Tishreen University
Information Engineering Faculty
Artificial Intelligence Department
Fourth Year

Artificial Neural Networks

Lecture 2 : Linear Regression & Gradient descent

Dr. Wisam Ibrahim

2024-2025 Semester 1

LINEAR REGRESSION

- **House pricing data example**

Supervised learning Linear Regression problem

Training set :

Area x's	Price y' s
150	15,000
100	11,000
75	6,500
200	21,000
250	????????

m = number of **training examples(samples)**

x's = input variables / features

y's = output variable "target" variables

(x, y) - single training sample

(x_i , y_i) - specific example (i training example) , i is an index to training set.

LINEAR REGRESSION

The price of the house with an area of 250 m^2 can be predicted by finding a relationship between the area of the house and its price in the given information.

Therefore, it can be said that the price of the house can be about 25,000

Given that there is only one variable ***x*** (the area of the house), the relationship here is a similar to the line equation i.e.

$y = ax + b$

For two points

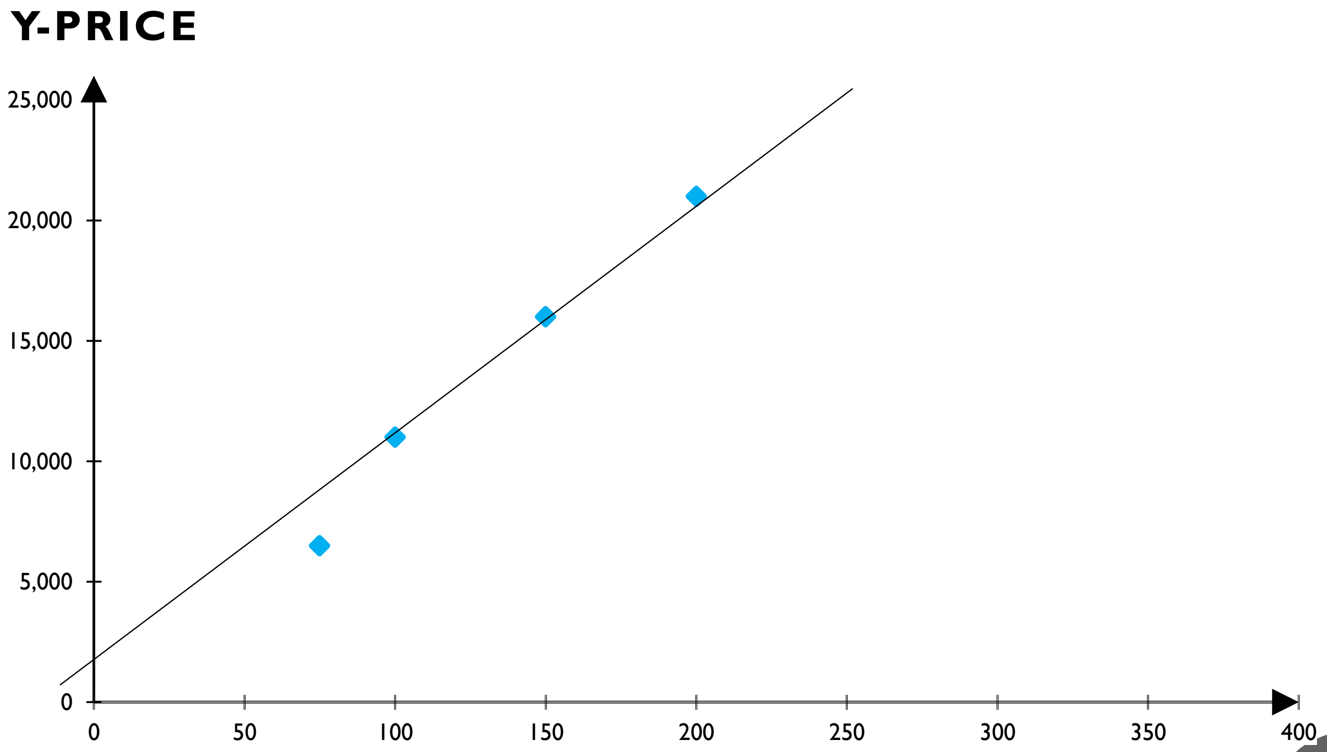
(100,11000) (150,15000)

a=80, b= 3000

So: ***y = 80x +3000***

Price for house with area ***250***

will be ***23000***



LINEAR REGRESSION

- Based on the previous line equation, the values of the previous dataset will be as follows:

Area x’s	Price y’ s
150	15,000
100	11,000
75	* 9,000
200	* 19,000
250	23,000

LINEAR REGRESSION

With our training set - how do we used it?

- Take training set
- Pass into a learning algorithm
- Algorithm outputs a function (denoted *h*) (h = **hypothesis)**

  This function takes an input (e.g. area of new house)
- Tries to output the estimated value of *Y*
- How do we represent hypothesis *h?*

Going to present *h* as:

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

h(x) (shorthand)

LINEAR REGRESSION

- What does this mean?
- Means Y is a linear function of x
- θ_i are parameters
- θ_0 is zero condition
- θ_1 is gradient
- This kind of function is a linear regression with one variable
- Also called **univariate linear regression**

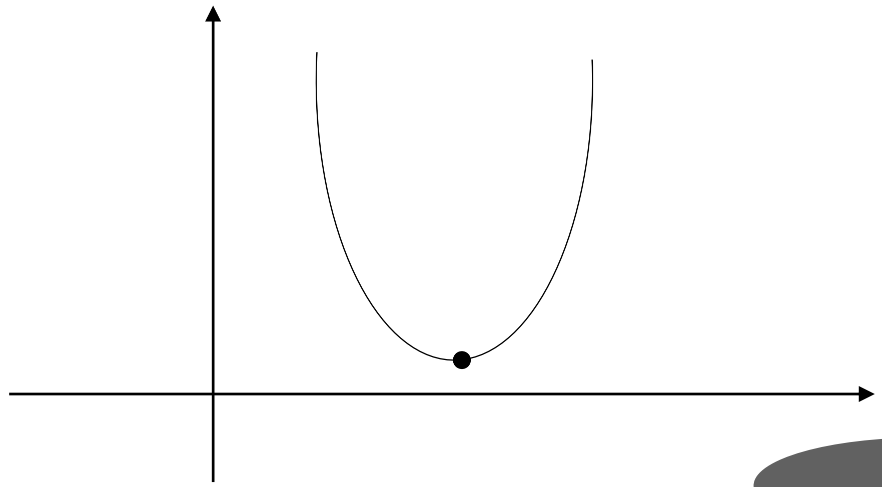
LINEAR REGRESSION - COST FUNCTION

- **Cost function** lets us figure out how to fit the best line to our data (Choosing values for θ parameters).
- Different values give you different functions.

- Based on our training set we want to generate parameters which make *h(x)* close to *y* for our training dataset.
- Basically, uses *x's* in training set with ***h(x)*** to give ***estimated output*** which is as close to the ***actual value*** ***y*** as possible.
- Think of *h (x)* as a "*y* imitator" - it tries to convert the *x* into *y*, and considering we already have *y* we can evaluate how well *h (x)* does this.

LINEAR REGRESSION - COST FUNCTION

- **To formalize this:**
- We want to solve a minimization problem
- Minimize $\boldsymbol{cost = \sum_i (h(x_i) - y_i)^2}$
- i.e. minimize the difference between *h(x)* and *y* for all training samples
- cost = $(15000-15000)^2 + (11000-11000)^2+(9000-6500)^2+(19000-21000)^2$
- The objective is reducing the cost function to the lowest possible value.
- we notice that the cost function is a parabola function:
- $y=ax^2$
- cost function is denoted as :
- $J(\theta_0, \theta_1) = J(\theta) = \frac{1}{2m} \sum_i (h_\theta(x_i) - y_i)^2$



