Google Developer Student Clubs

Head Start Machine Learning

Session 1: Basic Concepts of Machine Learning

Hong Jia Herng
Year 2 CS(IS)

Lew Jun Bin
Year 2 CS(SE)

Attendance

> Select Tab “Head Start Machine Learning”
> Mark your attendance by inserting “1”
> Do it within this 2 hours

https://docs.google.com/spreadsheets/d/1SP56Rr-9BZIEjb2Qz3O_VWml1pTRVB_alY4mVKgF98A/edit?usp=sharing

What you will learn….

- Types of ML
- Linear Regression & gradient descent
- Underfitting, overfitting, data splitting

Drop your question in the chat box if you don’t understand anything!

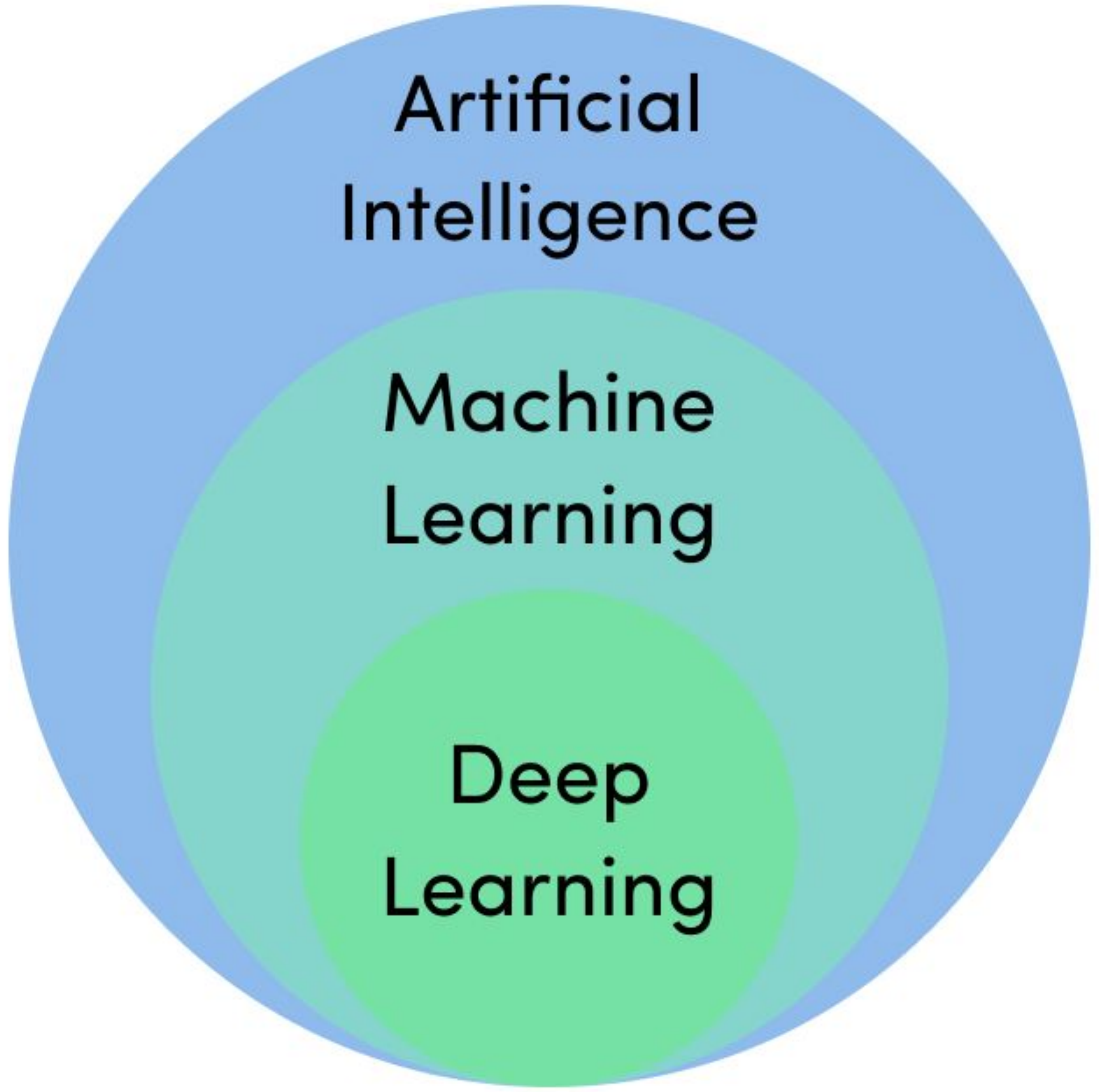
Introduction

What is Machine Learning?

- **Arthur Samuel (1959)**: *"Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed."*

- **Tom Mitchell (1997)**: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."*

Then what about Artificial Intelligence?



AI is an idea; ML is one of the methods to “achieve” AI; DL is a method in ML with the use of ANNs

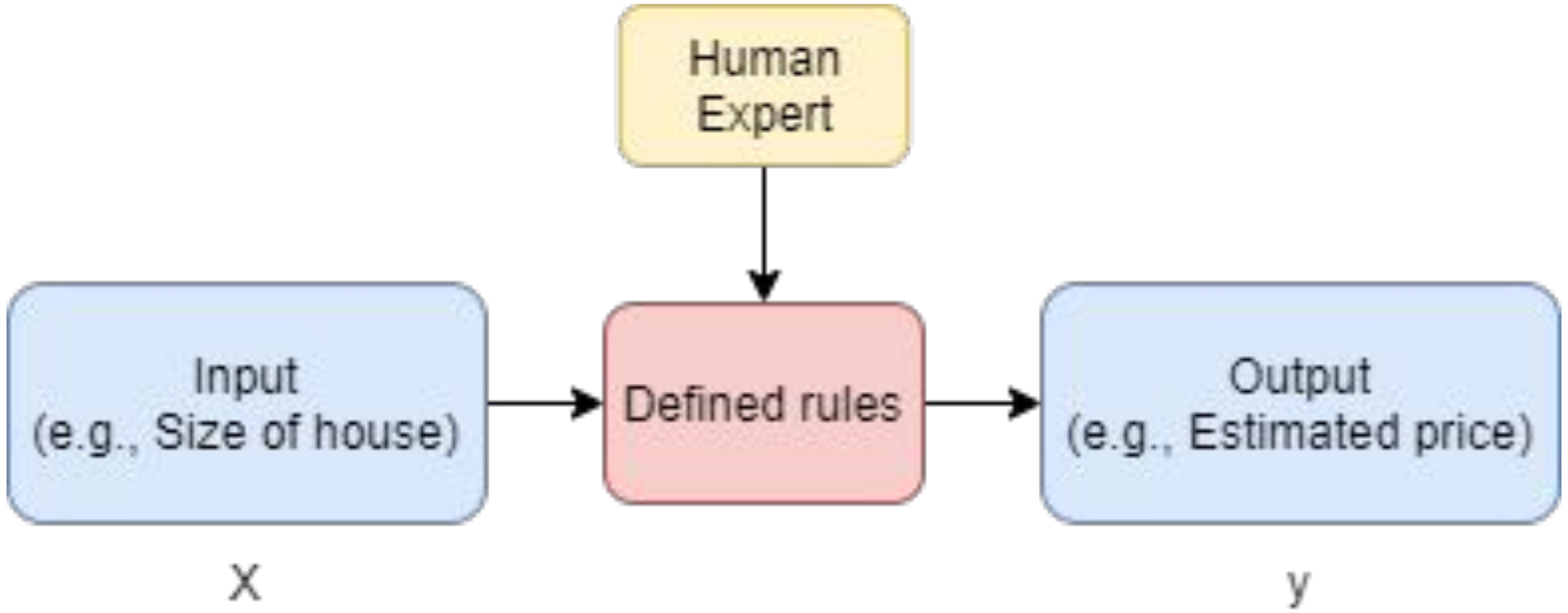
Why people opt for ML instead of symbolic AI in some cases?

Can we program an agent to play Go by using symbolic AI methods?

- Yes we can, but the program takes forever to run on current computer
- #legal positions = *~2.081681994 * 10^170*
- Searching (e.g., minimax algorithm & its variants) is still computationally feasible with Tic Tac Toe but impossible on Go
- Thus, Reinforcement Learning (a type of ML technique) is used to build AlphaGo and it achieved success
- https://www.youtube.com/watch?v=WXuK6gekU1Y

Reference: https://tromp.github.io/go/legal.html

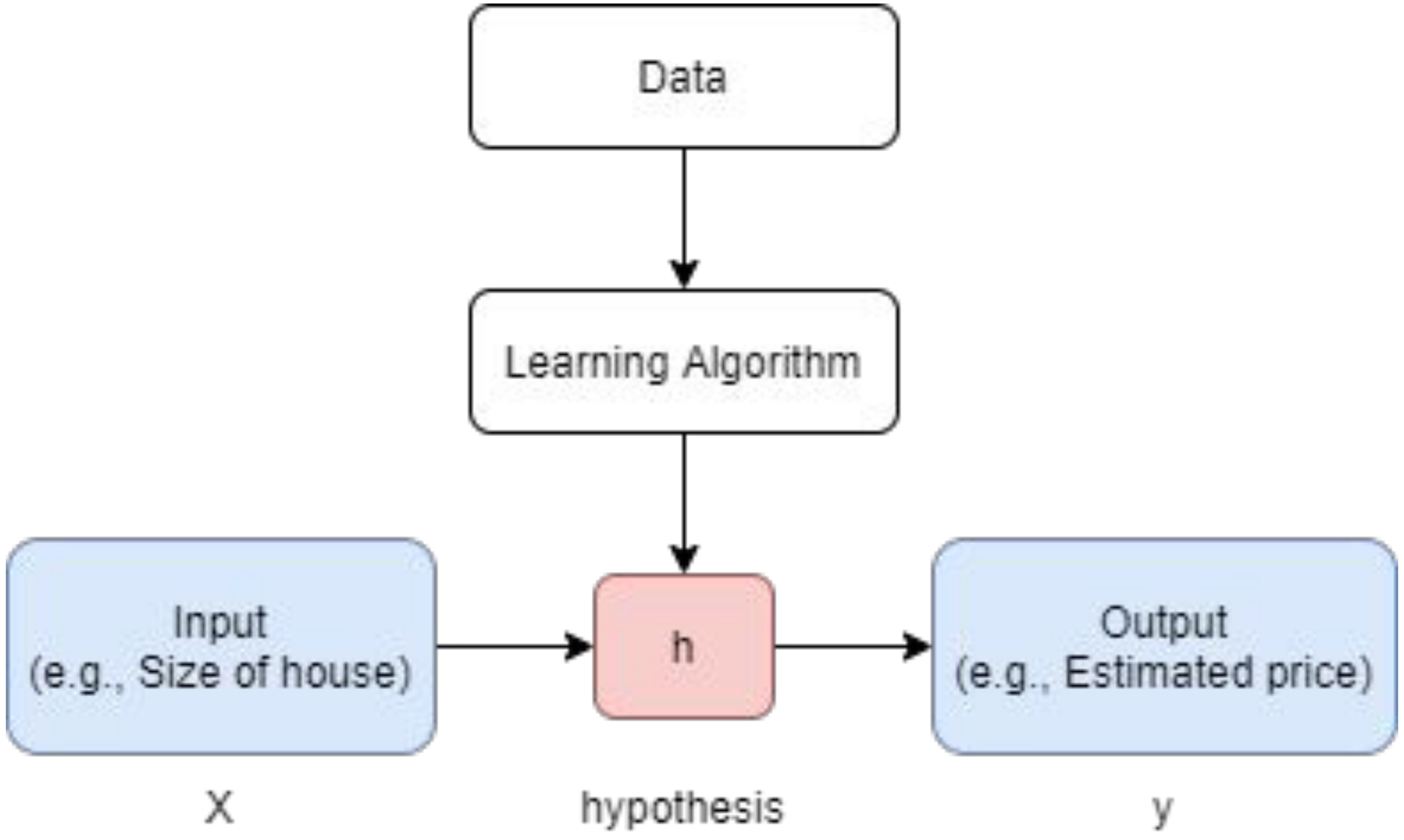
Traditional programming vs Machine Learning approach



Traditional programming approach

- Hire a human expert of the field (i.e., Real estate leasing agent) manually identify suitable rules to determine a price of a house and program it
- Over time, when the market changes, the previously defined rules can't adapt to the new trend
- Redefining the rule is needed

