Review Linear Regression

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH

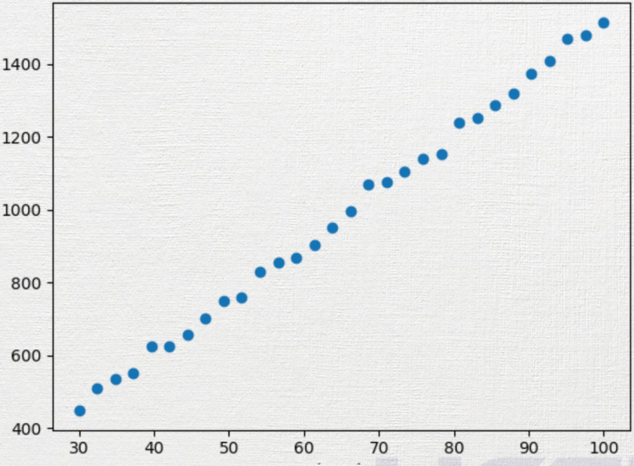
Review

Example

- Problem: House selling price prediction
- Input: Having data about areas and selling prices of 30 houses

Area (m^2)	Selling price (M VND)
30	448.524
32.4138	509.248
34.8276	535.104
37.2414	551.432
39.6552	623.418
....	....

Example



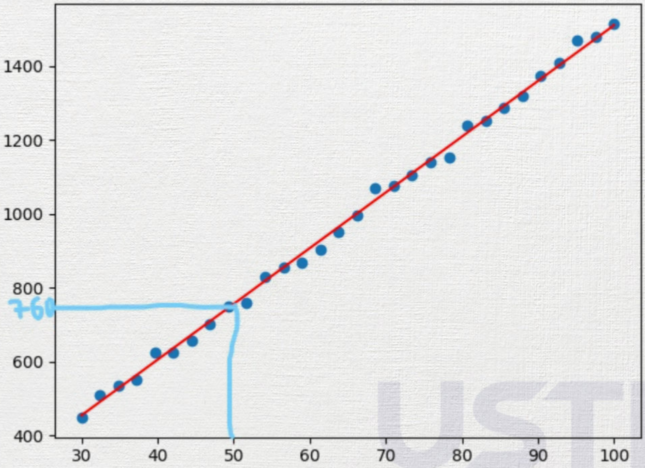
Visualization

Example

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

Solution

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



Solution

- Step 1 (Training)
  - Find the line closest to the data points (called model)
  - Gradient descend algorithm
- Step 2 (Prediction)
  - Predict how much a $50m^2$ house will cost based on the trained model

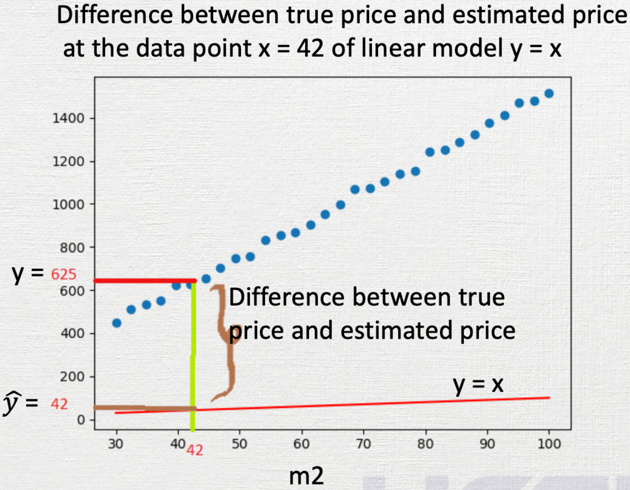
Formulation

- Linear model : $y = w_1 * x + w_0$
- Problem becomes: find w_1, w_0
- Represent input data points as: $(x_i, y_i), i = 1...30$
  - In which: $y_i = w_1 * x_i + w_0$
- Represent estimated data point: $\hat{y_i} = w_1 * x_i + w_0$

Training

- Random initial data point: $w_1 = 1, w_0 = 0$
  - Original model $y = w_1 * x + w_0$
  - Our model becomes: $y = x$
- Fine-tuning

Training



Too faaaaaaaaar!

Training

- How to reduce the difference?
  - Find a way to measure the difference
  - Evaluate the difference
  - Tune w_1, w_0

Loss Function

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

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

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

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

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 >= 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

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

- From: finding the linear model $y = w_1 * x + w_0$ closest to the data points
- To: finding the parameter (w_1, w_0) such that J obtains the minimum value
- Use Gradient descend algorithm to find minimum value of J

Gradient descend

- An **iterative** algorithm to find minimum value
- Idea: use derivative to find the minimum value of a function $f(x)$
    1. Random initialization: $x = x_0$
    2. Assign: $x = x - r * f'(x)$
    3. Re-compute $f(x)$. Stop if $f(x)$ is small enough, or repeat step 2 if not[1]

Where r >= 0 is called the *learning rate*.

