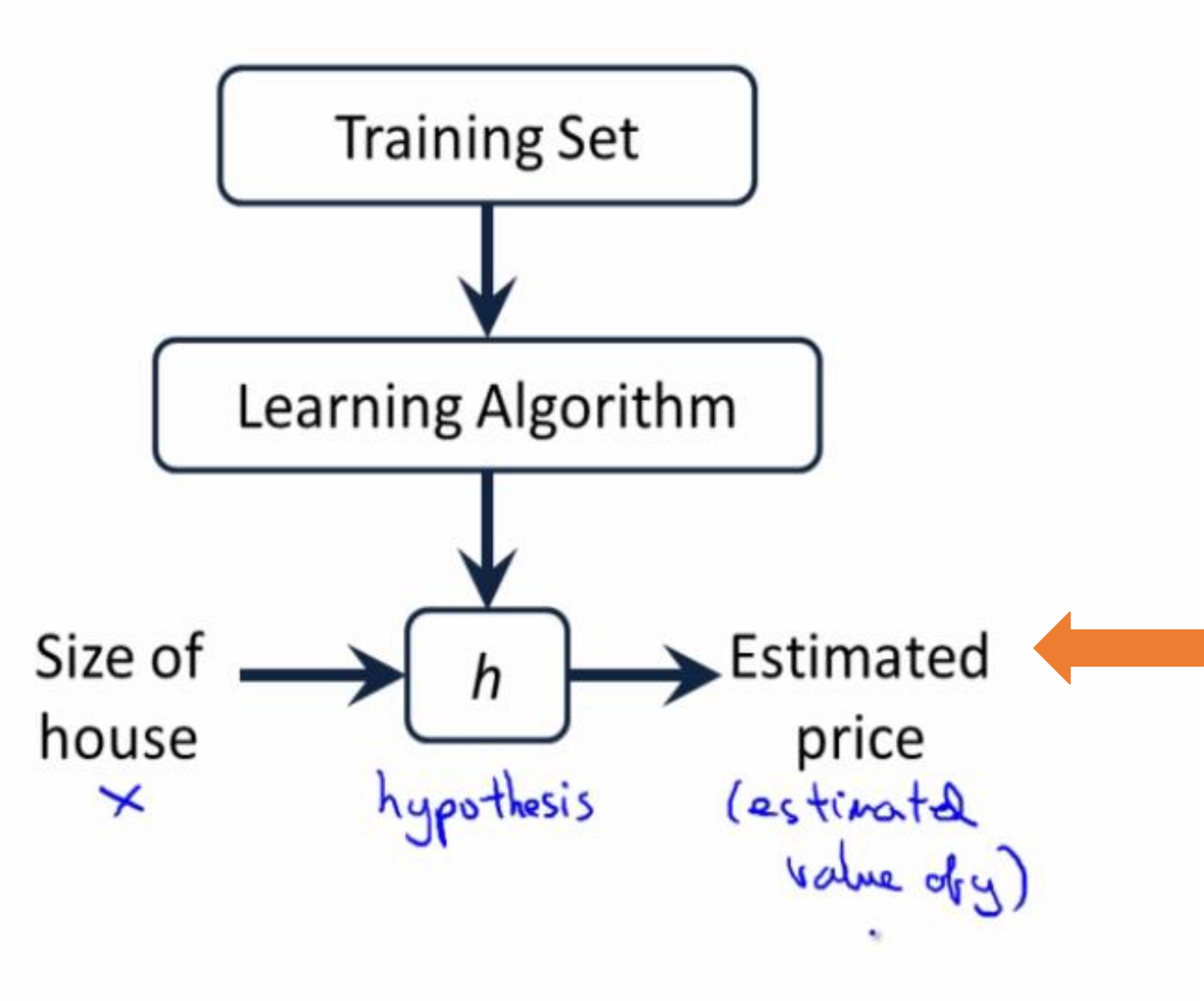
$x^{(1)} = 2104$

$x^{(2)} = 1416$

$y^{(1)} = 460$

AI6

Linear Regression Basic – Model Representation



PART 2

Linear Regression Basic – Model Representation

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

$m = 47$

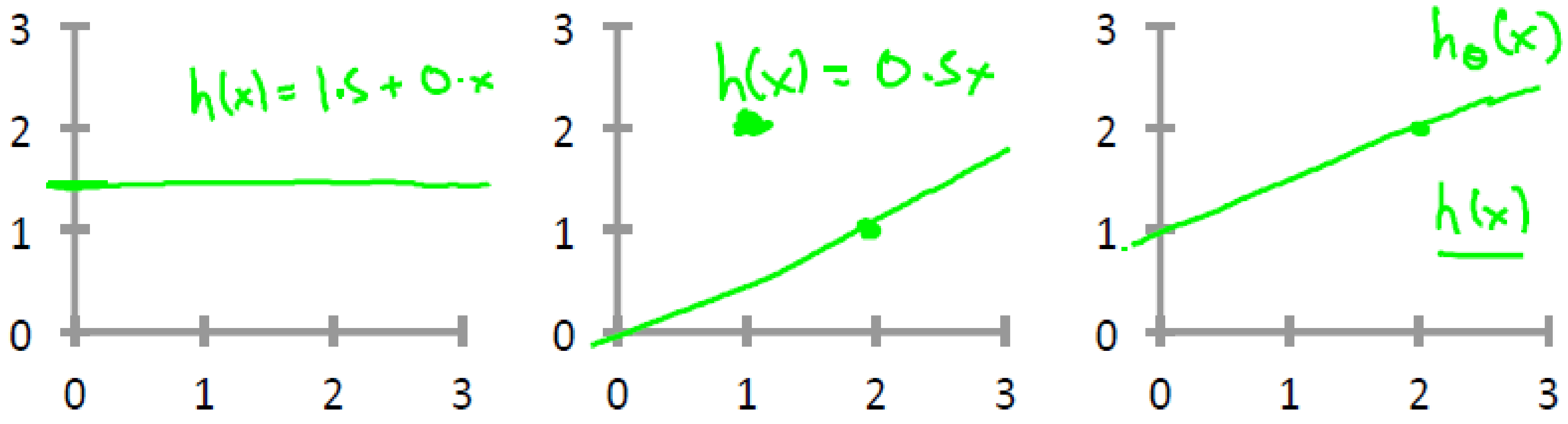
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's: Parameters

How to choose θ_i's ?

Linear Regression Basic – Model Representation

$$h_\theta(x) = \theta_0 + \theta_1 x$$



→ $\theta_0 = 1.5$	→ $\theta_0 = 0$	→ $\theta_0 = 1$
→ $\theta_1 = 0$	→ $\theta_1 = 0.5$	→ $\theta_1 = 0.5$

