

# Introduction to Deep Learning

## Review of Linear Regression

# Example

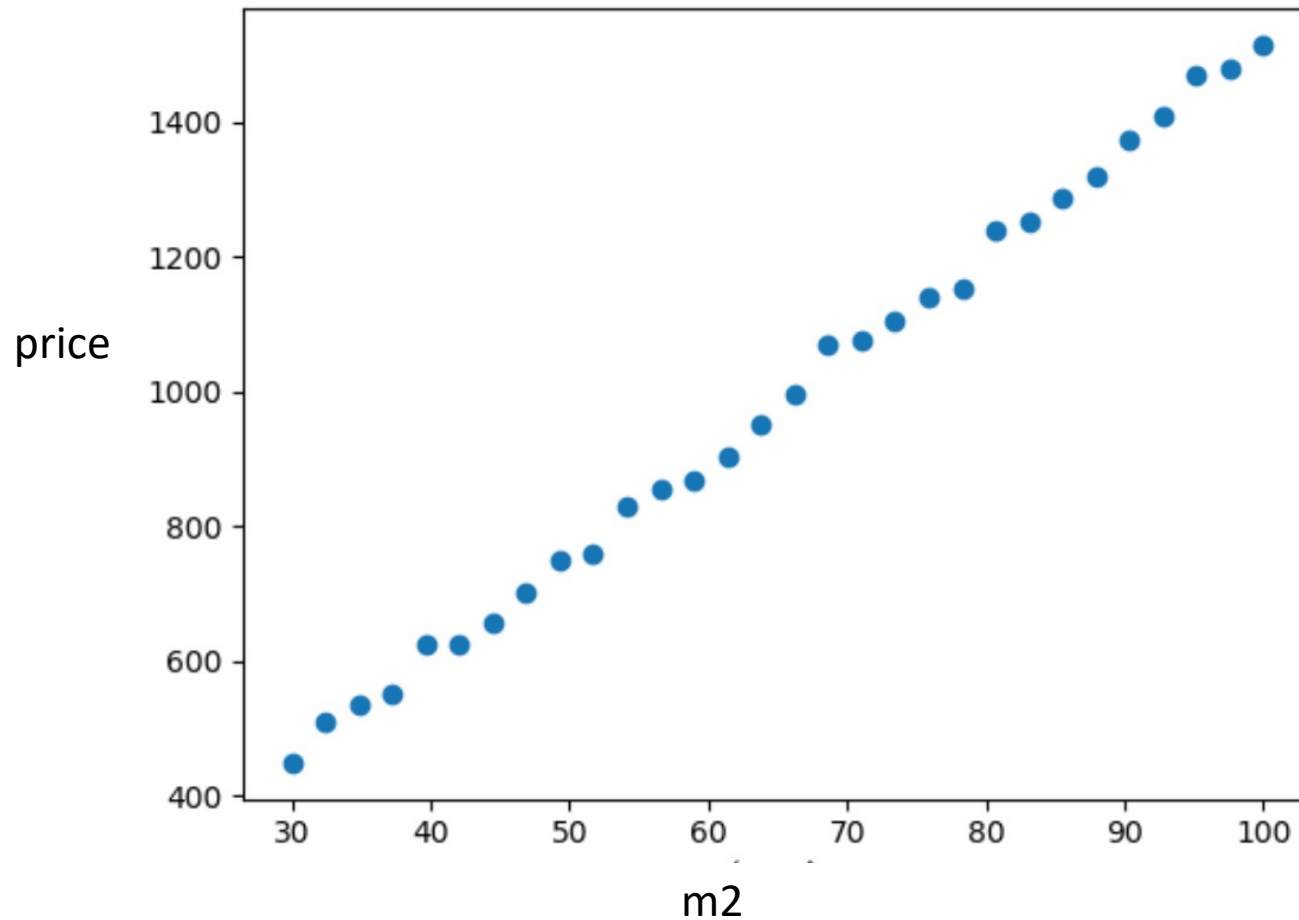
- Problem: House selling price prediction
- Input: Having data about areas and selling prices of 30 houses as in table 1 below:

Area (m2)	Selling price (Vnese million)
30	448.524
32.4138	509.248
34.8276	535.104
37.2414	551.432
39.6552	623.418
...	...

