COMP 5630/6630:Machine Learning

Lecture 3: Linear Regression

Slide Adapted: Drs. Aakur, Farhana, Andrew Ng, Moses Charikar, Byron Boots and Chris Ré

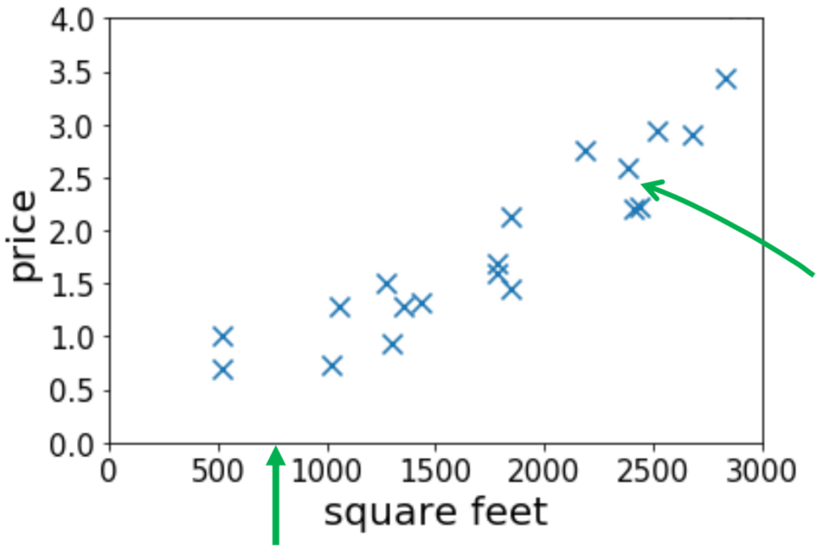
- ML
  - Learn a "mapping" from input to output h: X → Y
    - Fitting a function, f, from input to output using the *dataset*
- Supervised Learning
  - Training Data
    - Used to learn a "mapping" from input to output, f: X → Y
  - Hypothesis, *h*
    - Learned "mapping", h: X → Y using the training data
  - Test data
    - Use the hypothesis, *h* to predict new, unseen data

House Price Prediction

- Given: a dataset that contains n samples

$$\left(x^{(1)}, y^{(1)}\right), \ldots \left(x^{(n)}, y^{(n)}\right)$$

- Task: if a residence has x square feet, predict its price?



15th sample

$\left(x^{(15)}, y^{(15)}\right)$

$x = 800$

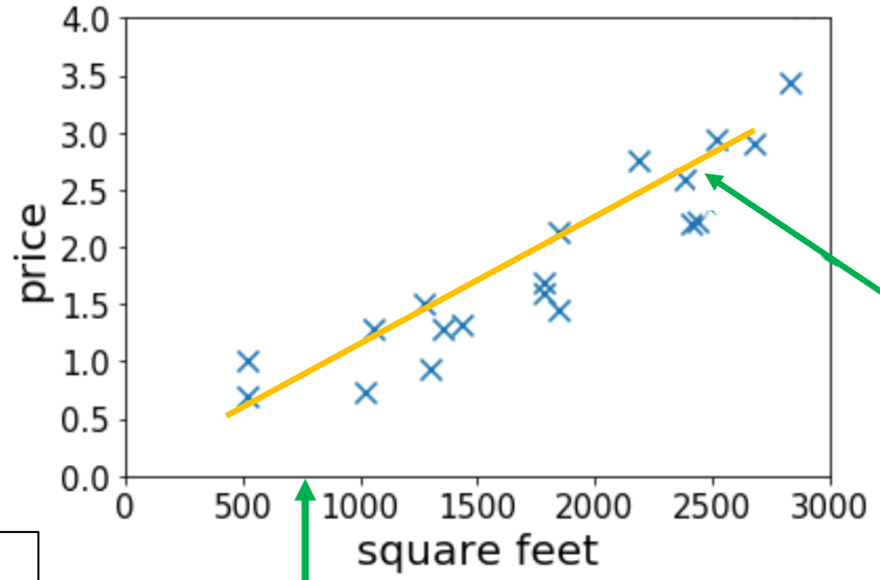
$y = ?$

House Price Prediction

- Given: a dataset that contains n samples

$$\left(x^{(1)}, y^{(1)}\right), \dots \left(x^{(n)}, y^{(n)}\right)$$

- Task: if a residence has x square feet, predict its price?