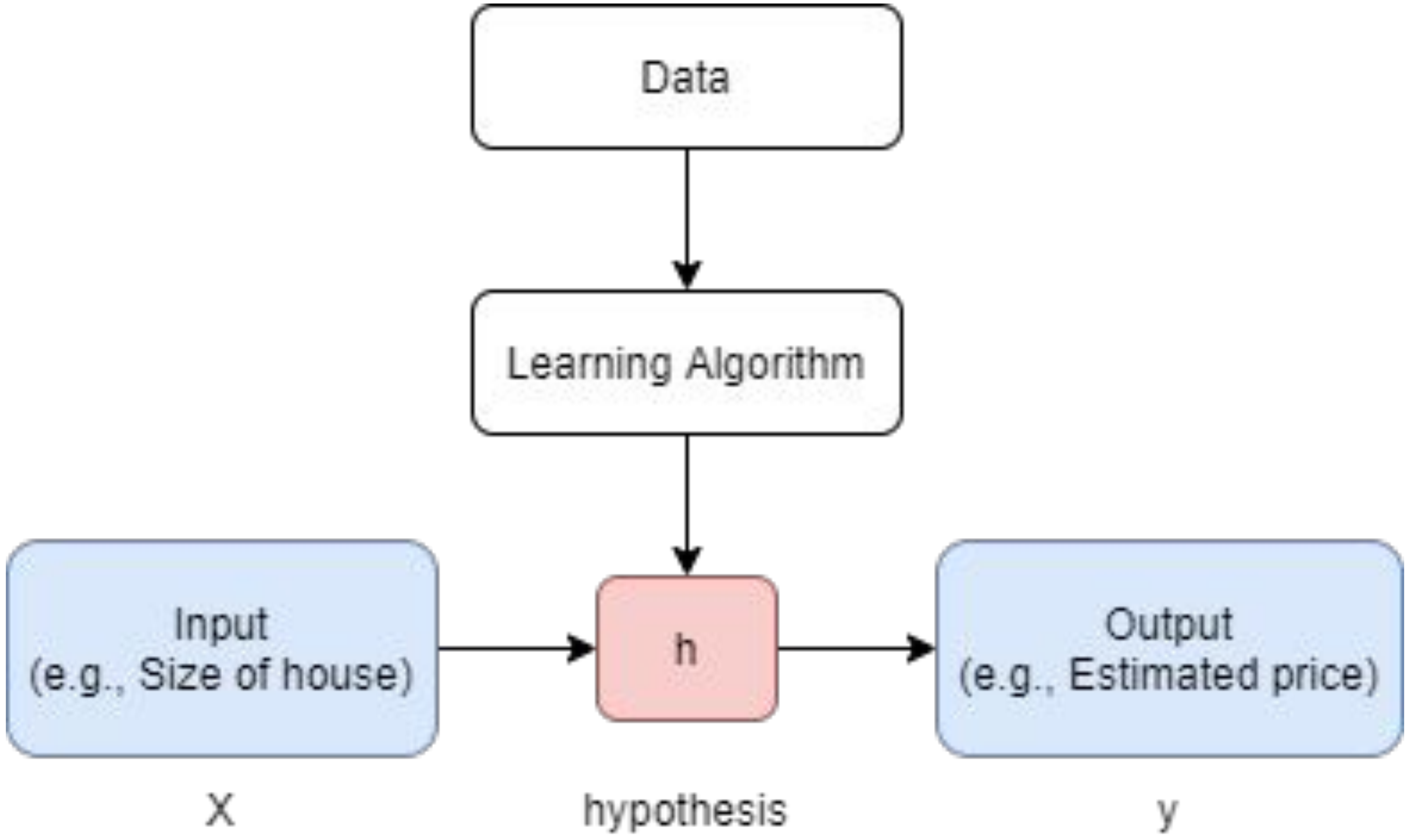
Traditional programming vs Machine Learning approach



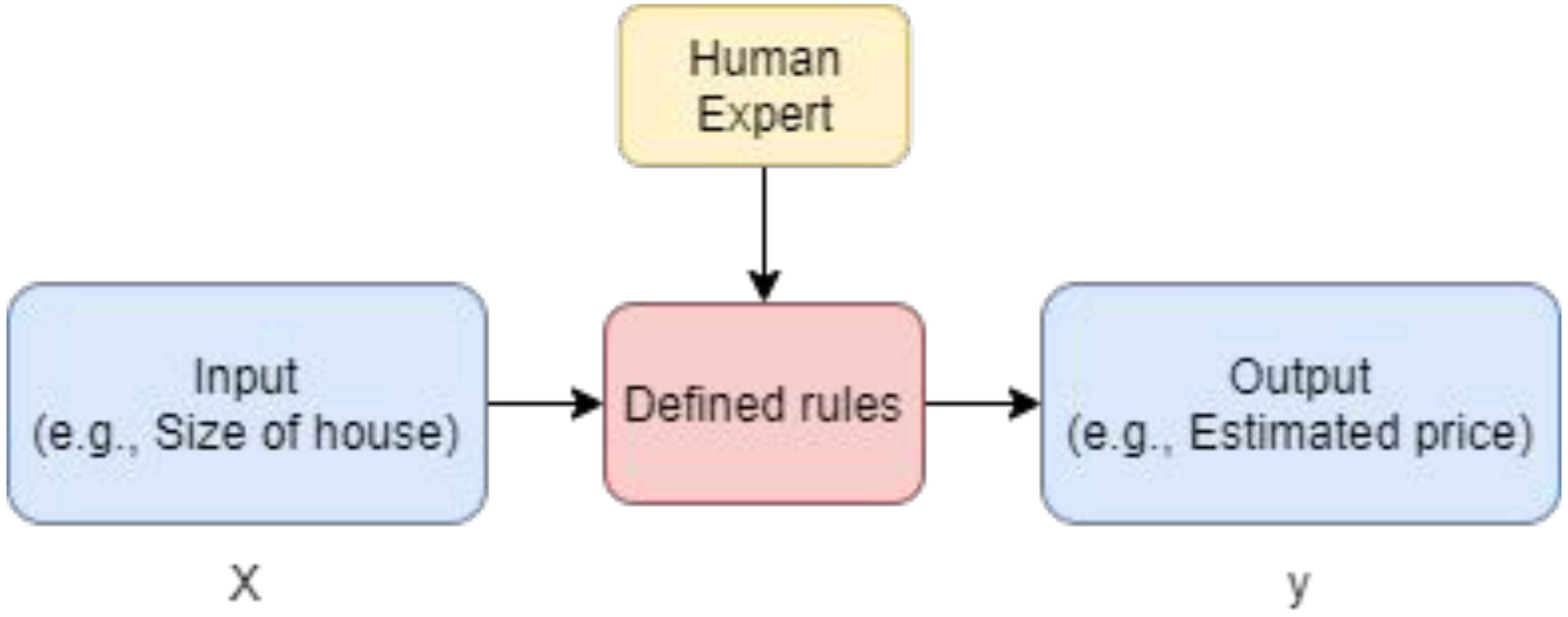
Machine learning approach

- Learn a function mapping ***h*** that maps X to y
- Can be written as either of below
    - $h(X) = y$
    - $X \xrightarrow{h} y$
- 'h' is called 'model hypothesis' or simply 'model'
- Data contains input-output pairs are fed to the ML algorithm to find pattern & learn function h

Side-to-side comparison

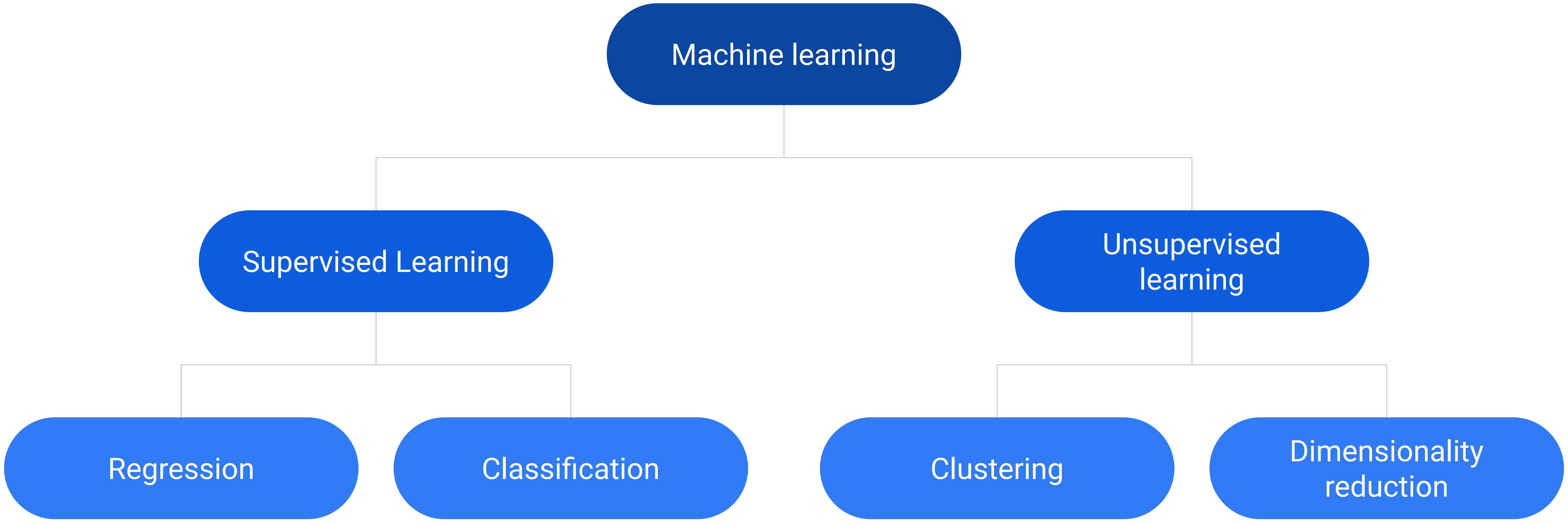


Machine learning approach



Traditional programming approach

Types of Machine Learning



Supervised learning vs unsupervised learning

Supervised Learning	Unsupervised learning
Given labeled data (x, y) pairs and find function mapping that maps X to Y	Given unlabeled data and discover pattern from it

Supervised learning - Regression

- Predict a **continuous** output
- Example: Predict the age based on their selfies