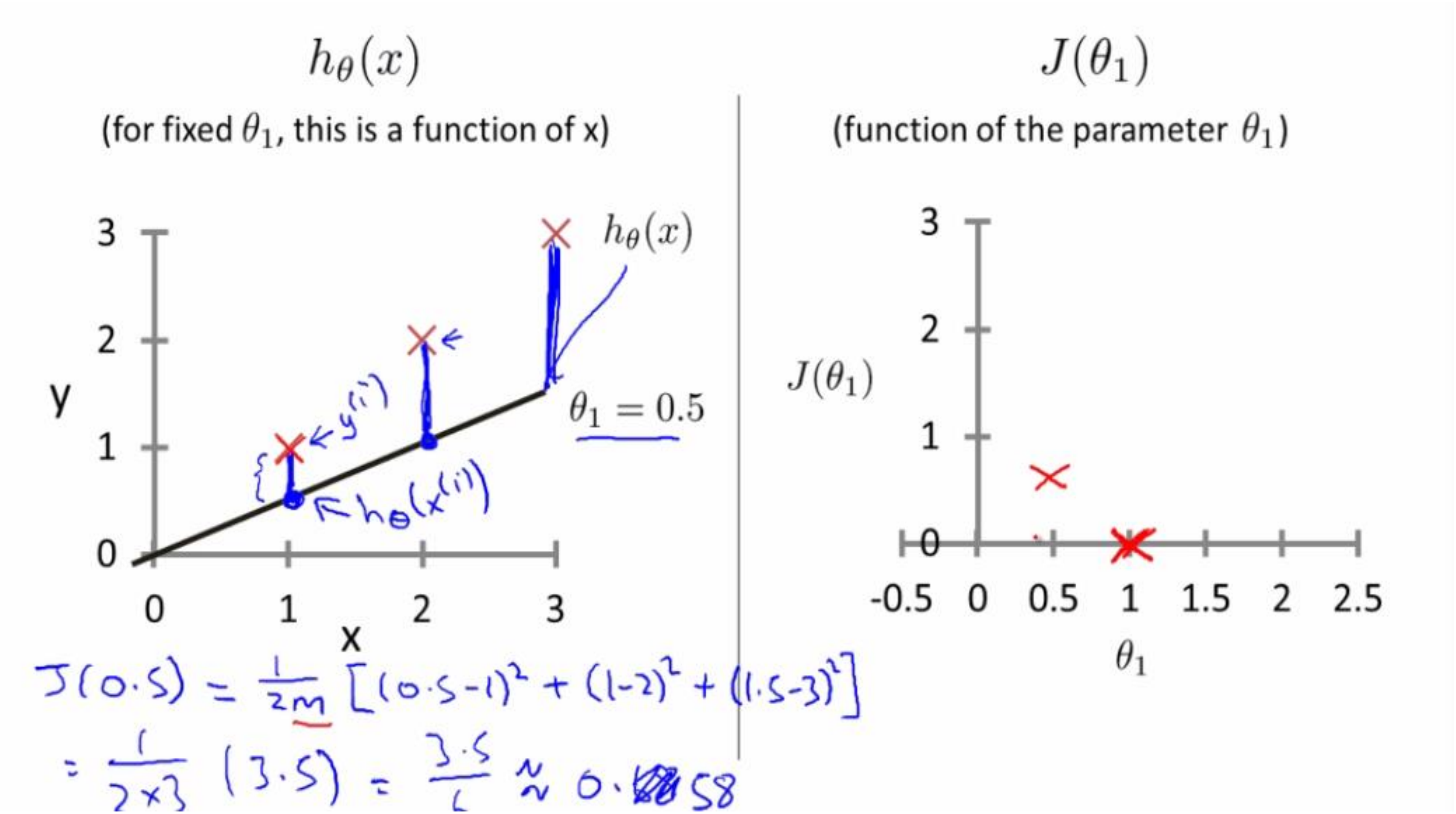
Linear Regression Basic – Cost Function

A cost function tells us “how good” our model is at making predictions for a given set of parameters. The cost function has its own curve and its own gradients. The slope of this curve tells us how to update our parameters to make the model more accurate.

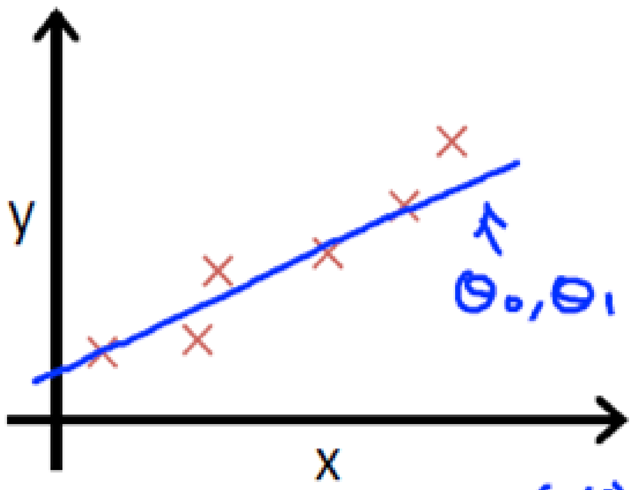
Cost Function:

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

Linear Regression Basic – Cost Function



Linear Regression Basic – Cost Function



θ_0, θ_1

$(x^{(i)}, y^{(i)})$

training examples

$$\underset{\theta_0, \theta_1}{\text{minimize}} \; \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