- Fit a line
- Estimate the price for x = 800 on fitted line

$$x = 800$$
$$y = ?$$

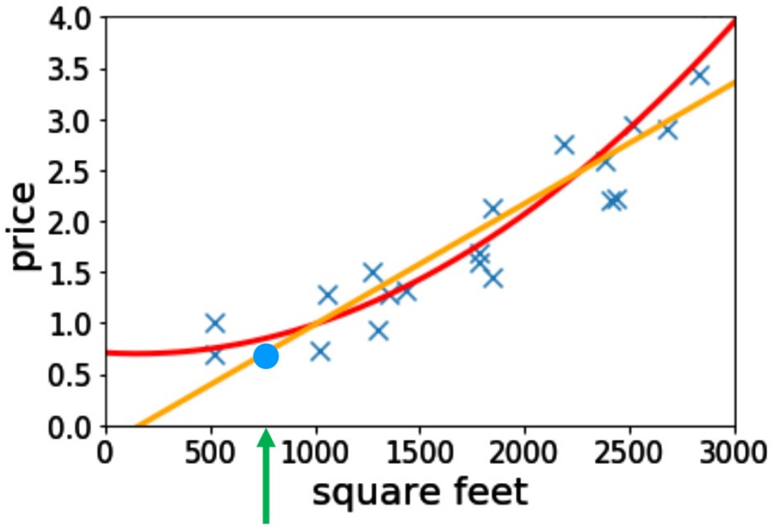
Is this the best way to learn a mapping from x → y?

House Price Prediction

- Given: a dataset that contains n samples

$$\left(x^{(1)}, y^{(1)}\right), \dots \left(x^{(n)}, y^{(n)}\right)$$

- Task: if a residence has x square feet, predict its price?



$x = 800$
$y = ?$

- Linear vs quadratic fit

How to choose between different mapping?

House Price Prediction

- Supervised Learning
  - Given a *training dataset*, learn a mapping *(hypothesis, h)* from x→ y, where y is labelled
  - Goal: Given a new datapoint, x (*test data*), predict the most accurate output, y, using the *learned hypothesis, h*
    - *learned mapping = trained model*

Predicting House Price: Learn a Mapping from x →y

Dataset of the living areas, bedrooms, and prices of 47 houses

Living area (ft^2)	# bedrooms	Price (1000$s)
1643	4	256
1356	3	202
1678	3	287
…	…	…
3000	4	400

Predicting House Price: Learn a Mapping from $X \rightarrow y$

Dataset of the living areas, bedrooms, and prices of 47 houses

Living area (ft^2)	# bedrooms	Price (1000$s)
1643	4	256
1356	3	202
1678	3	287
…	…	…
3000	4	400