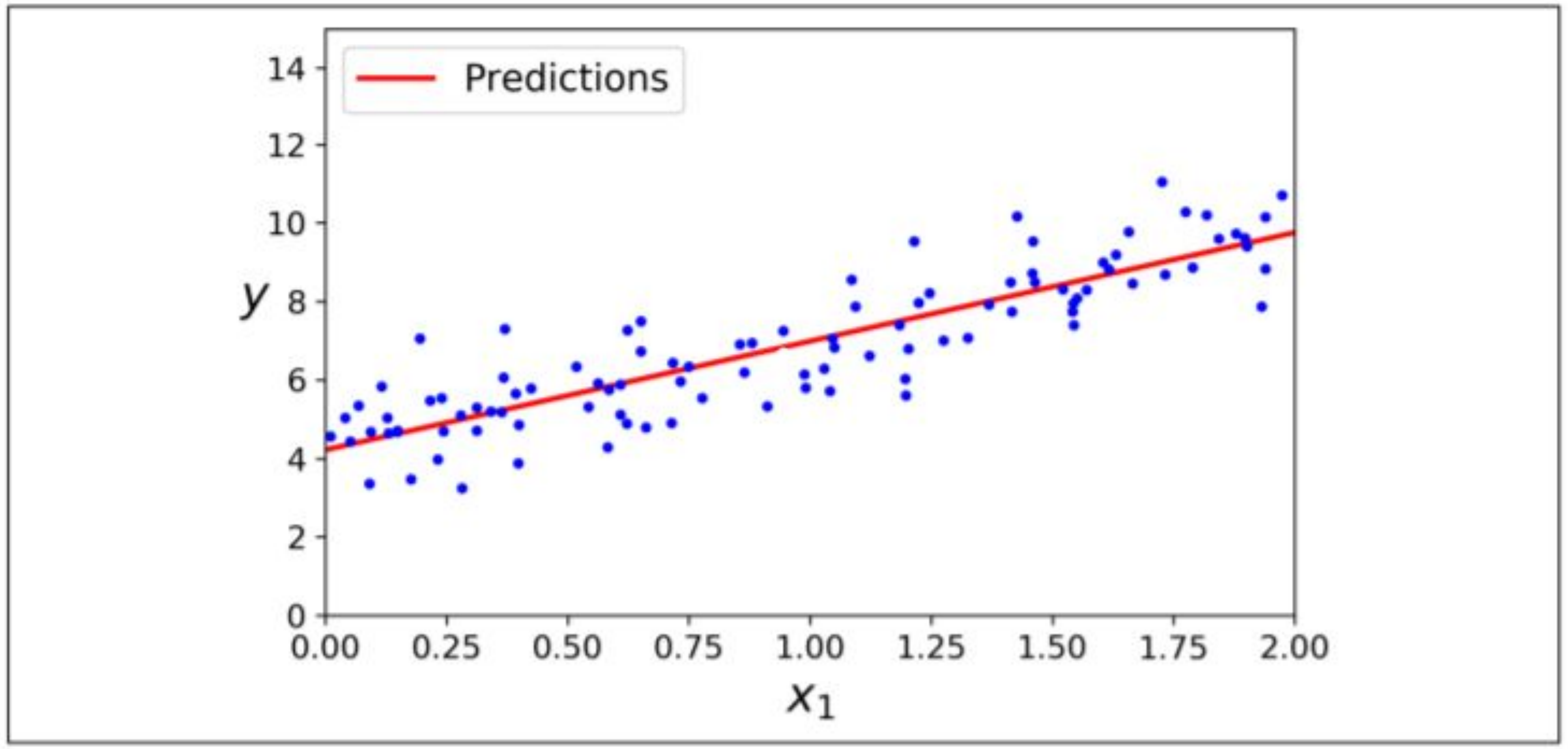


Figure 4-2. Linear Regression model predictions

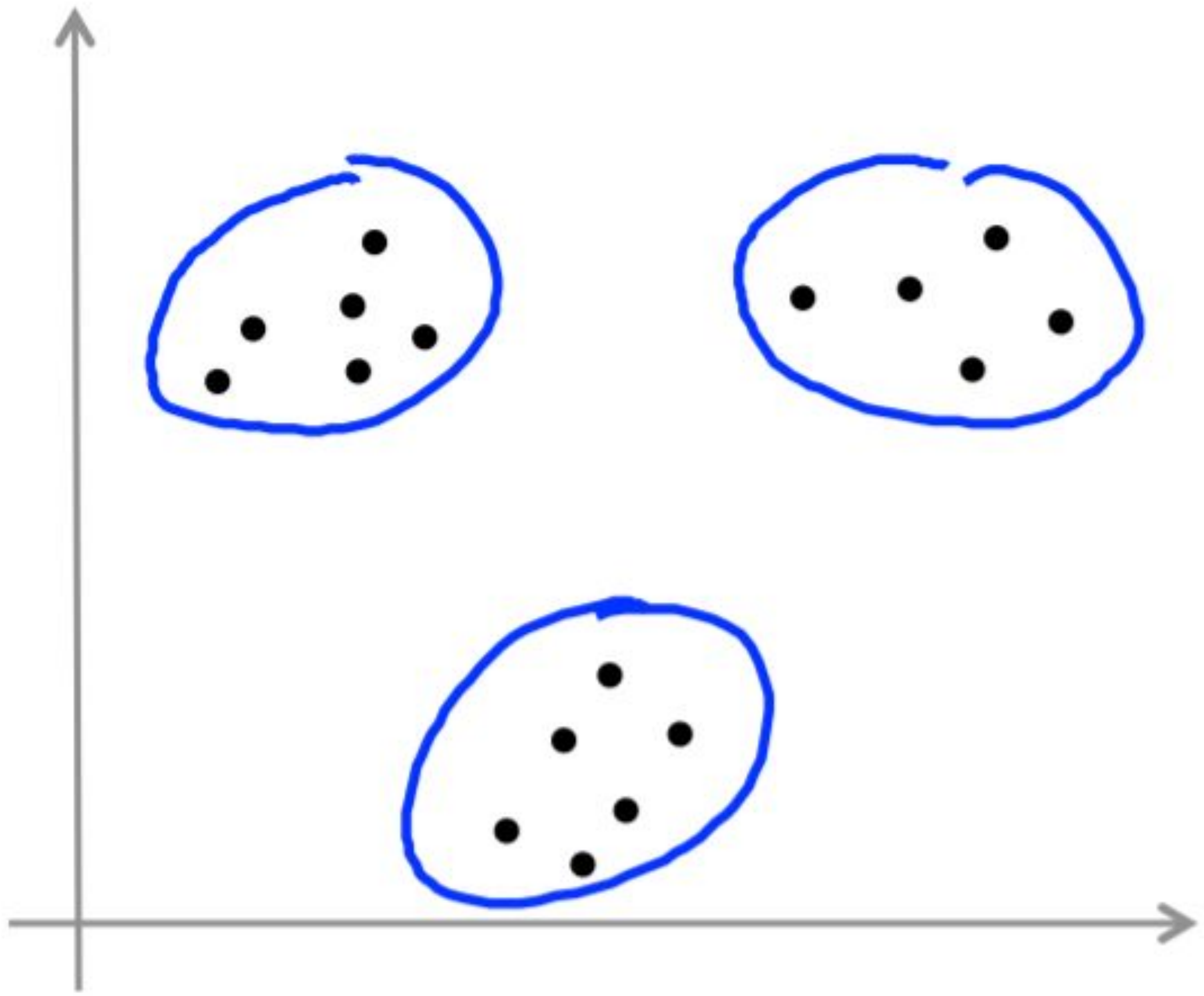
Supervised learning - Classification

- Predict a **discrete** output
- Predict their marital status (married or not) based on their selfies

What's that!? Cred
@atchoumfan | More 👉
@miinute

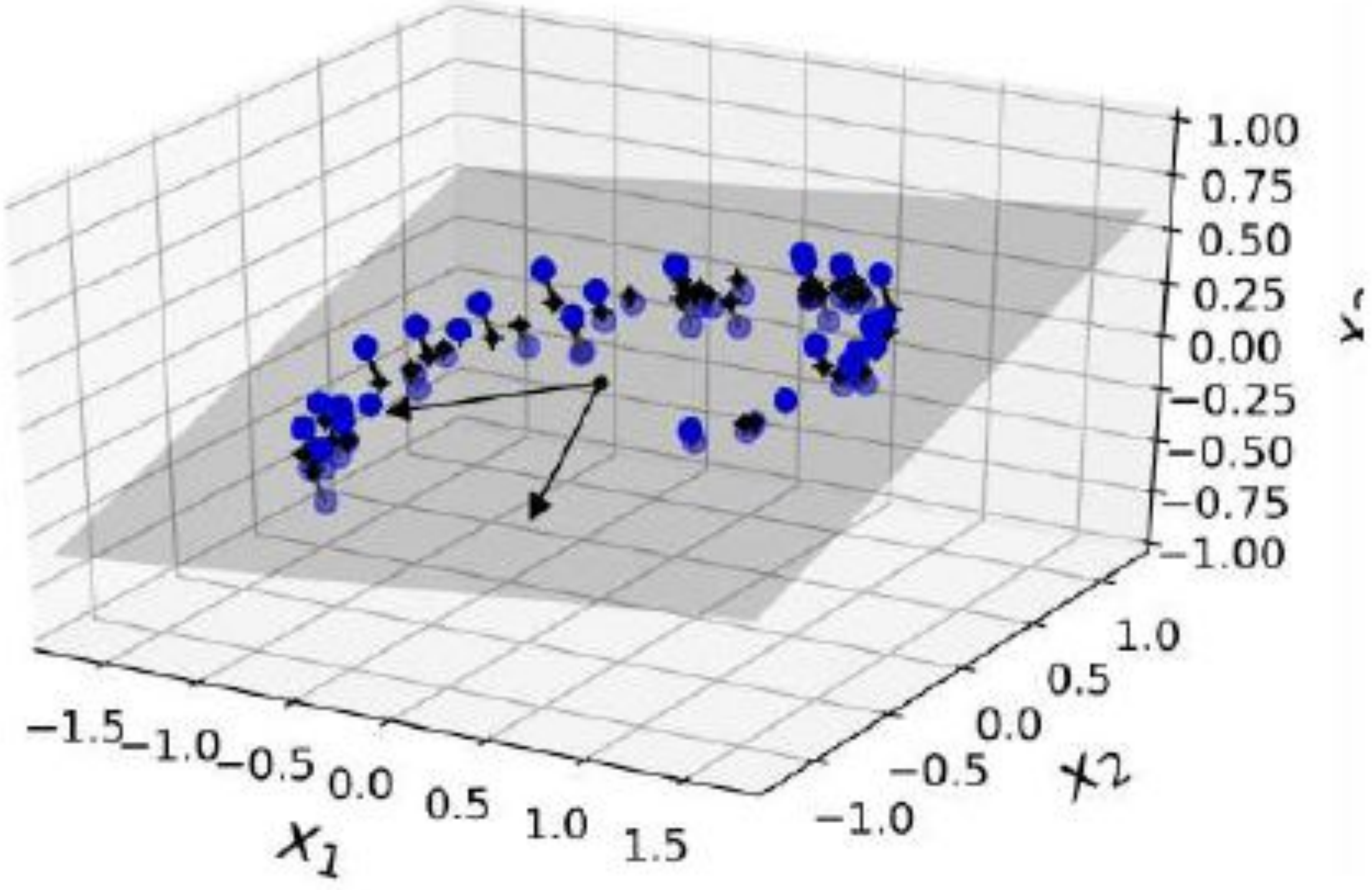
Unsupervised learning - Clustering

- Grouping data without labelled data

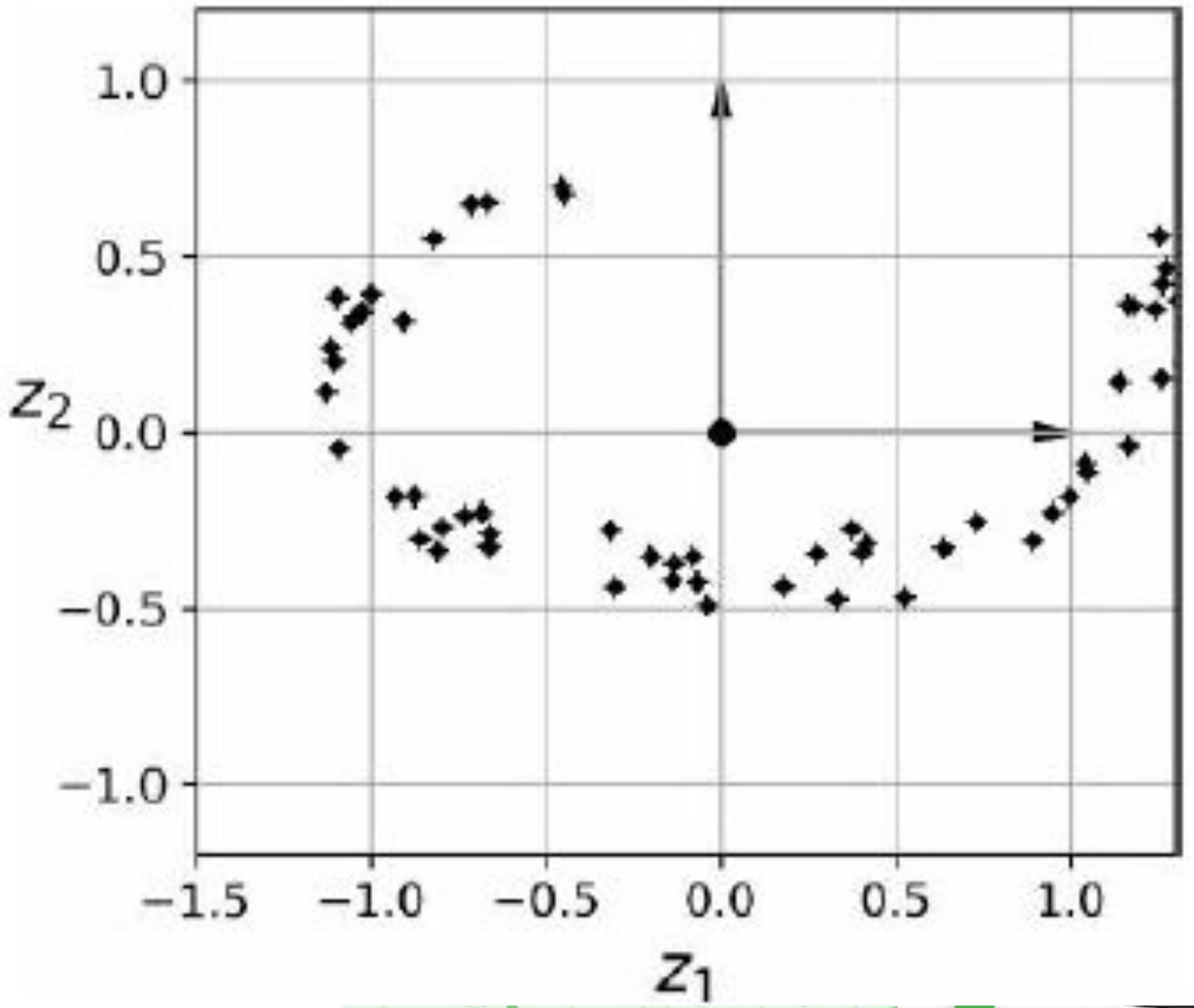


Unsupervised learning - Dimensionality reduction

- Singular Vector Decomposition (SVD)
- Principle Component Analysis (PCA)



PCA



Your first ML algorithm
Linear regression

Before diving in ...