LINEAR REGRESSION - COST FUNCTION

- $\frac{\mathbf{1}}{\boldsymbol{m}}$:means we determine the average
- **½** :makes the math a bit easier, and doesn't change the constants we determine at all (i.e. half of the smallest value is still the smallest value)
- Minimizing means we get the values of θ_0 and θ_1 which find on the average of minimal deviation of *h(x)* from *y* when we use those parameters in our hypothesis function
- *This cost function is also called the squared error cost function*
- **Hypothesis** - is like your prediction machine, throw in an *x* value, get a estimated *y* value
- **Cost** - is a way to - using your training data- determine values for your θ values which make the hypothesis as accurate as possible.

GRADIENT DESCENT ALGORITHM

- **Minimize cost function *J* by Gradient descent algorithm**:
- Used over all machine learning for minimization

- **Problem:**
- We have $J(\theta_0 , \theta_1)$
- We want to get min $J(\theta_0 , \theta_1)$

GRADIENT DESCENT ALGORITHM

- **Algorithm:**
- Start with initial guesses (Start at 0,0) or any other values.
- Keeping changing θ_0 and θ_1 a little bit to reduce $J(\theta_0 ,\theta_1)$
- Each time you change the parameters, you select the gradient which reduces $J(\theta_0 ,\theta_1)$ the most possible
- Repeat
- Do so until you converge to a local minimum.

GRADIENT DESCENT ALGORITHM

- Do the following until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{(for } j = 0 \text{ and } j = 1\text{)}$$

- Update θ_j by setting it to (θ_j - α times the partial derivative of the cost function with respect to θ_j)
- α (alpha) is a number called the learning rate (step value), controls how big a step you take.

  If α is big, we have an aggressive gradient descent.

  If α is small , algorithm takes tiny steps.

GRADIENT DESCENT ALGORITHM

- how is this gradient descent algorithm implemented?
- Do this for θ_0 and θ_1
- For $j = 0$ and $j = 1$ means we simultaneously update both
- How do we do this?
- Compute the right hand side for both θ_0 and θ_1
- So we need a temp value (temporary)
- Then, update θ_0 and $\theta_1$at the same time

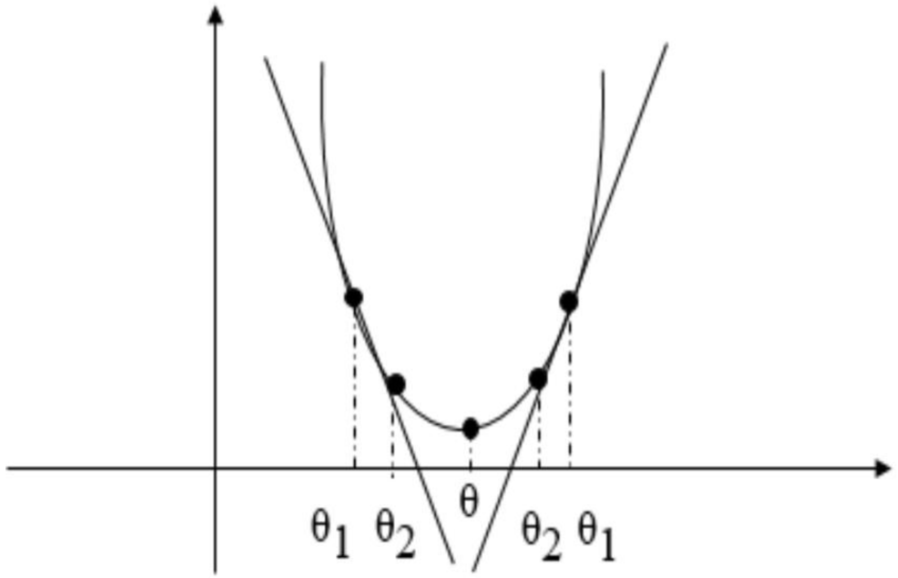
$$\begin{aligned}
\text{temp0} &:= \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
\text{temp1} &:= \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\
\theta_0 &:= \text{temp0} \\
\theta_1 &:= \text{temp1}
\end{aligned}$$