X **y**

Recap: Linear Classifier

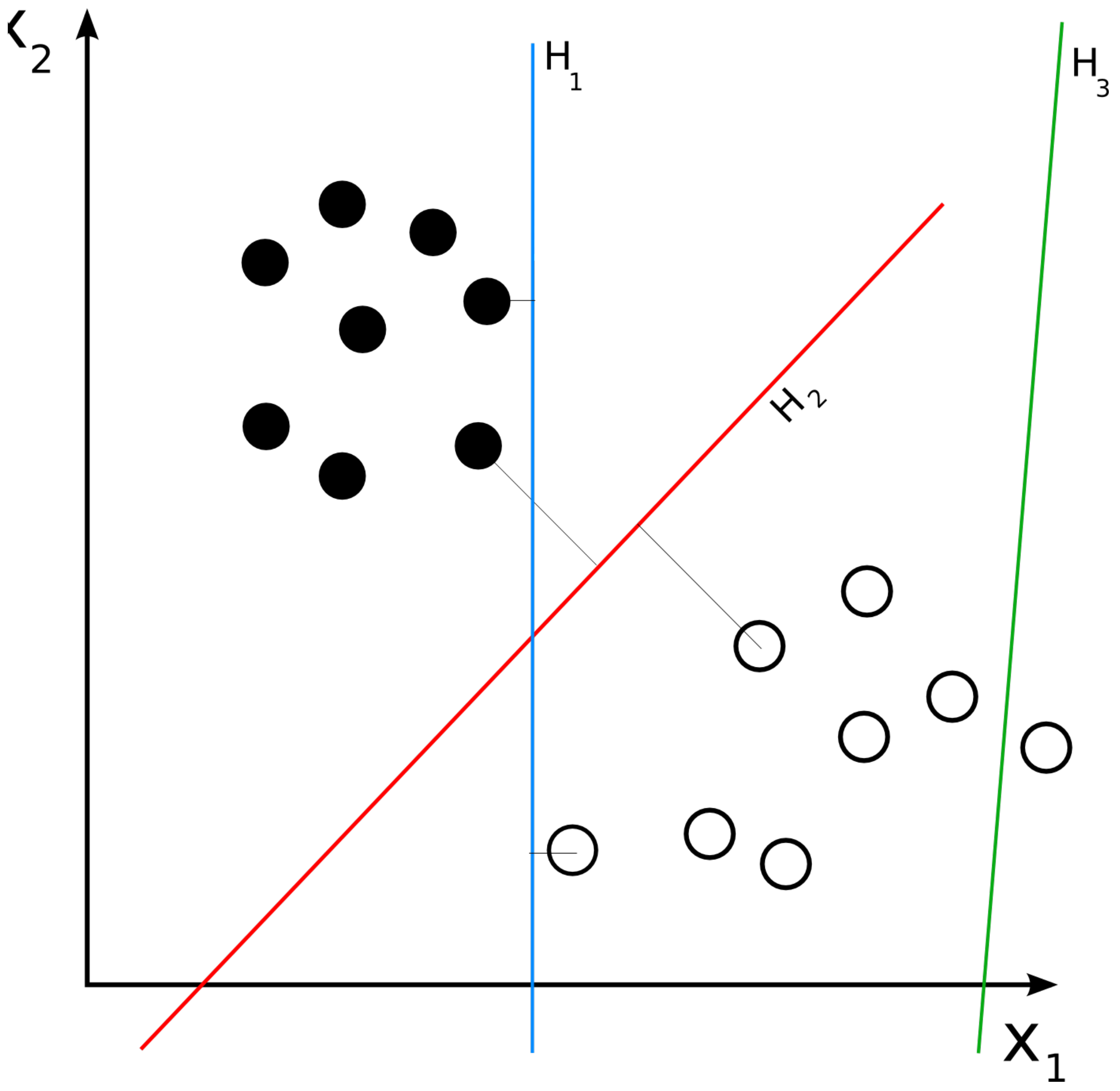
- Black and white circles are different labels. H_1, H_2, ... represent different *decision boundaries* i.e. linear functions that best map the classification process.
    - ***Goal:*** find the best linear function that has highest accuracy