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

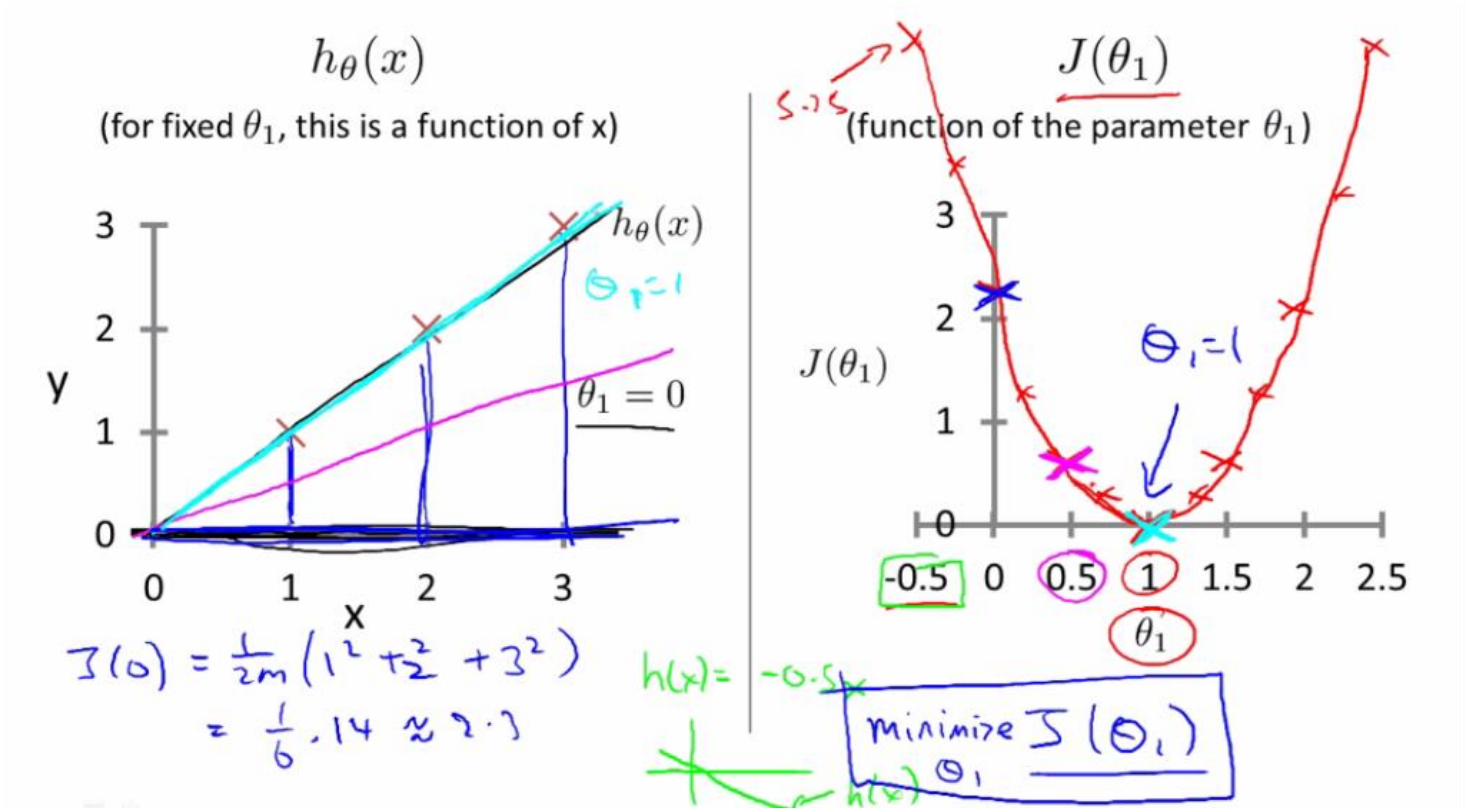
$$\underset{\theta_0, \theta_1}{\text{minimize}} \; \underbrace{J(\theta_0, \theta_1)}_{\text{Cost function}}$$