Preface

- ML is not easy!
- We can't make you an expert
- However, we can teach you the fundamentals to better understand the literature of ML
- There's some heavy maths later. However, it's unnecessary but useful to know
- We'll have hands on session later to solidify the concepts
- TIPS: Checking the shape of matrix is super helpful for understanding!

Univariate Linear Regression

Motivating example

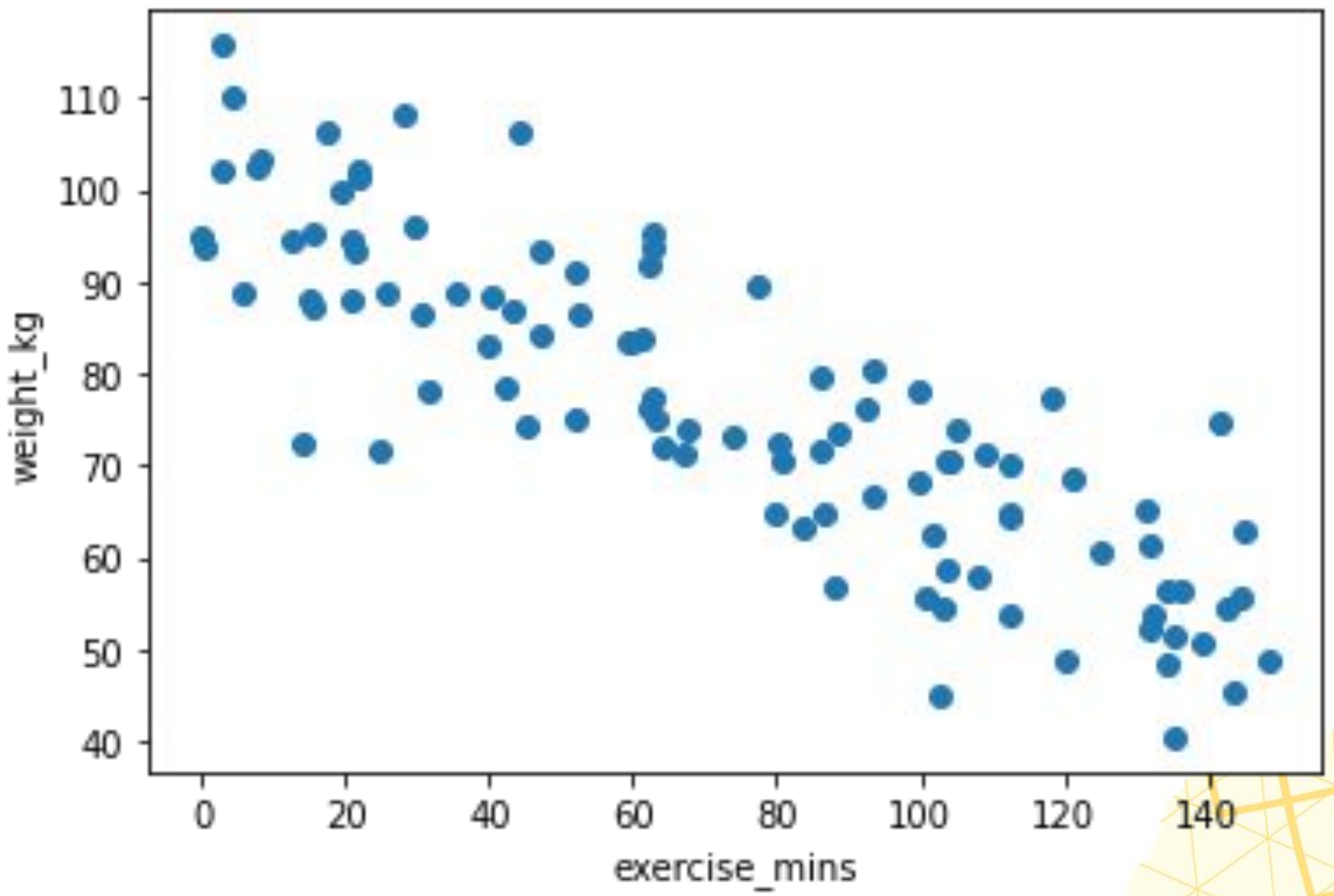
Input feature → exercise_mins

Target output → weight_kg

exercise_mins	weight_kg
62.553301	95.392353
108.048674	57.866211
0.017156	94.712564
45.349886	74.153685
22.013384	101.316282
...	...
35.554047	88.922052
135.506928	51.392487
86.051923	71.751994
0.430549	93.656475
92.571737	76.123075

Our problem:

- We want to build an algorithm that can take in total minutes of exercise per week as input and predict the person's weight



Model representation

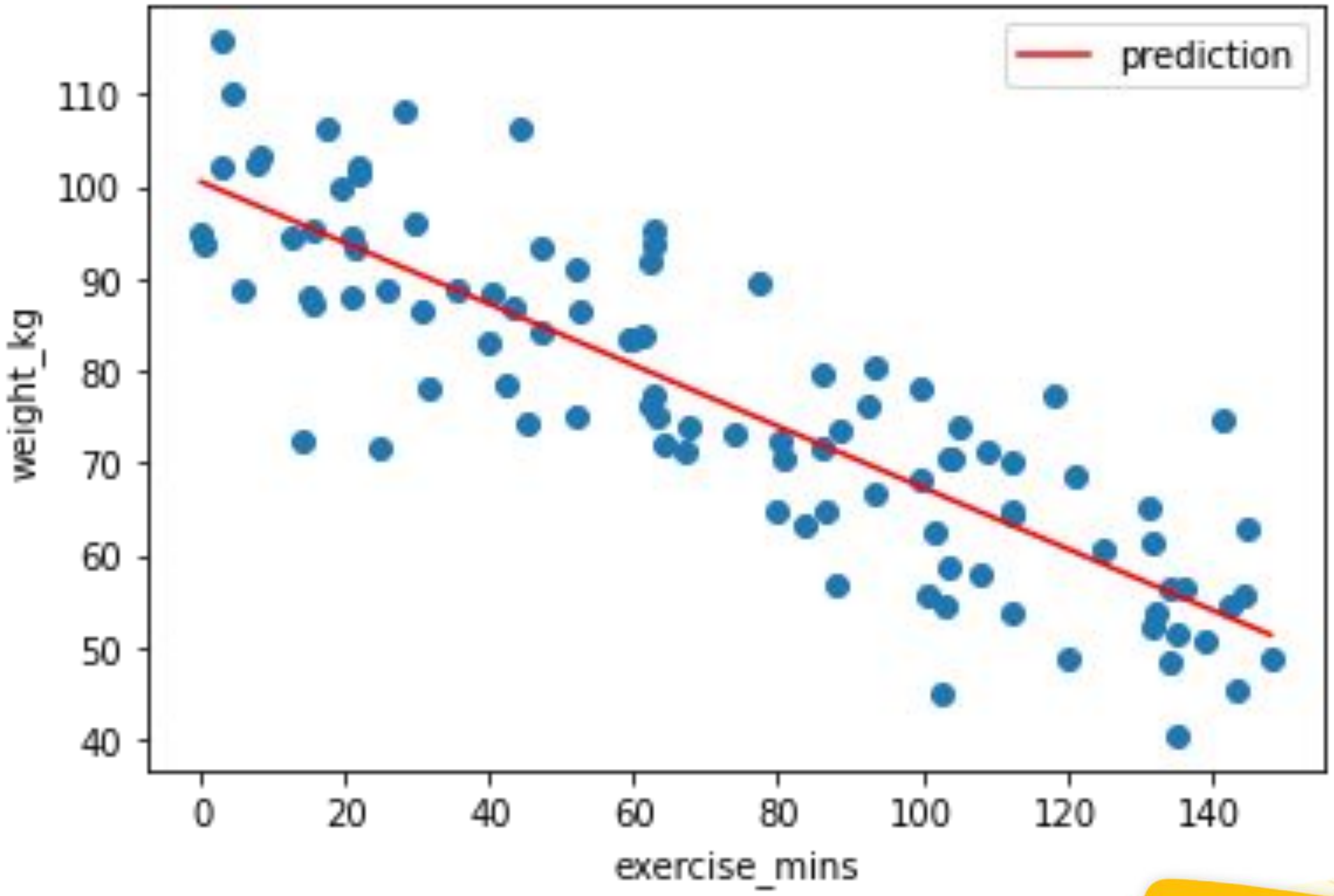
Equation of straight line

$$y = mx + c$$

Model hypothesis

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

We need to choose θ_0 and θ_1 for this dataset



Model parameters

Model hypothesis

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

θ_0 and θ_1 that we need to choose is called **model parameters**

Specifically,

θ_0 : Bias / intercept term
θ_1 : Weight

Notations

Define:

- m : number of training examples / instances
- n : number of features
- X : input “features” (capital X to denote matrix)
- y : “target” value (small letter y to denote vector)
- θ : parameters
- h(x) : model hypothesis / predicted target value

Let's put everything into matrices/vectors

Input feature, X

Target output, y

exercise_mins	weight_kg
62.553301	95.392353
108.048674	57.866211
0.017156	94.712564
45.349886	74.153685
22.013384	101.316282
...	...
35.554047	88.922052
135.506928	51.392487
86.051923	71.751994
0.430549	93.656475
92.571737	76.123075

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

In our case,

m = 100
n = 1

$$X = \begin{bmatrix} 62.553301 \\ 108.048674 \\ 0.017156 \\ \vdots \\ 92.571737 \end{bmatrix}$$

(m x n) = (100 x 1)

$$y = \begin{bmatrix} 95.392353 \\ 57.866211 \\ 94.712564 \\ \vdots \\ 76.123075 \end{bmatrix}$$

(m x 1) = (100 x 1)

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

(n+1 x 1) = (2 x 1)

$$h(X) = \begin{bmatrix} h(x^{(1)}) \\ h(x^{(2)}) \\ h(x^{(3)}) \\ \vdots \\ h(x^{(m)}) \end{bmatrix}$$

(m x 1) = (100 x 1)

Vectorized version of model hypothesis

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

If we implement the equation above in code,

- we'll need to run through a for loop 100 times to compute h(x) for all instances
- it's slow

```
# initialize m x 1 vector
y_pred = np.zeros((m, 1))

# Loop to compute y_pred
for i in range(m):
    y_pred[i] = theta0 + theta1 * X[i]
```

Vectorized version of model hypothesis

- We can make use of matrix multiplication library to speed up the computation
- Compute h(x) for m=100 instances in one go

$$h(X_{prep}) = X_{prep} \cdot \theta$$

(m x 1) = (m x n+1) x (n+1 x 1)

```
# Concatenate a column of 1's to left of X
X_prep = np.c_[np.ones((m, 1)), X]

# Compute model hypothesis
y_pred = np.dot(X_prep, theta)
```

Vectorized version of model hypothesis (Optional proof)