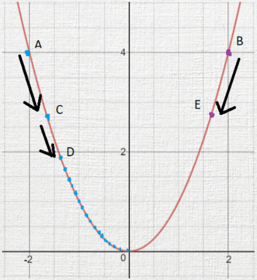
[1] can be many times!

Gradient descend Example

- Problem: Find minimum value of $f(x) = x^2$ using gradient descend algorithm
  - Use linear algebra
  - Use gradient descend

Gradient descend Example

- $f(x) = x^2$, therefore $f'(x) = 2x$
- Step 1: Random initialization $x = -2$ (Point A)
- Step 2: compute $f'(x)$ for
  - $x = x_A - L * f'(x_A) = x_A - 2 * L * x_A$
- Step 3: compute $f(x)$. Still big?
  - Move to point C
  - Repeat Step 2

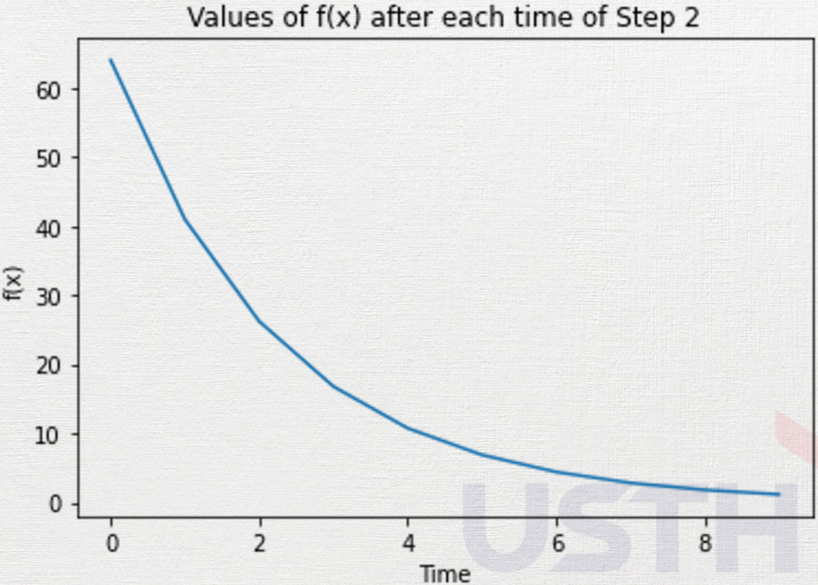


Gradient descend Example

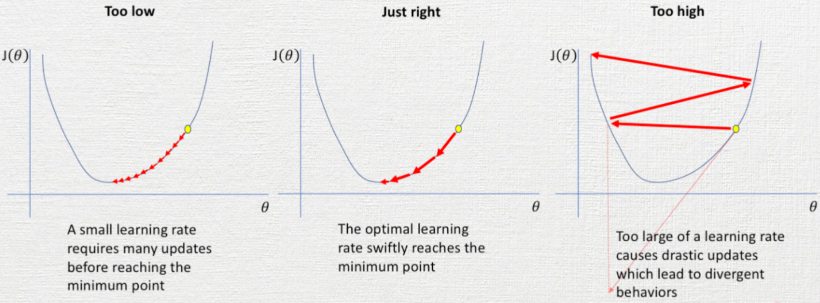
- In detail: $x_0 = 10, L = 0.1$, then the values of step 2 and step 3 will be

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

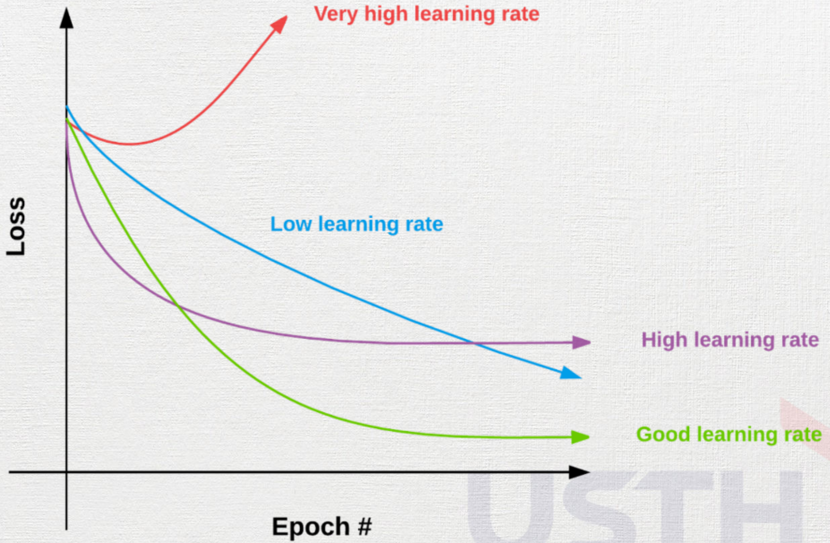
Gradient descend Example



Learning Rate



Learning Rate



Practice!