Table 1: Dataset for house selling price prediction

# Visualization of table 1 data

Relationship between house selling price and house area



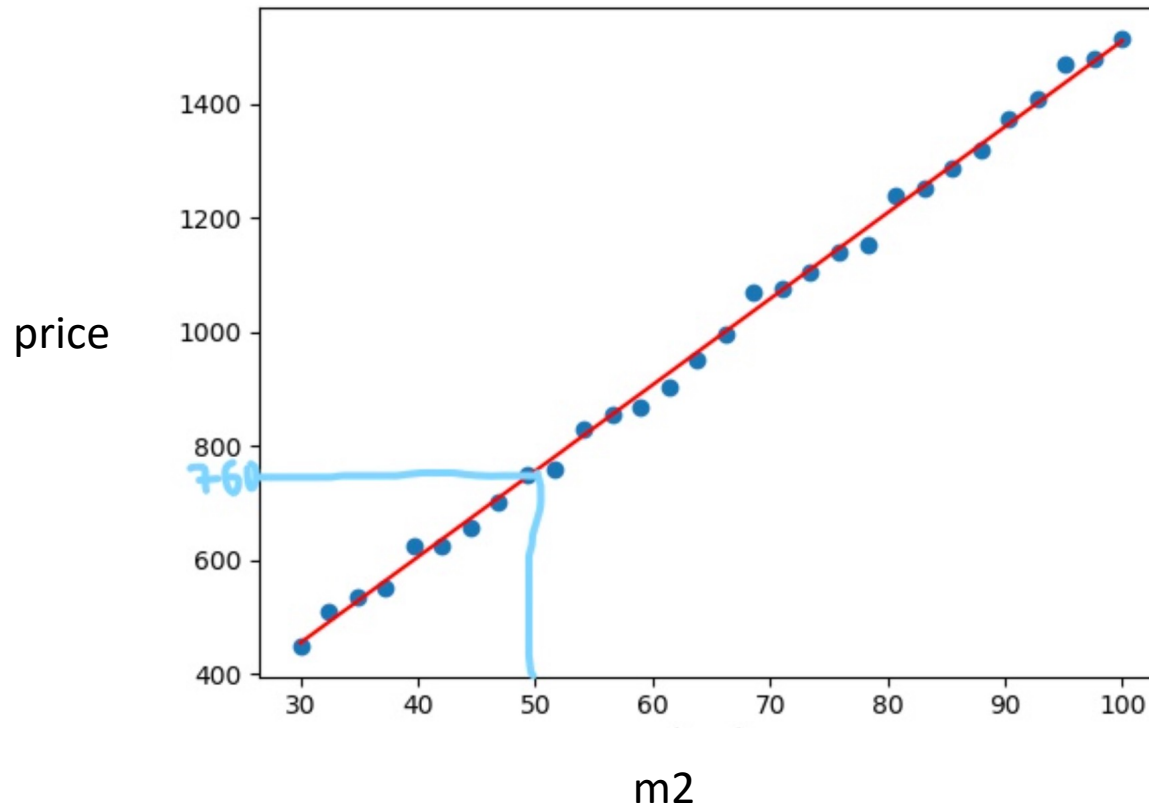
# Example (next)

- Requirement: estimate the selling price of a 50 square meter
- Output: estimated price?

# Solution: Idea

**Solution:** Draw a line closest to the data points and calculate the house price at 50

Estimated price of 50-m<sup>2</sup> house



# Solution: Programming

- Step 1- Training: Find the line closest to the data points (called *model*)

# Solution: Programming

- Step 1- Training: Find the line closest to the data points (called *model*) → using ***Gradient descent algorithm***

# Solution: Programming

- Step 1- Training: Find the line closest to the data points (called *model*) → using ***Gradient descent algorithm***
- Step 2 - Prediction: Predict how much a 50-m<sup>2</sup> house will cost based on the trained model



# Formulating model

- Model formula:  $y = w_1 * x + w_0$

# Formulating model

- Model formula:  $y = w_1 * x + w_0$   
→ Linear Model

# Formulating model

- Model formula:  $y = w_1 * x + w_0$   
→ Linear Model

- Problem becomes: find  $w_1, w_0$

- Represent input data points as:

$$\{(x_i, y_i), i = 1 \dots 30\}$$

In which:  $y_i = w_1 * x_i + w_0$

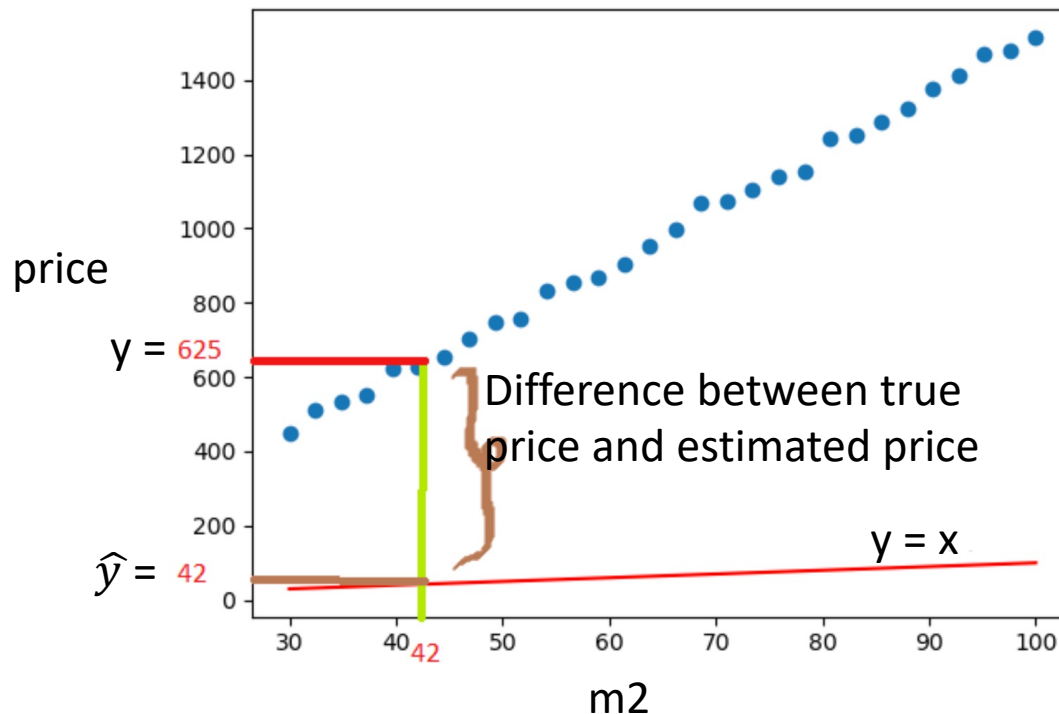
- Represent estimated data point as:

$$\hat{y}_i = w_1 * x_i + w_0$$

# Model training

- Random initial data point:  $w_1 = 1, w_0 = 0 \rightarrow$  Model becomes:  $y = x$
- Model fine-tuning

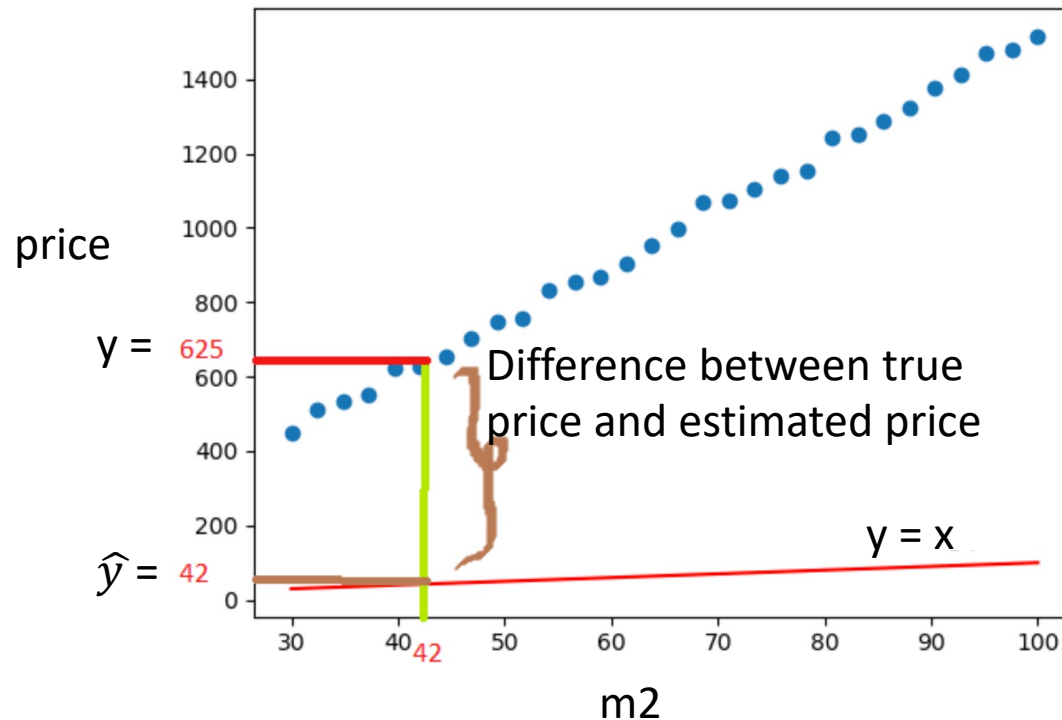
Difference between true price and estimated price at the data point  $x = 42$  of linear model  $y = x$