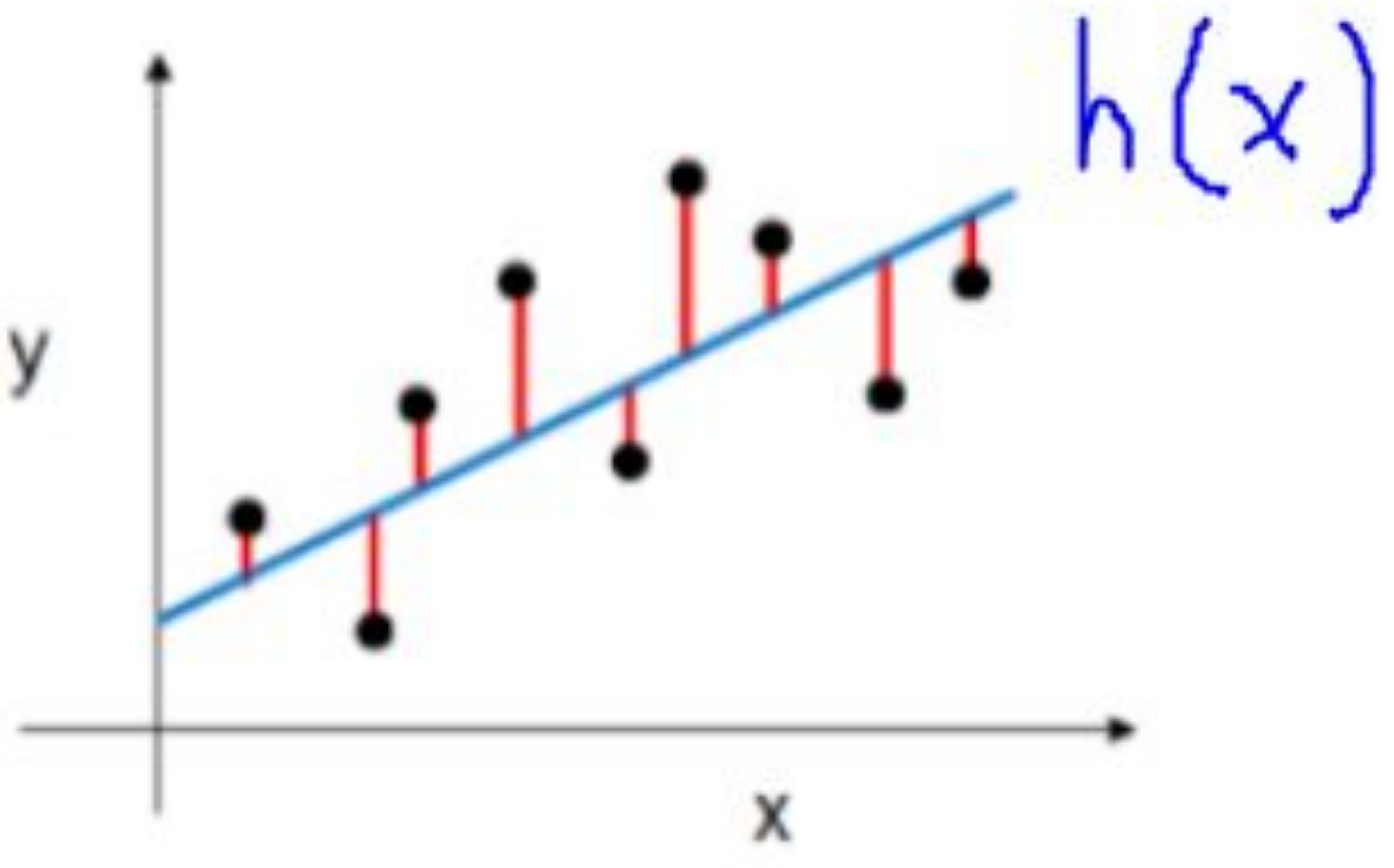
Step 1: Concatenate a column of 1's to X

$$X = \begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(100)} \end{bmatrix} \longrightarrow X_{prep} = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix}$$

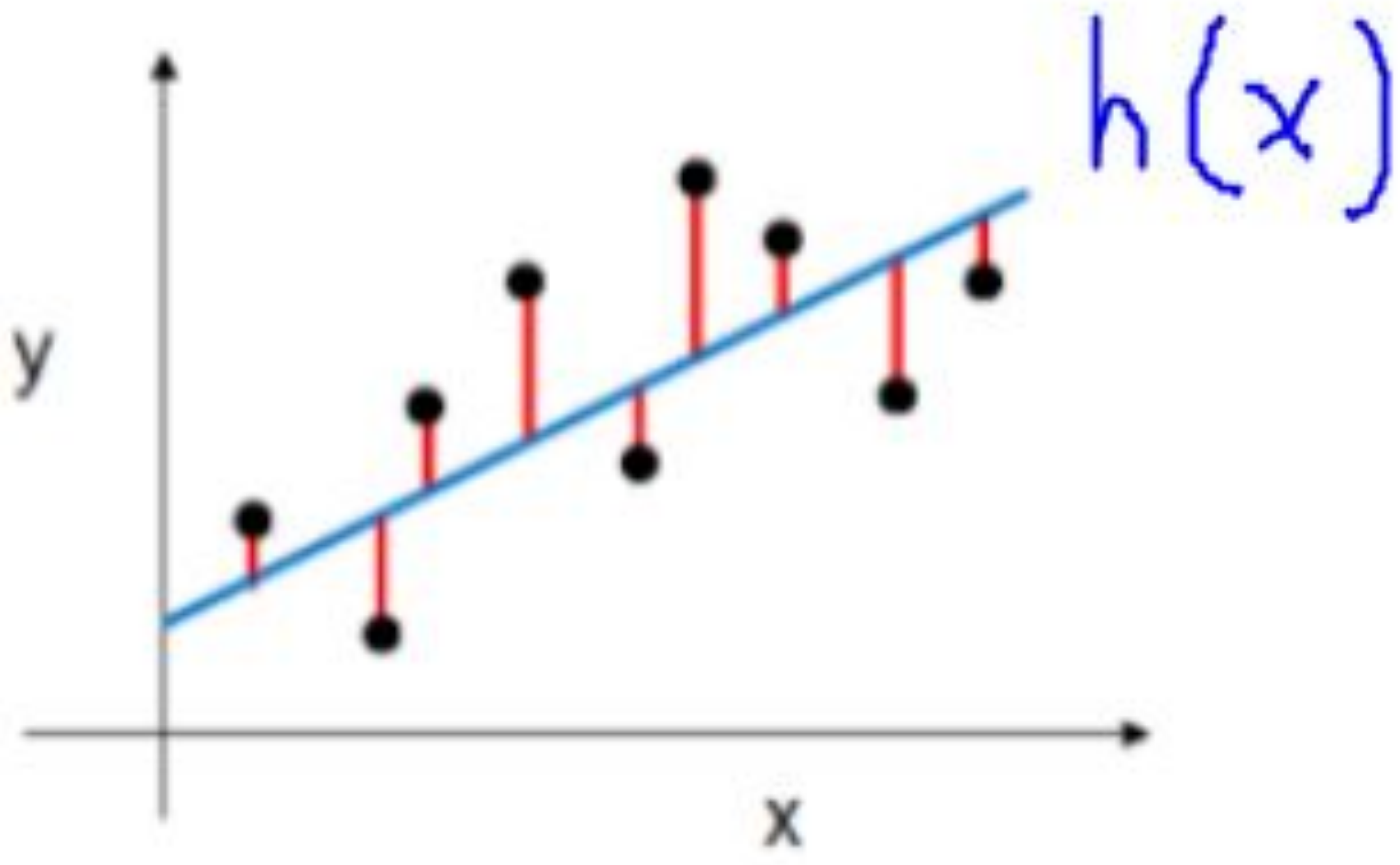
Step 2: Compute vectorized version of h(X)

$$\begin{aligned} h(X_{prep}) &= X_{prep} \cdot \theta \\ &= \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \\ &= \begin{bmatrix} \theta_0 \cdot 1 + \theta_1 x_1^{(1)} \\ \theta_0 \cdot 1 + \theta_1 x_1^{(2)} \\ \vdots \\ \theta_0 \cdot 1 + \theta_1 x_1^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h(x^{(1)}) \\ h(x^{(2)}) \\ \vdots \\ h(x^{(m)}) \end{bmatrix} \end{aligned}$$

How do we tell how well our predictions are?



Cost function of linear regression



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

This is called Mean Squared Error (MSE)

- diff between predict & actual y ↑, cost ↑
- diff between predict & actual y ↓, cost ↓
- square to eliminate -ve value

Our goal is to ...

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

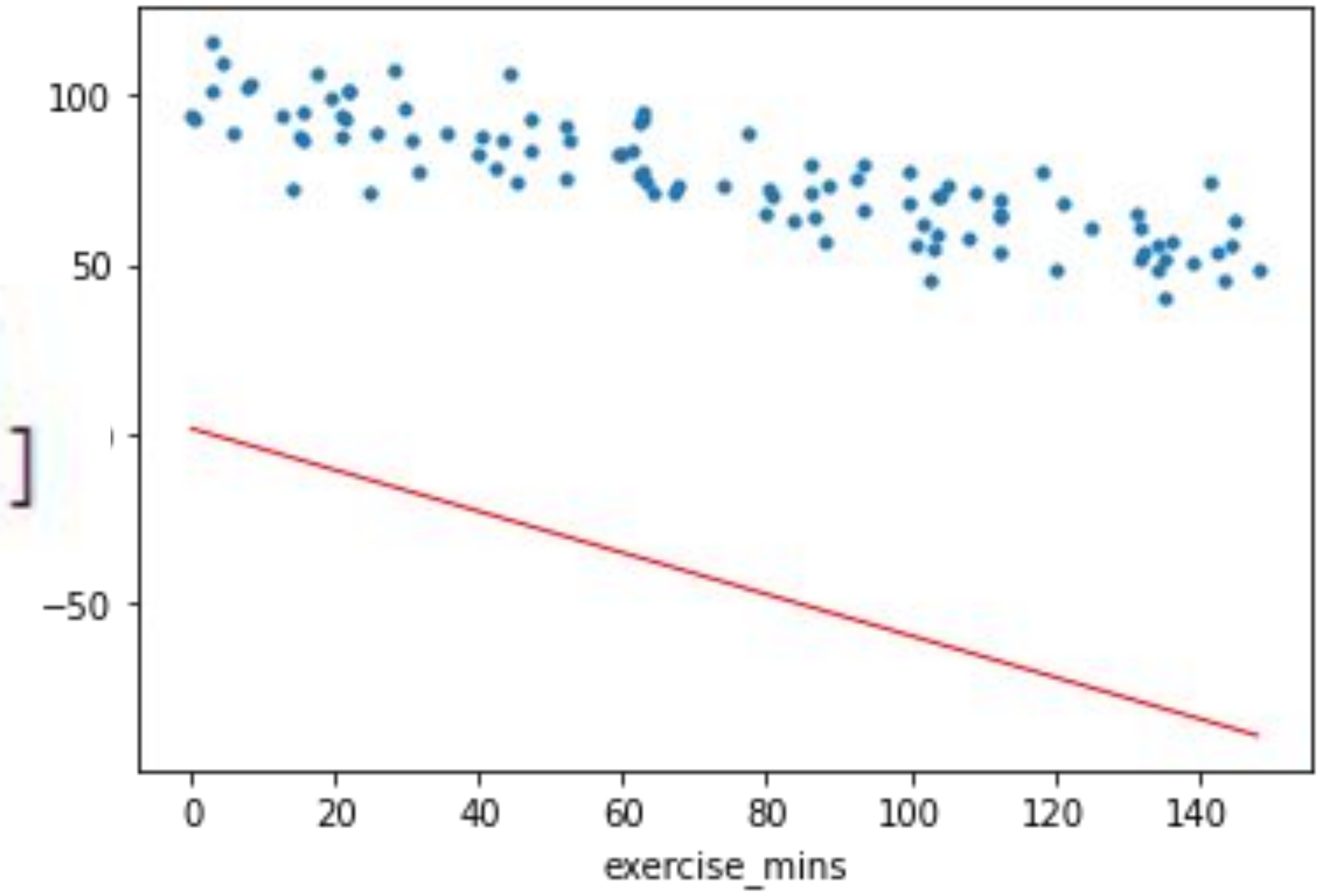
minimize the cost by choosing appropriate θ_0 and θ_1

Formally, it's written as :