Labwork 0: Hello world!

- Fork course's github repository - `https://github.com/SonTG/dl2024`
- Clone your forked repository
- Update README.md with your name
- Commit and push the change to your forked repository

Labwork 1: Gradient Descend

- Implement (from scratch!) gradient descend to find minimum value of a given function $f(x)$ and its first order derivative $f\_()$
    - Print the intermediate iterative steps, similar to the previous table (time, x, $f(x)$)
    - Try experimenting with the previous example $f(x) = x^2$
- Write a report (in LaTeX):
    - Name it « Report.1.Gradient.descend.tex »
    - How you implement the algorithm
    - Analyze the effect of different learning rate r
- Push your code and report to your forked repository

Labwork 2: Linear Regression

- Implement (from scratch!) linear regression using previous gradient descend code to optimize w_1 and w_0
  - Input: a CSV file
  - Output: w_1, w_0
  - Print the intermediate iterative steps
- Try experimenting with the previous example of house price
- Write a report (in LaTeX):
  - Name it « Report.2.Linear.Regression.tex »
  - How you implement the algorithm
  - Analyze the effect of different learning rate r for convergence
- Push your code and report to your forked repository

Labwork 2: Linear Regression (extras)

- Gradient descend for linear regression with mean squared error loss
- Remind: single loss value

$$L_i = \frac{1}{2} * (\hat{y}_i - y_i)^2 \tag{3}$$

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

$$L_i = \frac{1}{2} * (w_1 * x_i + w_0 - y_i)^2 \tag{4}$$

Labwork 2: Linear Regression (extras)

- Loss of all data points

$$J = \frac{1}{N}\sum_{i=1}^{N} L_i = \frac{1}{2} * \frac{1}{N} * \sum_{i=1}^{N}(w_1 * x_i + w_0 - y_i - y_i)^2 \qquad (5)$$

- Minimize this function w.r.t w_0, w_1
  - **2 variables !!!**
  - Needs parital derivatives

Labwork 2: Linear Regression (extras)

- Remind: gradient descend
  - Initial value x_0
  - Function $f(x)$
  - First order derivative $f'(x)$
  - Learning rate r
  - Threshold t

Labwork 2: Linear Regression (extras)

- 2D gradient descend for linear regression weights w_0, w_1
  - Initial value $w_0^{(0)}, w_1^{(0)}$
  - Function $f(w_0, w_1)$
  - First order parital derivatives $\frac{\mathrm{d}f}{\mathrm{d}w_0}, \frac{\mathrm{d}f}{\mathrm{d}w_1}$
  - Learning rate r
  - Threshold t

Labwork 2: Linear Regression (extras)

- Function to calculate single loss value:

$$f(w_0, w_1) = L_i(w_0, w_1) = \frac{1}{2} * (w_1 * x_i + w_0 - y_i)^2 \qquad (6)$$

- Therefore parital derivatives over w_0 and w_1 are

$$\frac{\mathrm{d}L}{\mathrm{d}w_0} = w_1 * x_i + w_0 - y_i \qquad (7)$$

$$\frac{\mathrm{d}L}{\mathrm{d}w_1} = x_i * (w_1 * x_i + w_0 - y_i) \qquad (8)$$

Labwork 2: Linear Regression (extras)

$$f(w_0, w_1) = L_i(w_0, w_1) = \frac{1}{2} * (w_1 * x_i + w_0 - y_i)^2$$

$$\frac{\mathrm{d}L}{\mathrm{d}w_0} = w_1 * x_i + w_0 - y_i, \text{ and } \frac{\mathrm{d}L}{\mathrm{d}w_1} = x_i * (w_1 * x_i + w_0 - y_i)$$

- 2D gradient descend for linear regression weights w_0, w_1
  - Step 1: Random initialization $w_0 = 0, w_1 = 1$
  - Step 2: descend...
    - $w_0 = w_0 - r * \frac{\mathrm{d}L}{\mathrm{d}w_0} = w_0 - r * (w_1 * x_i + w_0 - y_i)$
    - $w_1 = w_1 - r * \frac{\mathrm{d}L}{\mathrm{d}w_1} = w_1 - r * x_i * (w_1 * x_i + w_0 - y_i)$
  - Step 3: compute $f(w_0, w_1)$. Still big?
    - Move to point (w_0, w_1)
    - Repeat Step 2