# Model training

- Problem: estimated price is too far from true price. For example, at the point  $x = 42$

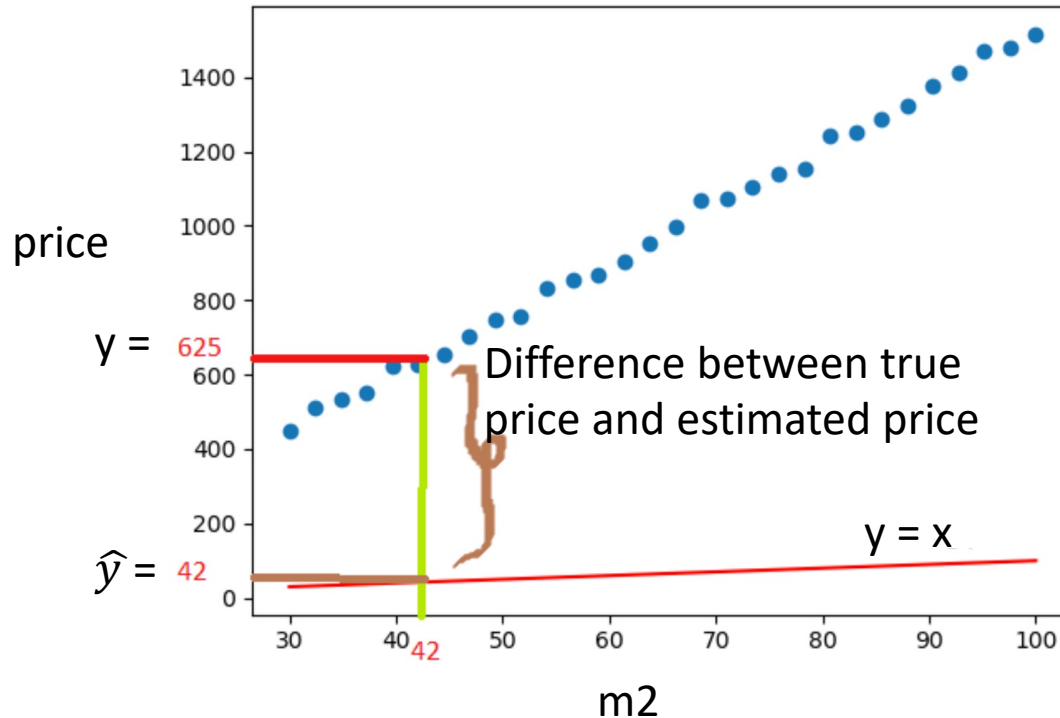
Difference between true price and estimated price at the data point  $x = 42$  of linear model  $y = x$



# Model training

- Need a metric to **evaluate** the linear model with parameter set:  $(w_0, w_1) = (0, 1)$

Difference between true price and estimated price at the data point  $x = 42$  of linear model  $y = x$



# Loss Function

- For each data point  $(x_i, y_i)$ , the difference between the actual price and the predicted price:

$$\frac{1}{2} * (\hat{y}_i - y_i)^2$$

- The difference across the entire data set as the sum of the differences of each data point:

$$J = \frac{1}{2} * \frac{1}{N} * (\sum_{i=1}^N (\hat{y}_i - y_i)^2)$$

Where N is number of data points

# Loss Function

$$J = \frac{1}{2} * \frac{1}{N} * (\sum_{i=1}^N (\hat{y}_i - y_i)^2)$$

- $J \geq 0$
- The smaller  $J$  is, the model is more close to the actual data points
- If  $J = 0$  then the model passes through all data points