Squared error function

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

x, y

Linear Regression Basic – Cost Function



Linear Regression Basic – Cost Function

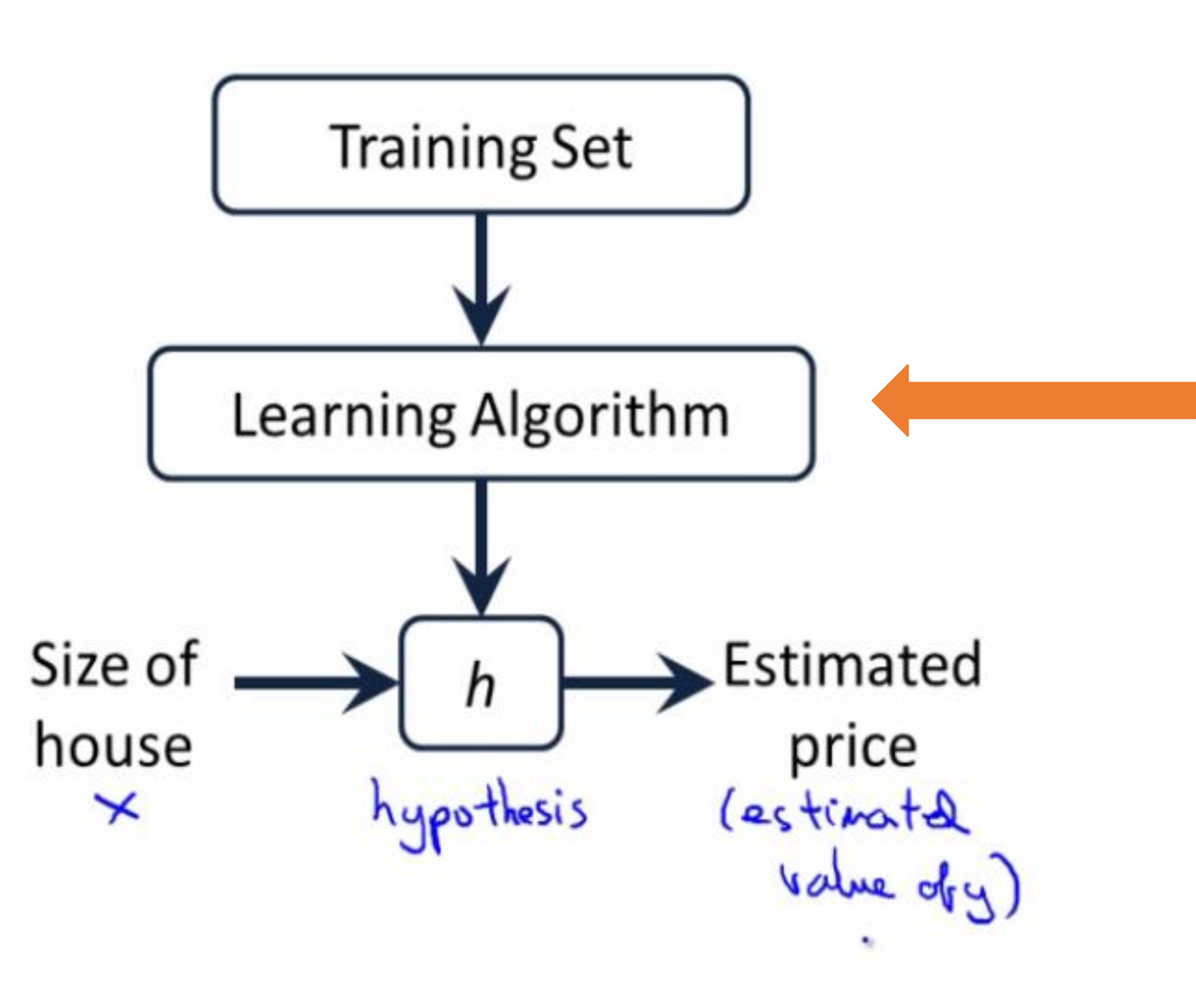
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Linear Regression Basic – Model Representation



PART 3

Linear Regression Basic – Optimization

"The core of ML is optimization"- **Siraj Raval**

Lovely Gradient Descent Article: [Link]()

Linear Regression Basic – Optimization

Gradient Descent is the most common optimization algorithm in *machine learning* and *deep learning*.

It is a first-order optimization algorithm. This means it only takes into account the first derivative when performing the updates on the parameters. On each iteration, we update the parameters in the opposite direction of the gradient of the objective function(aka Cost function) *J(w)* w.r.t the parameters where the gradient gives the direction of the steepest ascent.

The size of the step we take on each iteration to reach the local minimum is determined by the learning rate α. Therefore, we follow the direction of the slope downhill until we reach a local minimum.

Parameters:

$$\theta_0, \theta_1$$

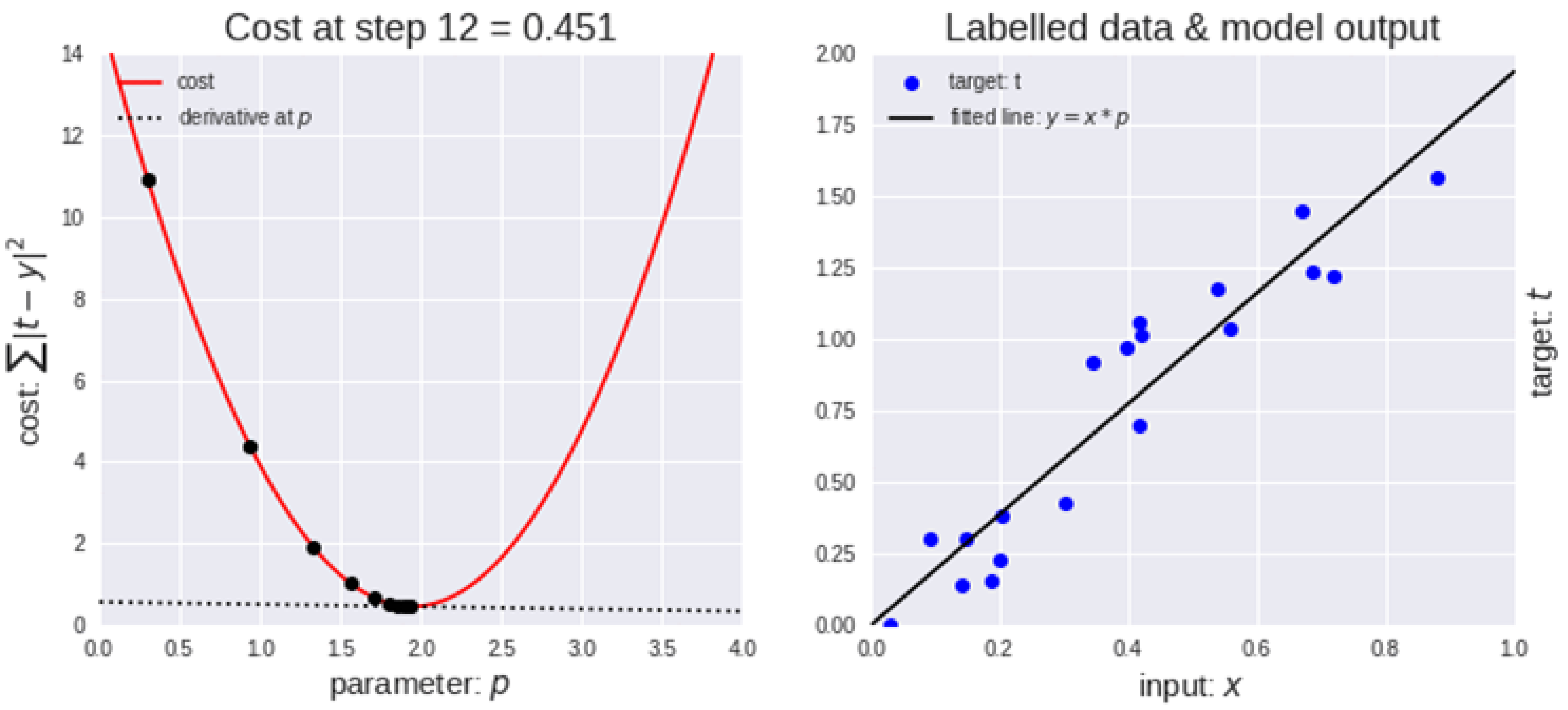
Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Cost Function:

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

Linear Regression Basic – Optimization

Gradient Descent: It is an iterative optimization algorithm used in machine learning to find the best result (minima of a curve).

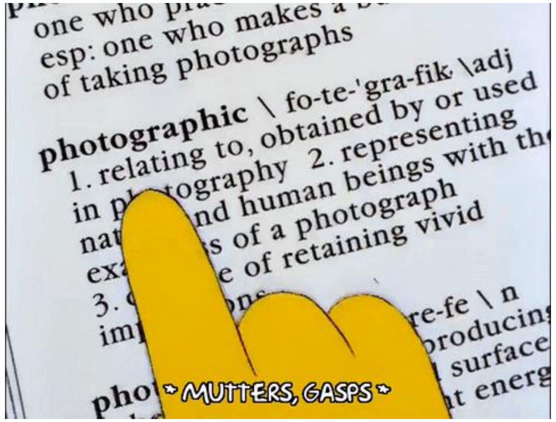


- Start with some m, b value(0,0)
- Keep changing m, b to reduce J(m,b) until we get to minimum

Hyperparameters:
α - learning rate

- Iterative means we need to get the results multiple times to get the most optimal result.