GRADIENT DESCENT ALGORITHM

- **Understanding the algorithm**
- To understand gradient descent, we'll return to a simpler function, where we minimize function with one parameter to help explain the algorithm
- min $J(\theta_j)$ where θ_j is a real number
- Two key terms in the algorithm
  - Alpha
  - Derivative term
- Partial derivative vs. derivative
- Use partial derivative when we have multiple variables but only derive with respect to one.
- Use derivative when we are deriving with respect to all the variables
- Derivative term: $\frac{\partial J(\theta)}{\partial \theta} J(\theta_j)$

GRADIENT DESCENT ALGORITHM

- Lets take the tangent at the point and look at the slope of the line.
- If the value of the tangent (the slope of the line) at a point is positive (the angle of the line with the axis *x* is acute), then the value of x converges to the minimum value. Similarly, if the value of the tangent at a point is negative (the angle of the line with the axis is obtuse), then the value of x converges to the minimum value.



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

GRADIENT DESCENT ALGORITHM

- **(Alpha term)** α
- What happens if alpha is too small or too large
- Too small

  Take tiny steps

  Takes too long
- Too large

  Can overshoot the minimum and fail to converge

GRADIENT DESCENT ALGORITHM

- Linear regression with gradient descent
- Apply gradient descent to minimize the squared error cost function $J(\theta_0 , \theta_1)$
- Now we have a partial derivative
- $\frac{\partial J(\theta)}{\partial \theta} J(\theta_0 , \theta_1) = \frac{\partial J(\theta)}{\partial \theta} (\frac{1}{2m} \sum_i (h_\theta (x_i) - y_i)^2)$
- $= \frac{\partial J(\theta)}{\partial \theta} (\frac{1}{2m} \sum_i ((\theta_0 + \theta_1 x_i) - y_i)^2)$
- So we need to determine the derivative for each parameter   i.e.
- When $j = 0$
- When $j = 1$
- When we derive this expression respect to $j = 0$ and $j = 1$ we get the following:

GRADIENT DESCENT ALGORITHM

- *j=0 :* $\frac{\partial J(\theta)}{\partial \theta} J(\theta_0 , \theta_1) = \frac{1}{2m} \sum_i 2((\theta_0 + \theta_1 x_i) - y_i)$
- $= \frac{1}{m} \sum_i ((\theta_0 + \theta_1 x_i) - y_i)$
- *j=1 :* $\frac{\partial J(\theta)}{\partial \theta} J(\theta_0 , \theta_1) = \frac{1}{2m} \sum_i 2((\theta_0 + \theta_1 x_i) - y_i)\, x_i$
- $= \frac{1}{m} \sum_i ((\theta_0 + \theta_1 x_i) - y_i)\, x_i$

MULTIPLE FEATURES

- **Larger number of features (dimensions)**
- e.g. with houses
    - Area
    - Age
    - Number bedrooms
    - Number floors
- x1, x2, x3, x4
- With multiple features becomes hard to plot
- Can't really plot in more than 3 dimensions
- Notation becomes more complicated too
- Best way to get around with this is the notation of linear algebra
- Gives notation and set of things you can do with matrices and vectors

MULTIPLE FEATURES

- $X = \begin{bmatrix} 150 & 2 & 2 & 12 \\ 100 & 2 & 1 & 3 \\ 75 & 1 & 2 & 4 \\ 200 & 3 & 3 & 8 \end{bmatrix}$

- $Y = \begin{bmatrix} 15000 \\ 11000 \\ 9000 \\ 19000 \end{bmatrix}$

Area	Bedroom	Floor	Age	Price y' s
150	2	2	12	15,000
100	2	1	3	11,000
75	1	2	4	9,000
200	3	3	8	19,000