→  $J$  is called the **loss function**



# Loss Function

$$J = \frac{1}{2} * \frac{1}{N} * (\sum_{i=1}^N (\hat{y}_i - y_i)^2)$$

- The problem transfers from: finding the linear model  $y = w_1 * x + w_0$  closest to the data points
- → to: finding the parameter  $(w_0, w_1)$  such that J obtains the **minimum** value
- Use ***Gradient descent*** algorithm to find minimum value of J

# Gradient Descent Algorithm

- Idea: use derivative to find the minimum value of a function  $f(x)$
- Algorithm:
  - (1) Random initialization:  $x = x_0$
  - (2) Assign:  $x = x - \text{learning\_rate} * f'(x)$
  - (3) Re-compute  $f(x)$ . Stop if  $f(x)$  is small enough, or repeat step (2) if not

# Gradient Descent Algorithm

## Note:

- learning\_rate is non-negative constant
- step 2 will be repeated until a large enough number of times or  $f(x)$  is small enough

# Gradient Descent: example

**Problem:** Find minimum value of  $f(x) = x^2$   
using gradient descent algorithm

# Gradient Descent: example

**Problem:** Find minimum value of  $f(x) = x^2$  using gradient descent algorithm

- Solution 1: use Linear Algebra
- Solution 2: use gradient descent algorithm

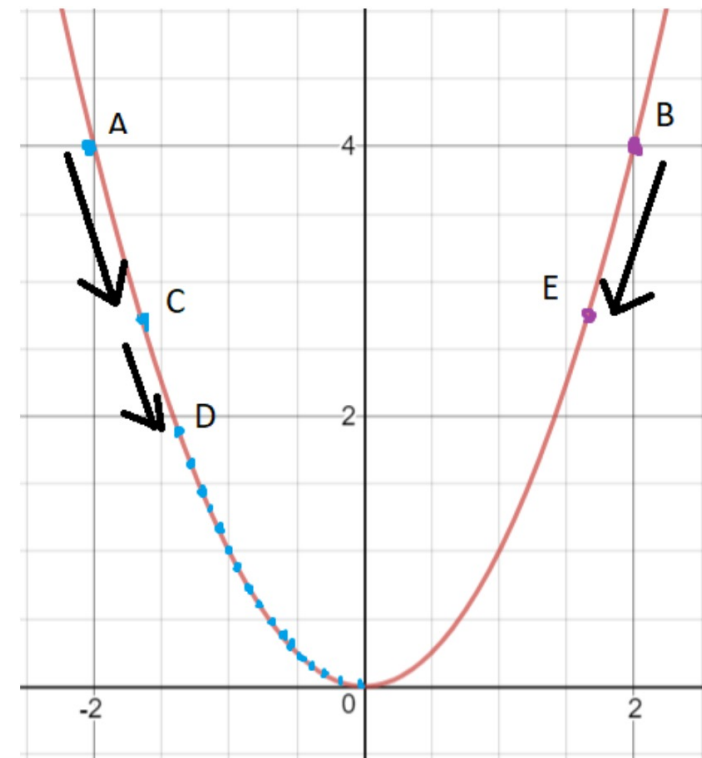
# Gradient Descent: example

Step 1: Random initialization  $x = -2$  (Point A)

Step 2: compute  $f'(x)$

$$\text{then } x = x_A - \text{learning\_rate} * f'(x_A)$$

Step 3: compute  $f(x) \rightarrow$  still big  $\rightarrow$  move to point C, and repeat Step 2