Linear Regression Basic – Optimization



Parameters:

θ_0, θ_1

θ_0 : Reading time

θ_1

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

:Reading Technique

Linear Regression Basic – Optimization

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$\theta_0: \quad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\theta_1: \quad \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

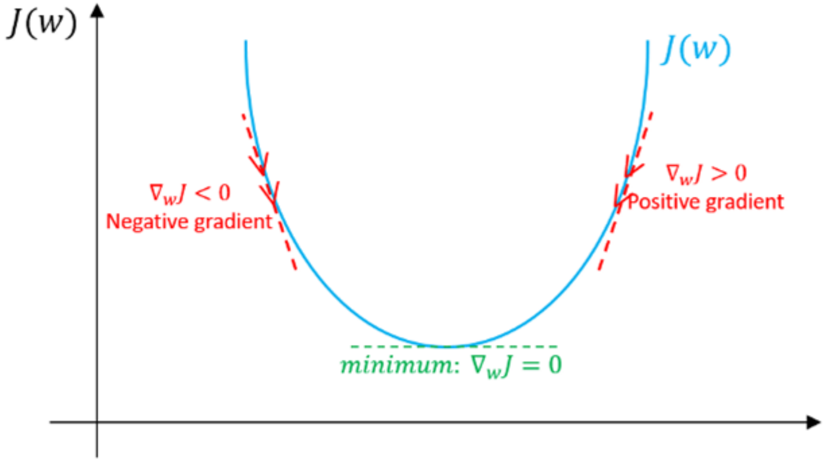
repeat until convergence {

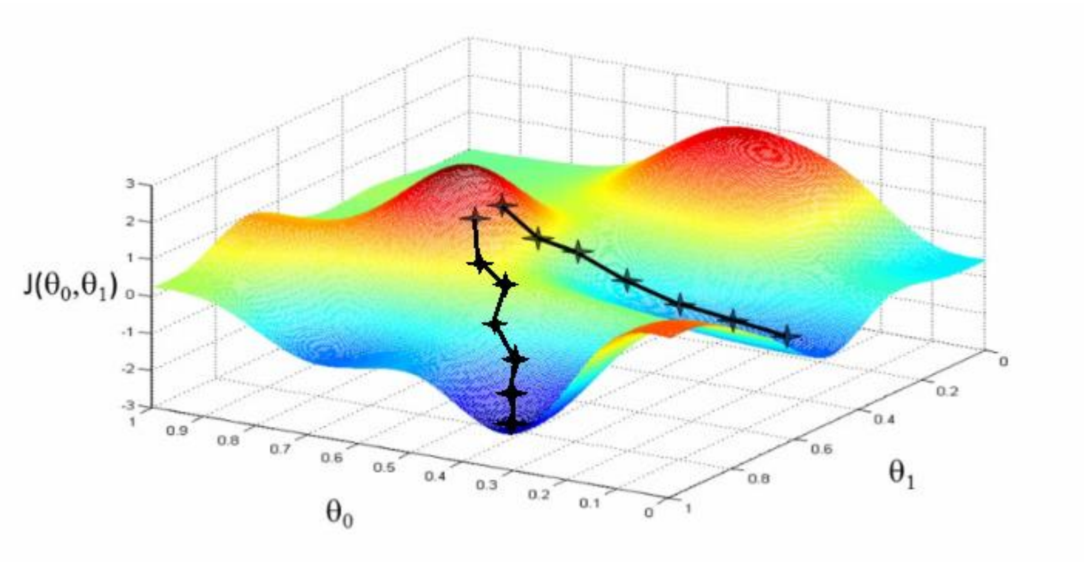
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

}

Linear Regression Basic – Optimization





- A convex function is a function that is local optima free.In our linear regression problem, there was only one minimum.

Note: a non convex function has multiple optima and is not appropriate for machine learning (although we can still use the algorithms, but they may get stuck in non-optimal solutions).

Linear Regression Basic – Optimization

Choosing a proper learning rate can be difficult...

- Learning rate is too small
  - => This is painfully slow convergence
- Learning rate is too large
  - => This can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.