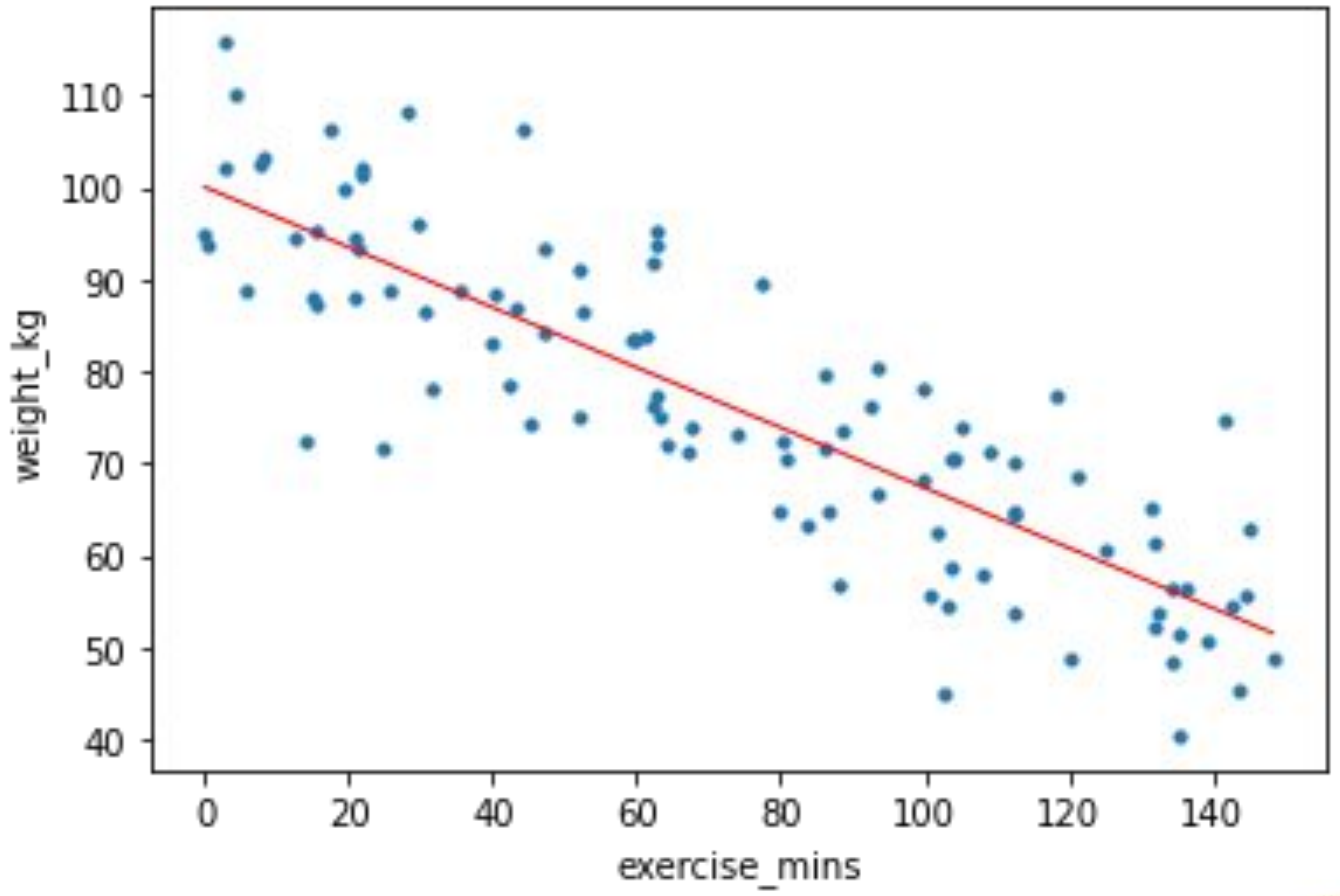
$$\min_{\theta} J(\theta)$$

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

```
[[ 1.62434536]
 [-0.61175641]]
```



cost = 72289625.26

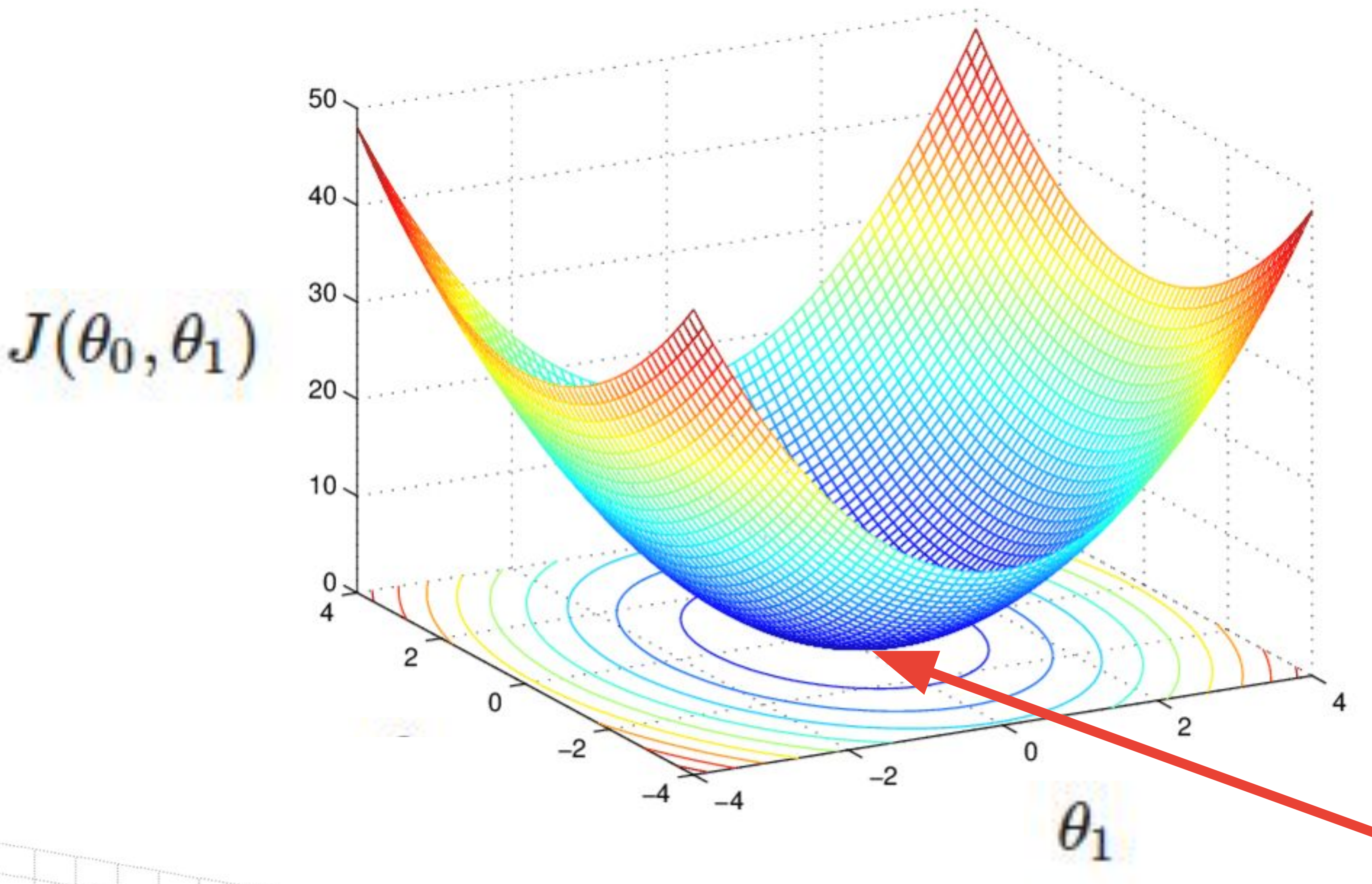



```
[[99.99500899]
 [-0.326646  ]]
```

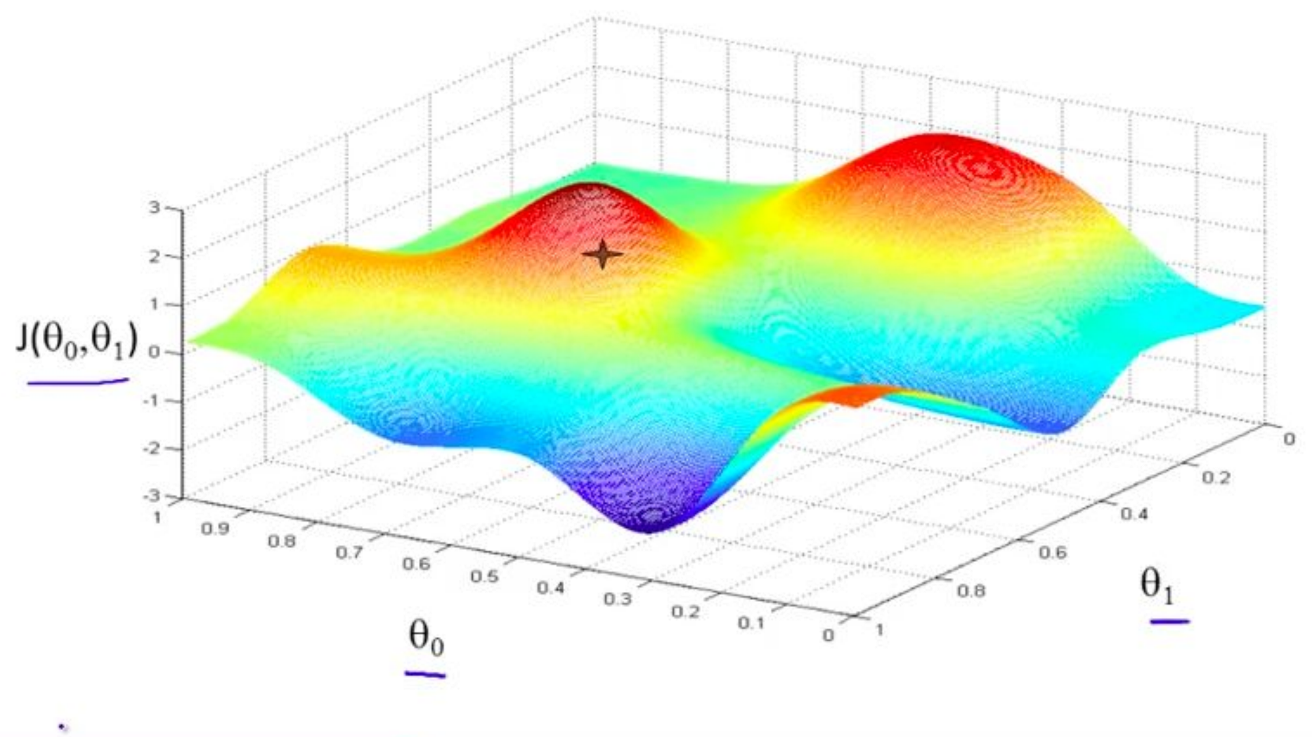
cost = 391992.58

It turns out that the cost function looks like this ...

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



a convex function that has global minima

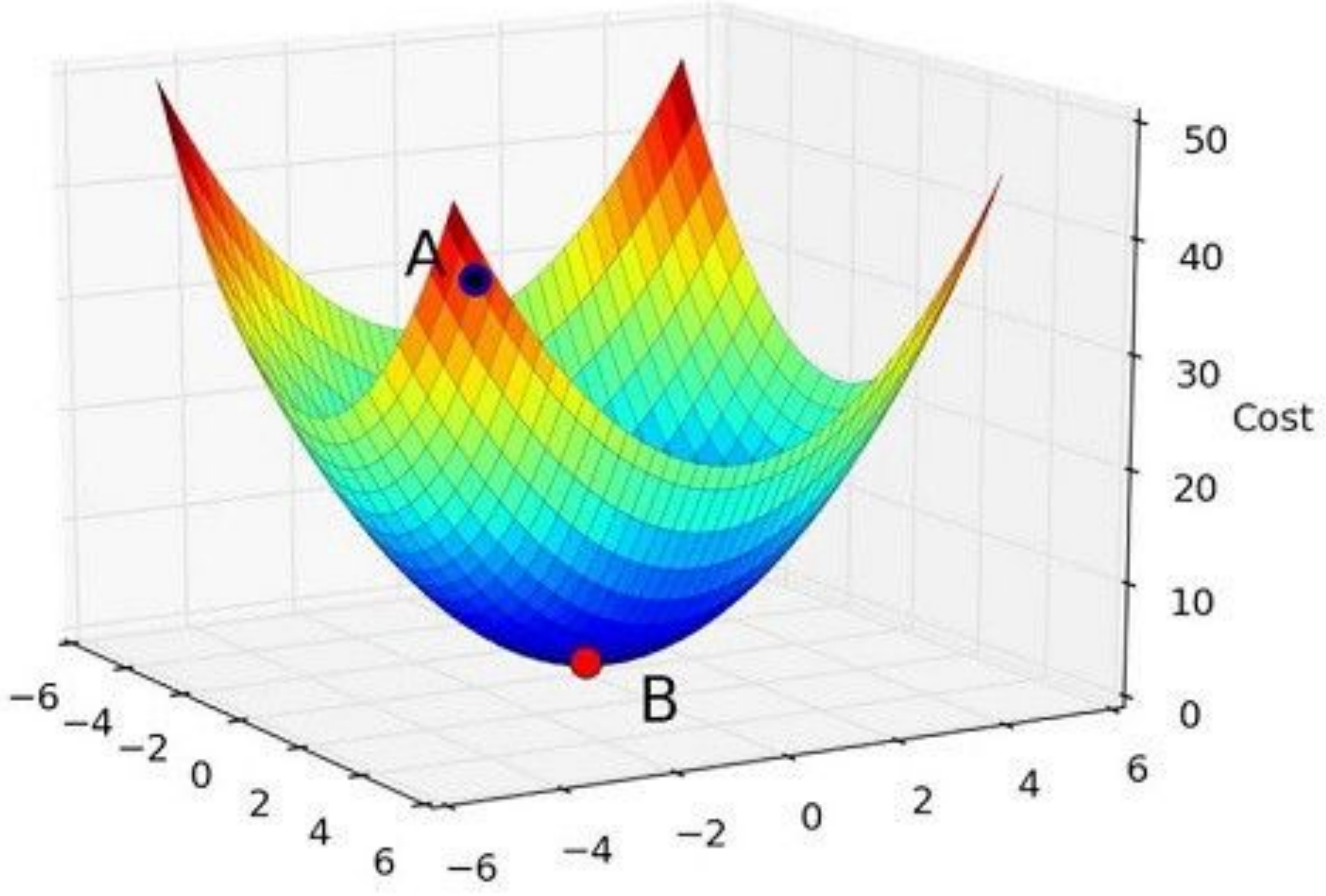


What can we do if the function is a convex function? :")

We can compute the derivatives (gradients) !!!