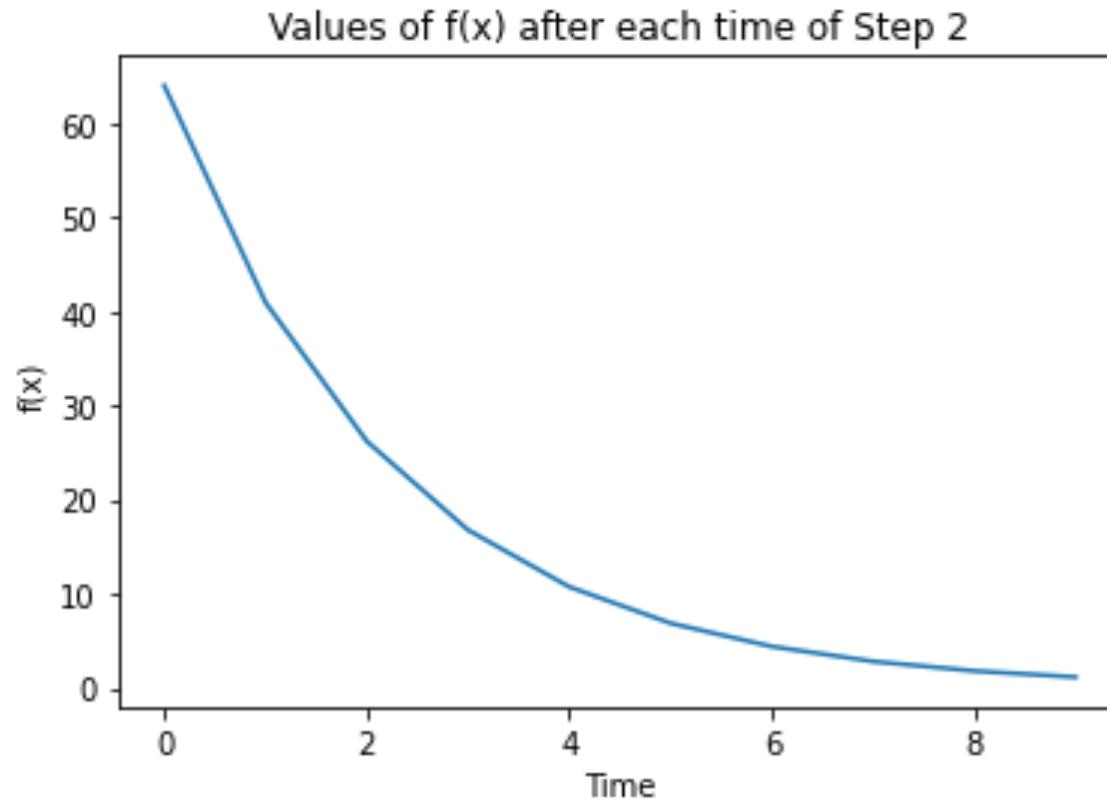
# Gradient Descent: example

In detail: if we choose initial value:  $x = 10$ ,  $\text{learning\_rate} = 0.1$ , then the values of step 2 and step 3 will be as in the following table:

Time	x	f(x)
1	8.00	64.00
2	6.40	40.96
3	5.12	26.21
4	4.10	16.78
5	3.28	10.74
6	2.62	6.87
7	2.10	4.40
8	1.68	2.81
9	1.34	1.80
10	1.07	1.15