α :Reading Technique

The learning rate is an hyperparameter

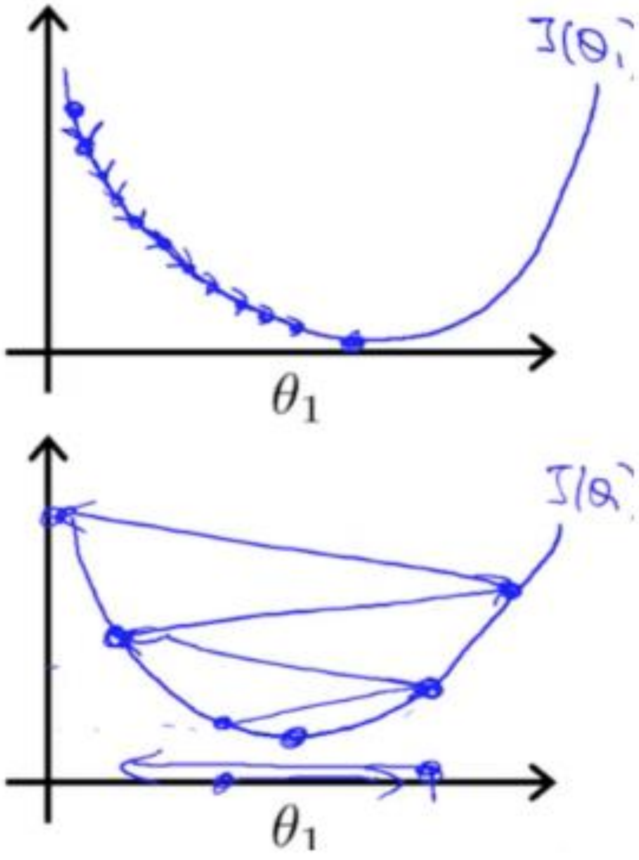
Linear Regression Basic – Optimization

Choosing a proper learning rate can be difficult...

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

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

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



- The most commonly used rates are : *0.001, 0.003, 0.01, 0.03, 0.1, 0.3*.

Linear Regression Basic – Optimization

Choosing a proper learning rate can be difficult...

A model parameter is a configuration variable that is internal to the model and whose value can be estimated from data.

- They are required by the model when making predictions.
- They are often not set manually by the practitioner.
- They values define the skill of the model on your problem.

A model hyperparameter is a configuration that is external to the model and whose value cannot be estimated from data.

- They are often used in processes to help estimate model parameters.
- They are often specified by the practitioner.
- They are often tuned for a given predictive modeling problem.

θ_0 : Reading time

θ_1 : Sleeping time

α :Reading Technique

Gradient Descent for Normal Guys

```python
def gradient_descent(x, y, m_current=6, b_current=5, epochs=1000, learning_rate=0.0001):
    N = float(len(y))
    list_cost = []
    for i in range(epochs):
        y_current = (m_current * x) + b_current
        cost = sum([data**2 for data in (y-y_current)]) / N
        list_cost.append(cost)
        m_gradient = -(2/N) * sum(x * (y - y_current))
        b_gradient = -(2/N) * sum(y - y_current)
        m_current = m_current - (learning_rate * m_gradient)
        b_current = b_current - (learning_rate * b_gradient)
    return m_current, b_current,list_cost
```

$$\frac{\partial}{\partial\theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial\theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$= \frac{\partial}{\partial\theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)^2$$

$$\theta_0 \quad j = 0 : \frac{\partial}{\partial\theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 \quad j = 1 : \frac{\partial}{\partial\theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial\theta_0} J(\theta_0, \theta_1)$$

$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial\theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp0}$$

$$\theta_1 := \text{temp1}$$

Gradient Descent for PhD… vectorization

```python
def computeCost(X, y, theta=[[0],[0]]):
    m = y.size
    J = 0
    h = X.dot(theta)
    J = 1/(2*m)*np.sum(np.square(h-y))
    return(J)
```

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

```python
def gradientDescent(X, y, theta=[[0],[0]], alpha=0.0001, num_iters=1500):
    m = y.size
    J_history = np.zeros(num_iters)
    for iter in np.arange(num_iters):
        h = X.dot(theta)
        theta = theta - alpha*(1/m)*np.dot(X.T,h-y)
        J_history[iter] = computeCost(X, y, theta)
    return(theta, J_history)
```

The gradient can be calculated as:

$$f'(m,b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N} \sum -2(y_i - (mx_i + b)) \end{bmatrix}$$

Hilton's Closing Prayer

Our father who art in n-dimensions

hallowed by the backprop,

thy loss be minimized,

thy gradients unvarnished,

on earth as it is in Euclidean space.

Give us this day our daily hyperparameters,

and forgive us our large learning rates,

as we forgive those whose parameters diverge,

and lead us not into discrete optimization,

but deliver us from local optima.

For thine are dimensions,

and the GPUs, and the glory,

forever and ever. Dropout.

AI6