Gradient is the direction of the steepest ascent

Gradient will lead us from point A to point B (global minimum)

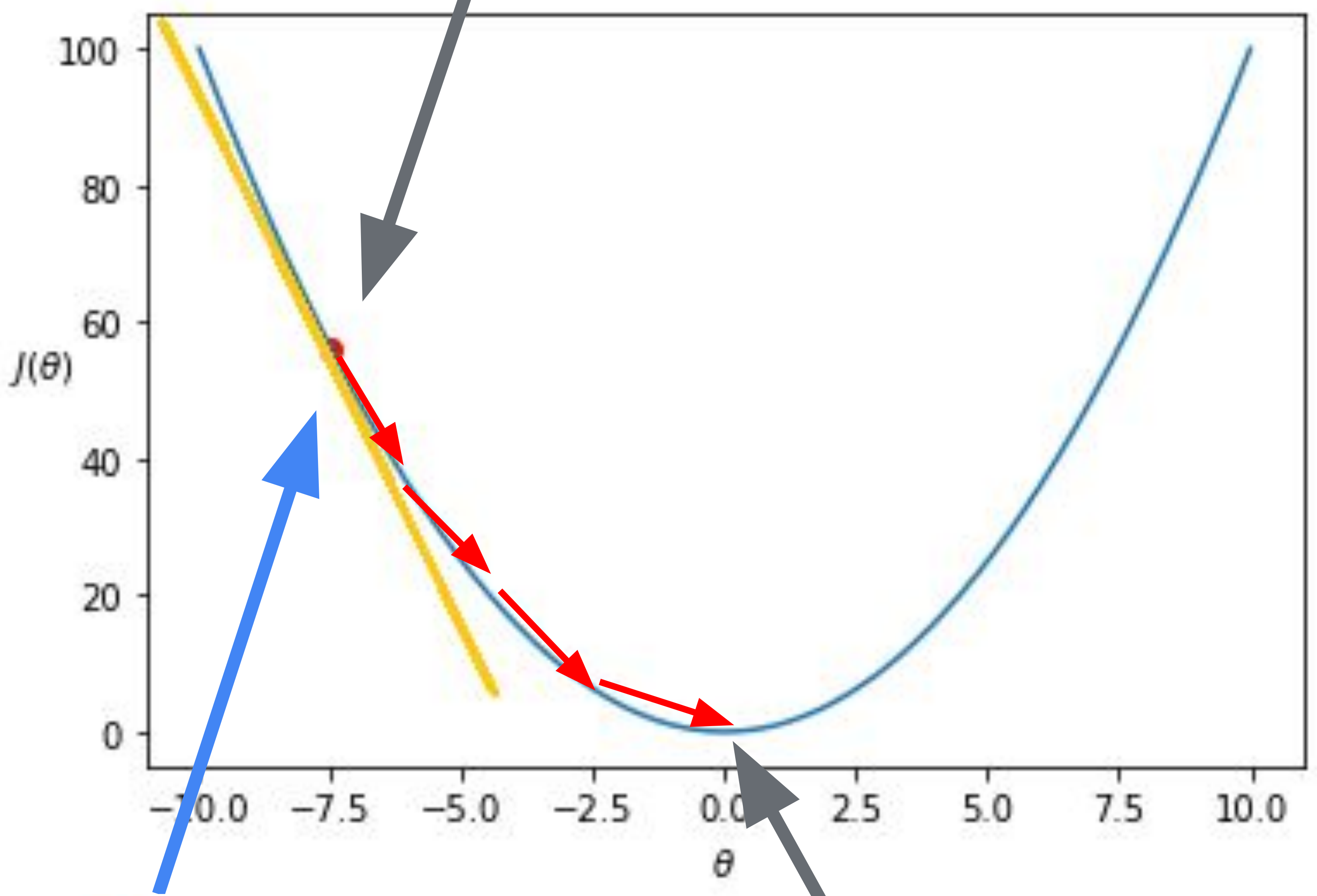


Google Developer Student Clubs

Assume this is our new cost (with only $\boldsymbol{\theta}$)

current θ

$J(\theta) = \theta^2$



$\frac{dJ}{d\theta}$

θ that we want

gradient when θ = -7.5 :

$$\begin{aligned}\frac{dJ}{d\theta} &= 2\theta \\ &= 2 \times -7.5 \\ &= -15\end{aligned}$$

repeat until convergence: $\{$

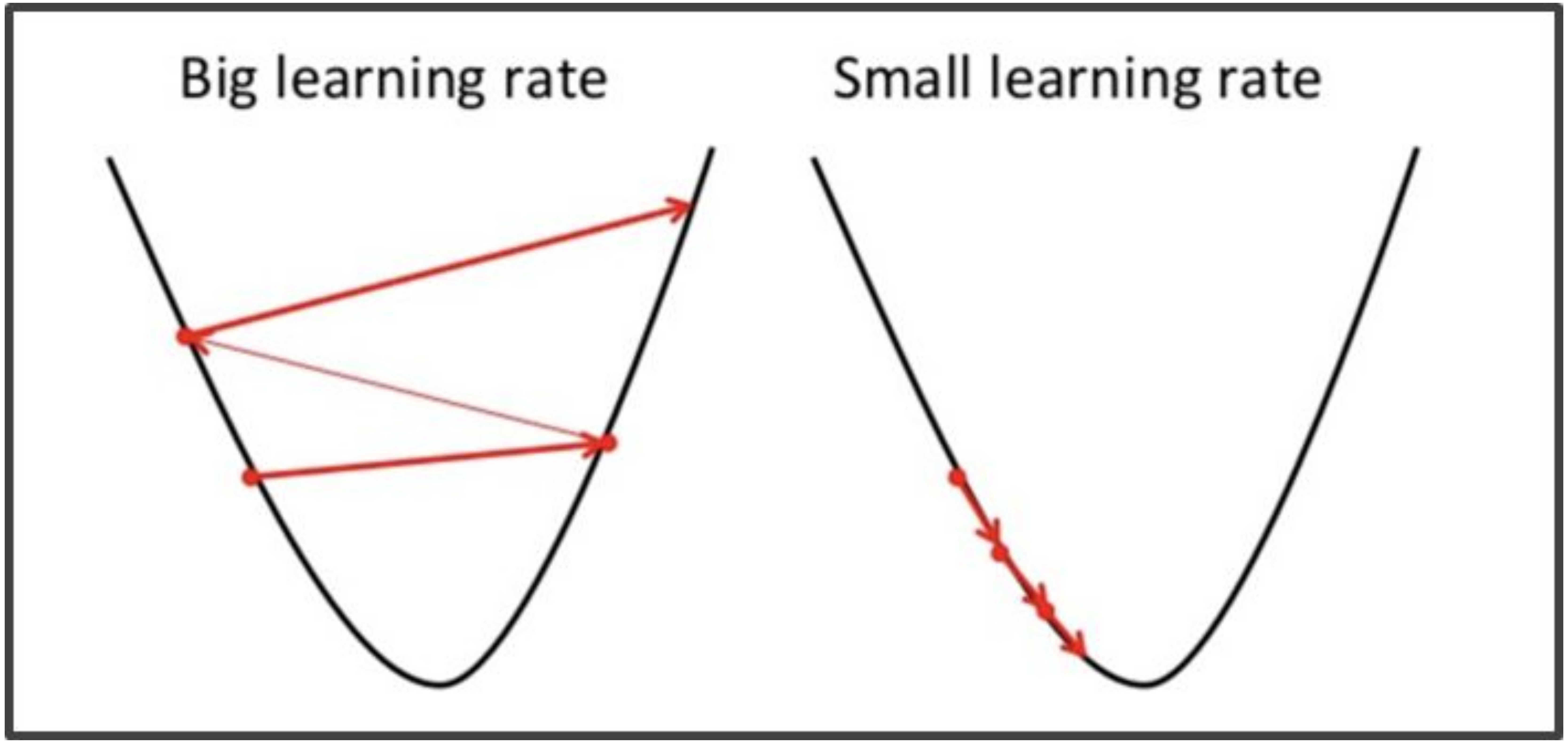
$$\theta := \theta - \alpha \frac{dJ(\theta)}{d\theta}$$

$\}$

let $\alpha = 0.4$

$$\begin{aligned}
\theta &= -7.5 - 0.4(2 \times -7.5) = -1.5 \\
\theta &= -1.5 - 0.4(2 \times -1.5) = -0.3 \\
\theta &= -0.3 - 0.4(2 \times -0.3) = -0.06 \\
\theta &= -0.06 - 0.4(2 \times -0.06) = -0.012 \\
\theta &= -0.012 - 0.4(2 \times -0.012) = -0.0024 \\
&\vdots \\
\theta &\approx 0
\end{aligned}$$

Choosing learning rate, *α*



Gradient descent algorithm

Initialize θ_0 and θ_1 randomly
repeat until convergence: {

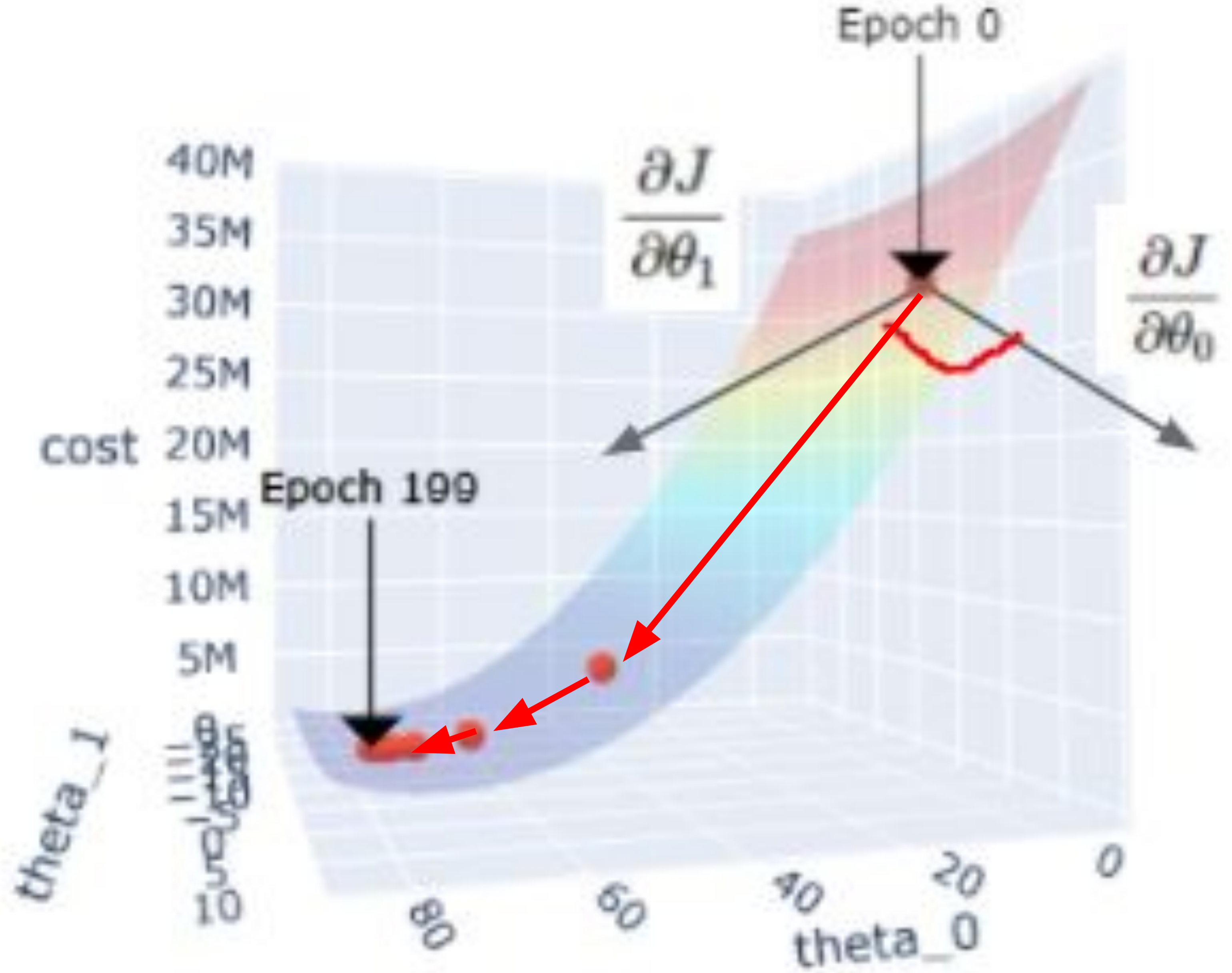
$$\theta_0 := \theta_0 - \alpha \frac{\partial J(\theta)}{\partial \theta_0}$$