Table 2: values of  $f(x)$  after 10 times of step 2 calculation

# Gradient Descent: example

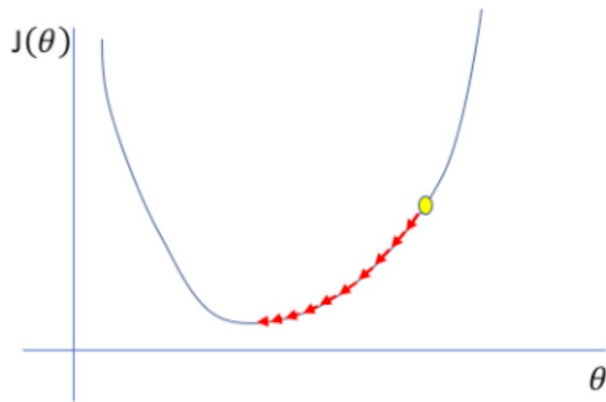


Visualization of Table 2



# Effect of Learning Rate Selection

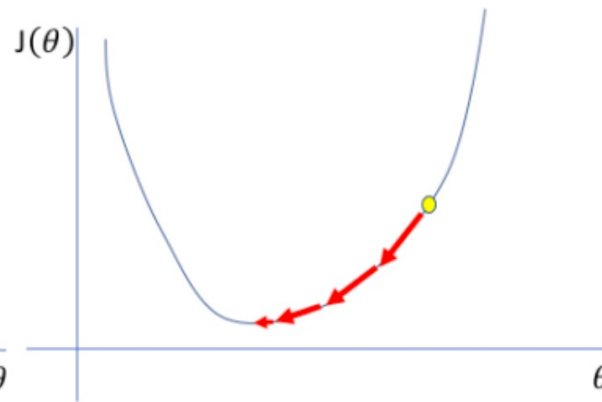
Too low



A small learning rate requires many updates before reaching the minimum point

Many times to calculate step 2

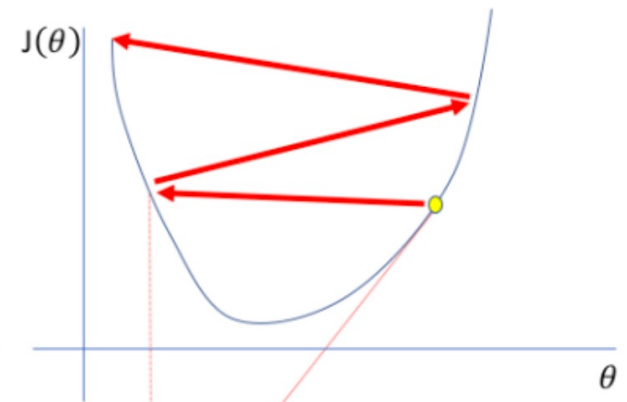
Just right



The optimal learning rate swiftly reaches the minimum point

Proper times to calculate step 2

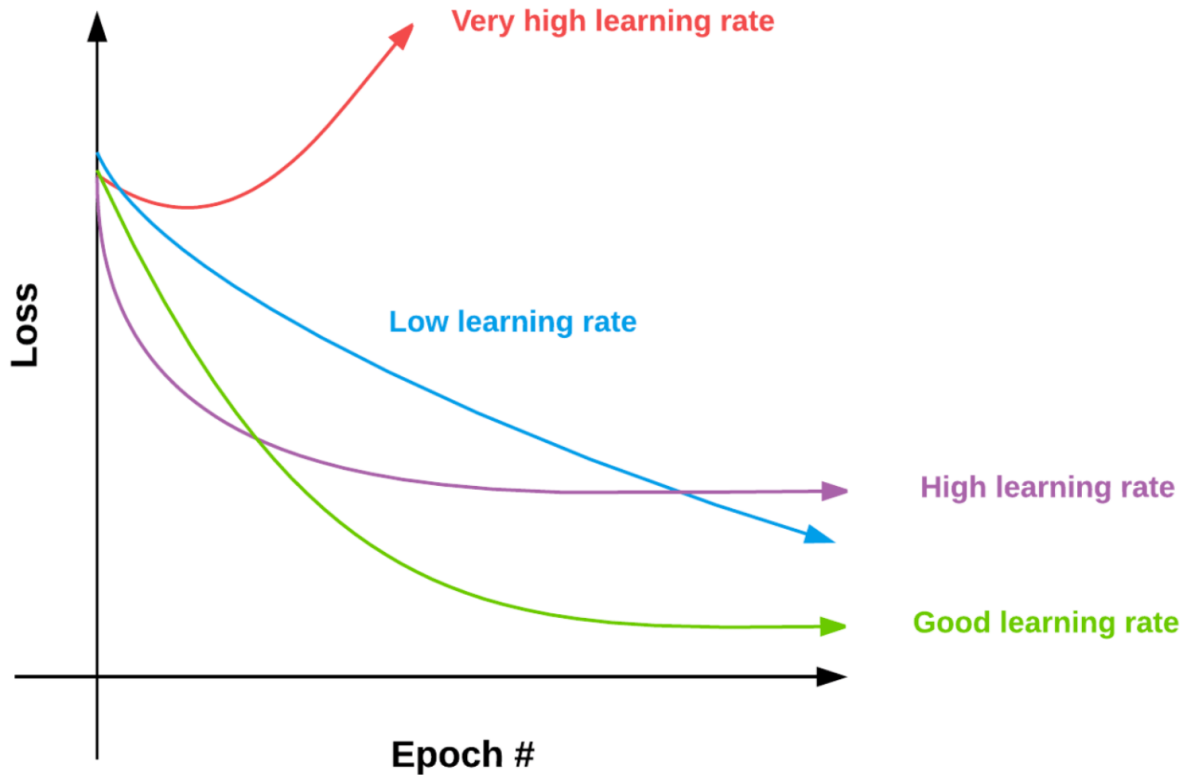
Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

Overshoot problem

# Effect of Learning Rate Selection



Epoch: Number of times of step 2, Loss: the function to find the minimum value

# Practical Work 01

- Setup python environment
  - Anaconda
  - Or virtualenv
  - ...
- Write Python code to plot data on previous table

# Practical Work 01

Time	x	f(x)
1	8.00	64.00
2	6.40	40.96
3	5.12	26.21
4	4.10	16.78
5	3.28	10.74
6	2.62	6.87
7	2.10	4.40
8	1.68	2.81
9	1.34	1.80
10	1.07	1.15