Idea

Repeat

1. Draw a line to separate two classes
2. Calculate accuracy
3. Stop if accuracy can't be improved

Linear function to fit the data is a line y = mx+c



Recap: Linear Classifier Example

stretch pixels into single column

input image

56	231
24	2

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} \longrightarrow \begin{array}{c} \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \end{array} \begin{array}{l} \text{cat score} \\ \text{dog score} \\ \text{ship score} \end{array}$$

$$\boxed{f(x_i; W, b) = W x_i + b}$$

Linear function has the form $f = W x_i + b$

Predicting House Price: Learn a Mapping from X →y

Dataset of the living areas, bedrooms, and prices of 47 houses

Living area (ft²)	# bedrooms	Price (1000$s)
1643	4	256
1356	3	202
1678	3	287
…	…	…
3000	4	400

$\underbrace{\text{Living area, \# bedrooms}}_{\mathbf{X}}$ $\underbrace{\text{Price}}_{\mathbf{y}}$

- Predict y from one feature, living area

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

x_1 = Living area
x_2 = # bedrooms

- Predict y from two features, living area and # bedrooms

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- θ_i = the ***parameters*** or ***weights*** of the linear model characterizing X→Y

- A more generic model is $h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$

Linear Regression (cntd.)

- **Learning:** Given this formulation, we will need to identify a way to find the values of θ.

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

- We will need to use the ***training*** data to learn these parameters. This process is called ***learning***

- **What do we need to achieve this?**

Linear Regression (cntd.)

- **Learning:** Given this formulation, we will need to identify a way to find the values of θ.

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

- We will need to use the ***training*** data to learn these parameters. This process is called ***learning***

- **What do we need to achieve this?**

- We will define a function that measures the quality of predictions for each value of θ.
- This is called the **cost function or objective function**

How to Define the Cost or Objective Function

- Idea: Minimize the ***squared*** difference between the hypothesis, $h_\Theta(x)$ and y

1. $h_\Theta(x) - y$ — difference between the hypothesis, $h_\Theta(x)$ and y
2. $(h_\Theta(x) - y)^2$ — squared difference between $h_\Theta(x)$ and y
3. $\min_\Theta (h_\Theta(x) - y)^2$ — choose Θ values to minimize the squared difference between $h_\Theta(x)$ and y
4. $\min_\Theta \sum_{i=1}^{m} (h_\Theta(x^{(i)}) - y^{(i)})^2$ — take the summation of squared difference of Step 3 for all training examples, m
5. $\min_\Theta \frac{1}{2}\sum_{i=1}^{m} (h_\Theta(x^{(i)}) - y^{(i)})^2$ — Multiply by a constant, ½, for convention

$J(\Theta) = \frac{1}{2}\sum_{i=1}^{m} (h_\Theta(x^{(i)}) - y^{(i)})^2$ Objective function to minimize ***Ordinary least squares regression model***

Learning: Gradient Descent

- Goal: To find a set of parameters θ that will minimize the cost function J(θ).
- Common approach: *gradient descent*
- What does the *gradient descent* do?
    - Start with an initial “guess” for θ
    - Update values of θ that will gradually move towards the “optimal solution”
    - What is the optimal solution?
    - The value of θ that minimizes the cost function

- How do we do it computationally?