MULTIPLE FEATURES

- This matrix shows us
    - Area
    - Number of bedrooms
    - Number floors
    - Age of home
- All in one variable
- Block of numbers, take all data organized into one big block (Vector Shown as y)
- Y shows us the prices

MULTIPLE FEATURES

- Here we introduce some of the terms:
- **Dataset**: is a set of samples that are used to train the machine, and it contains input and output data. The purpose is obtaining the hypothesis function.
- ***m***: The number of samples in the data set. (4 houses)
- ***n***: The number of features .(4 feature)
- **Feature Vector**: the array of single sample’s values in all features.
- sample$_3$ FV= (x_3,y_3) =*(75,1,2,4,9000)*
- x_3=*(75,1,2,4)*
- y_3=*(9000)*

MULTIPLE FEATURES

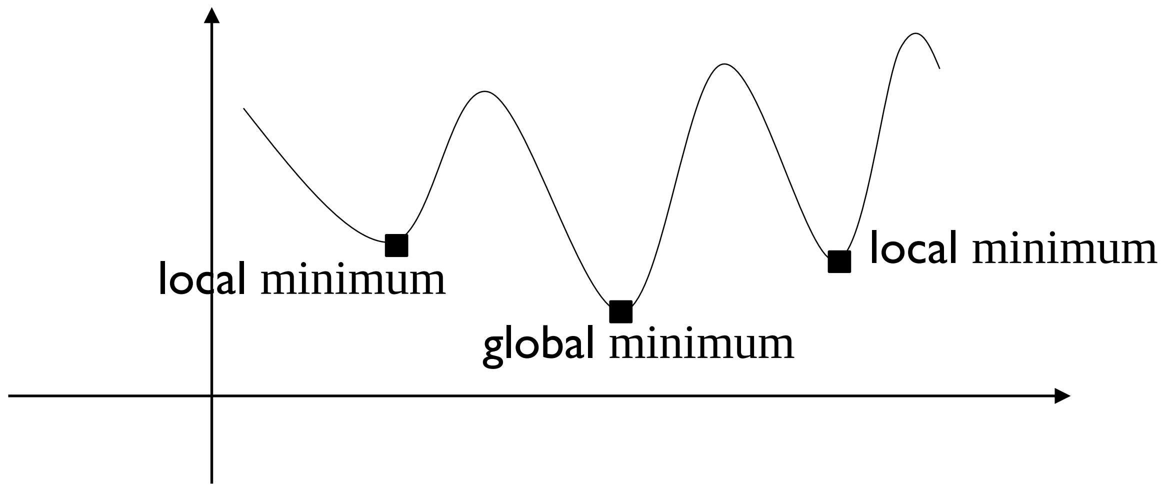
- For one feature the hypothesis is:
- $h_\theta(x) = \theta_0 + \theta_1 x$
- but for multiple features the hypothesis function is:
- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots\ldots + \theta_n x_n$
- So the cost function will be:
- $J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_i (h_{\boldsymbol{\theta}}(x_i) - y_i)^2 = J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_i ((\theta_0 + \theta_1 x_1 + \theta_1 x_2 + \ldots + \theta_n x_n) - y_i)^2$

the derivative is:

- $\frac{\partial J(\theta_j)}{\partial \theta_j} J(\theta_j) = \frac{1}{m} \sum_i ((\theta_0 + \theta_1 x_1 + \theta_1 x_2 + \ldots + \theta_n x_n) - y_i)\, x_i$

MULTIPLE FEATURES

- To solve this we need to know multivariate calculus
- So we can plug these values back into the gradient descent algorithm
- Risk of meeting different local optimum
- The univariate linear regression cost function is always a convex function - always has a single minimum – (One global optimum)
- So gradient descent will always converge to global optimum


local minimum
global minimum
local minimum