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

}

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

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$



Gradient is the direction of the steepest ascent at certain axis (e.g., x-axis, y-axis)

Derivatives of J(θ) w.r.t θ_0 and θ_1 (Optional proof)

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

$$\begin{aligned}
\frac{\partial J}{\partial \theta_0} &= \frac{\partial}{\partial \theta_0}\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \\
&= \frac{\partial}{\partial \theta_0}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)})^2 \\
&= \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial \theta_0}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)})^2 \\
&= \frac{1}{2m}\sum_{i=1}^{m}(2)(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)})\frac{\partial}{\partial \theta_0}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \\
&\qquad\qquad \cdots + \theta_n x_n^{(i)} - y^{(i)}) \\
&= (\cancel{2})\frac{1}{\cancel{2}m}\sum_{i=1}^{m}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)}) \cdot x_0^{(i)} \\
&= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot 1 \\
&= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})
\end{aligned}$$

Derivatives of J(θ) w.r.t θ_0 and θ_1 (Optional proof)

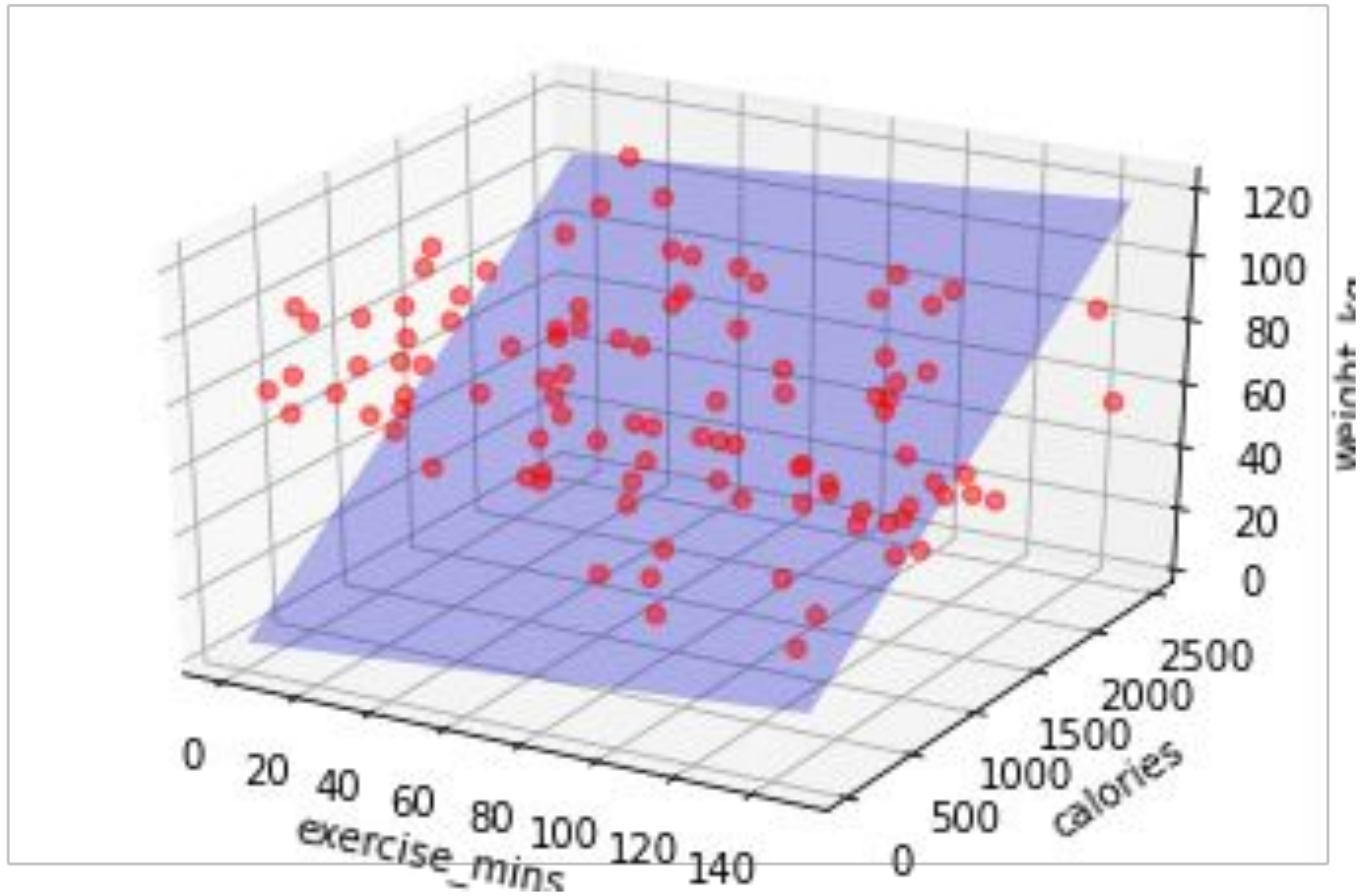
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$$

$$\begin{aligned}
\frac{\partial J}{\partial \theta_1} &= \frac{\partial}{\partial \theta_1}\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \\
&= \frac{\partial}{\partial \theta_1}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)})^2 \\
&= \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial \theta_1}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)})^2 \\
&= \frac{1}{2m}\sum_{i=1}^{m}(2)(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)})\frac{\partial}{\partial \theta_1}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots \\
&\qquad + \theta_n x_n^{(i)} - y^{(i)}) \\
&= (\cancel{2})\frac{1}{\cancel{2}m}\sum_{i=1}^{m}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)}) \cdot x_1^{(i)} \\
&= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}
\end{aligned}$$

Multivariate Linear Regression

We can have more than 1 feature, n > 1

exercise_mins	calories	weight_kg
62.553301	1089.987255	88.472200
108.048674	64.815580	38.643998
0.017156	1374.156195	91.202438
45.349886	1088.305982	67.213357
22.013384	1050.919505	93.927316
...	...	...
35.554047	1041.883600	81.424655
135.506928	1609.604835	50.707745
86.051923	1653.703317	71.596434
0.430549	426.192834	78.770789
92.571737	2204.130589	82.572642



n = 2

General form of X, y, and θ

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

(m x n)

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

(m x 1)

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

(n+1 x 1)

$$h(X) = \begin{bmatrix} h(x^{(1)}) \\ h(x^{(2)}) \\ h(x^{(3)}) \\ \vdots \\ h(x^{(m)}) \end{bmatrix}$$

(m x 1)

$$X_{prep} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

(m x n+1)

General form (m training instances, n features)

Hypothesis: $h(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \ldots + \theta_n x_n^{(i)}$

Parameters:
$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Cost function:
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{\partial J(\theta)}{\partial \theta_0}$$

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

$\ldots$

$$\theta_n := \theta_n - \alpha \frac{\partial J(\theta)}{\partial \theta_n}$$

}

Vectorized version

$$h(X_{prep}) = X_{prep} \cdot \vec{\theta}$$

$$J(\theta) = \frac{1}{2m} (h(X_{prep}) - \vec{y})^{\top} (h(X_{prep}) - \vec{y})$$

repeat until convergence: {

$$\vec{\theta} := \vec{\theta} - \alpha \nabla J(\theta)$$

}

General form (m training instances, n features)

Gradient of cost function:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

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

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \cdot \left(h_\theta(x^{(i)}) - y^{(i)} \right) \quad \text{for } j = 1 \text{ to } n$$

Vectorized version

$$\nabla J(\theta) = \frac{1}{m} X_{prep}^{\top} (h(X_{prep}) - \vec{y})$$

Model hypothesis (vectorized ver) (optional proof)

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \longrightarrow X_{prep} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

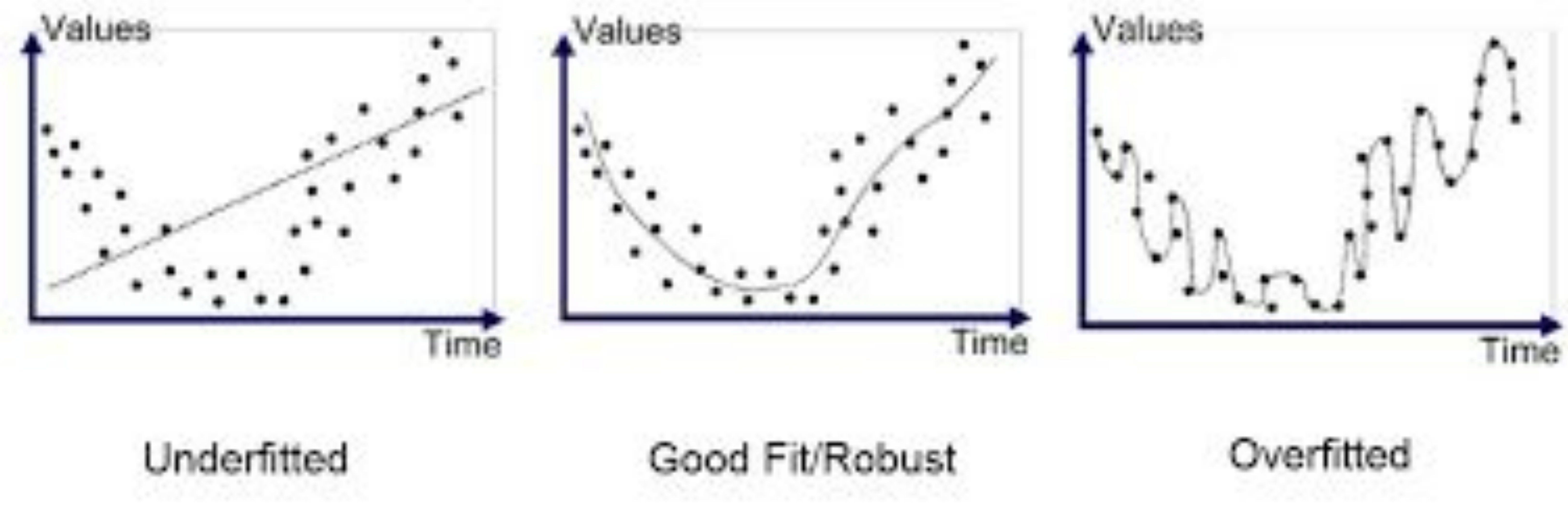
(m x n) → (m x n+1)

$$\begin{aligned} h(X_{prep}) &= X_{prep} \cdot \theta \\ &= \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \\ &= \begin{bmatrix} \theta_0 \cdot 1 + \theta_1 x_1^{(1)} + \theta_2 x_2^{(1)} + \dots + \theta_n x_n^{(1)} \\ \theta_0 \cdot 1 + \theta_1 x_1^{(2)} + \theta_2 x_2^{(2)} + \dots + \theta_n x_n^{(2)} \\ \vdots \\ \theta_0 \cdot 1 + \theta_1 x_1^{(m)} + \theta_2 x_2^{(m)} + \dots + \theta_n x_n^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h(x^{(1)}) \\ h(x^{(2)}) \\ \vdots \\ h(x^{(m)}) \end{bmatrix} \end{aligned}$$

(m x n+1) x (n+1 x 1) → (m x 1)

Overfitting/underfitting

- Overfitting is the situation where the model fits the training set too well, that it fails to generalise the real-world data (data that’s not used for training).
- Underfitting is the situation where the model does not fit the training set well.
- Both affects the performance of the model



Data splitting

- Data is usually split into three sets, namely training , cross validation and test sets. The objective of data splitting is to make sure that the model performs well with new data.
- Data splitting makes sure that the model does not know the test dataset since the model is trained with training set only.

Coding session

Get started by clicking the link below:

https://colab.research.google.com/github/dscum/Head-Start-ML/blob/main/session-1/workshop%201%20nb%20(live%20ver).ipynb

Q&A

Google Developer Student Clubs

Leave us your feedback

https://forms.gle/oVrJEvExPPEA6mem9