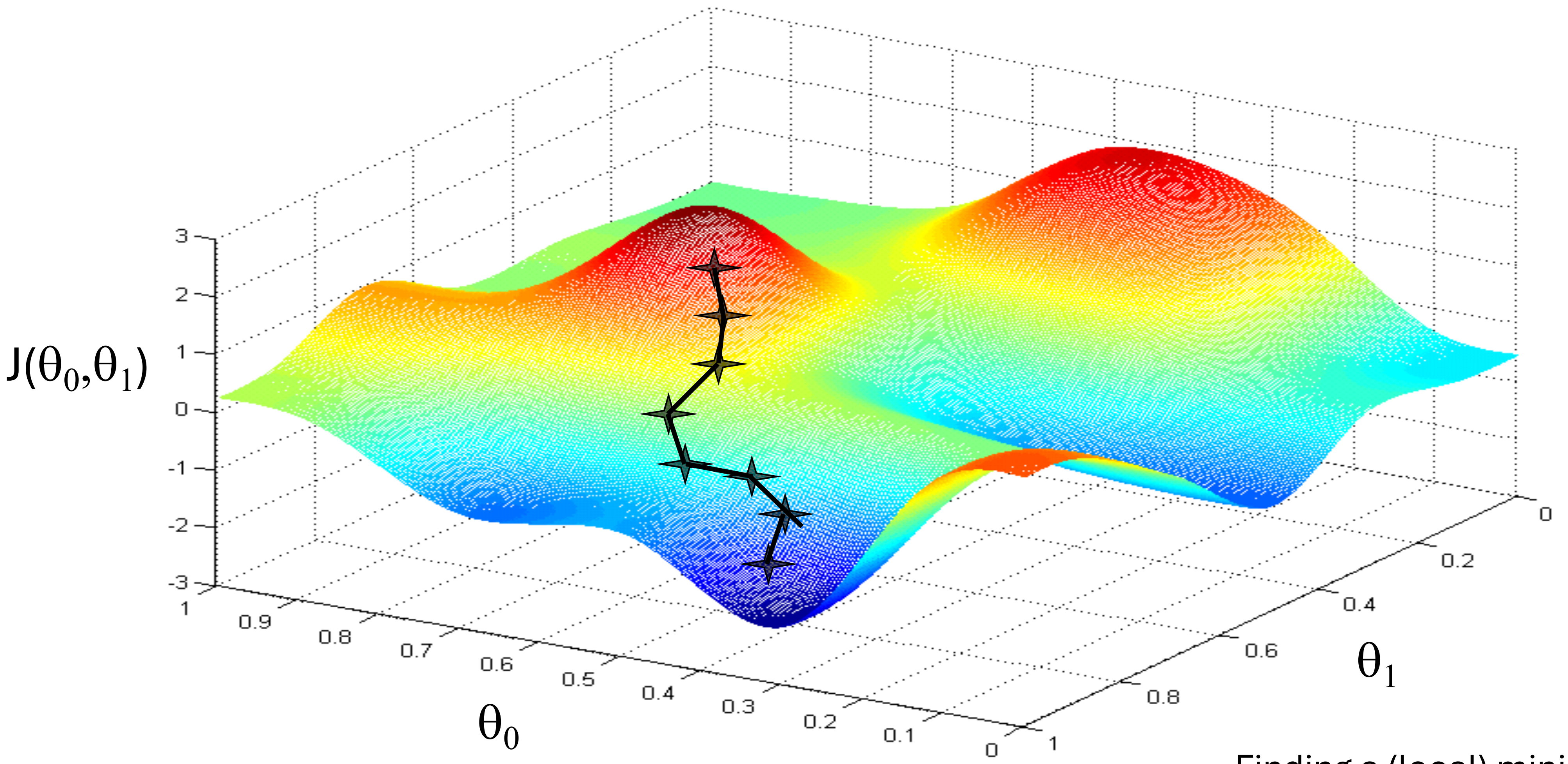
Gradient Descent

- How do we do it computationally?

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

- α is the learning rate
  - Modulates how much of the change that we need to propagate at each instant
- Each update of θ will be a step in the *steepest decrease of the cost function J(*θ)
- How to compute the derivative of the cost function J(θ)?

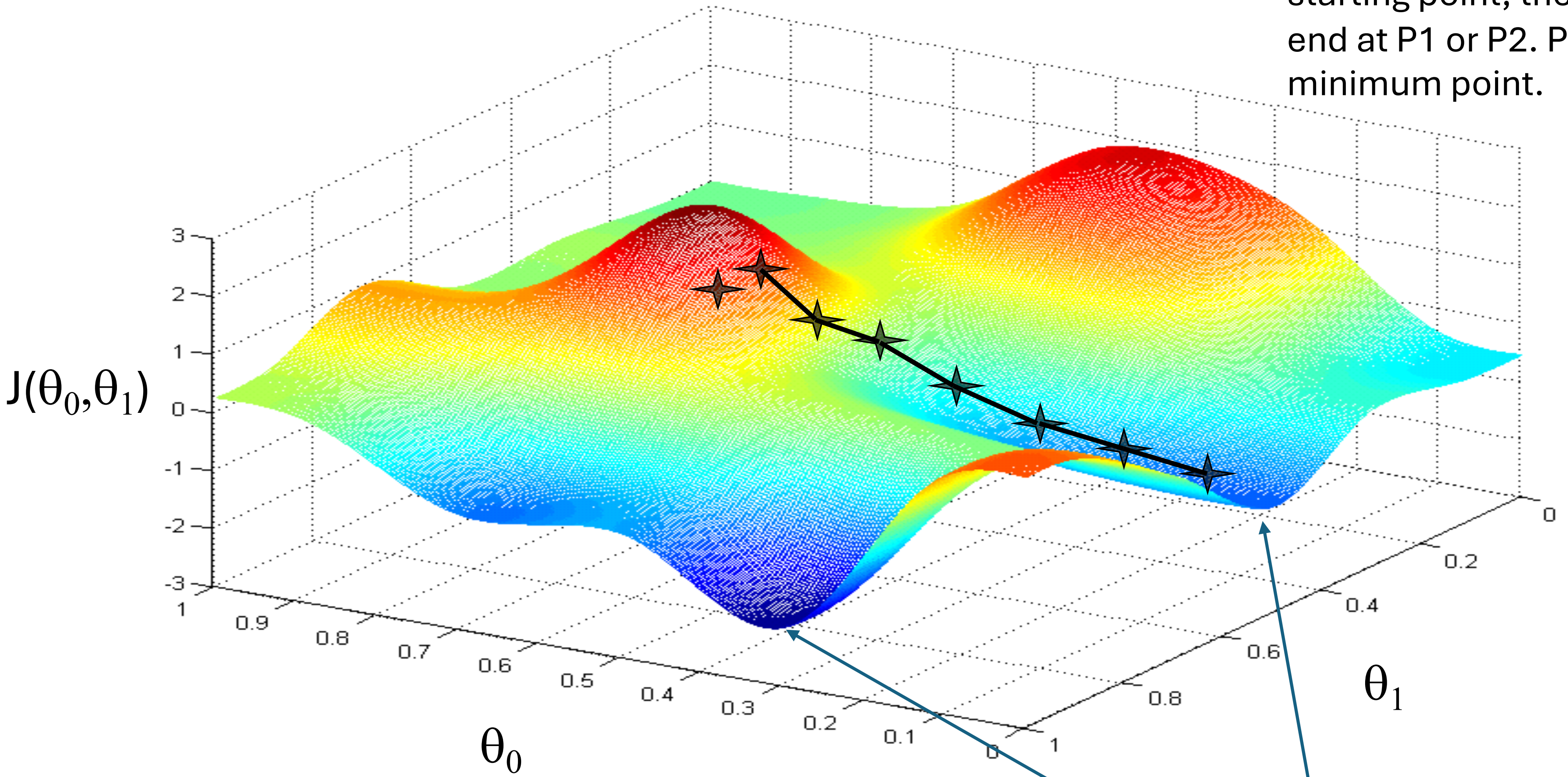
Gradient Descent Visualization Example



Finding a (local) minimum depend on the starting point and step size, α, to change direction

Gradient Descent Visualization

In this example, depending on the starting point, the algorithm can end at P1 or P2. P2 is global minimum point.



P1 =(local) minimum

P2 = (local) minimum

Gradient Descent Computation

- $\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\Theta(x) - y)^2$

$J(\Theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\Theta(x^{(i)}) - y^{(i)})^2$ ***Assume only 1 datapoint ignoring summation***

$= 2.\frac{1}{2} (h_\Theta(x) - y) \quad \frac{\partial}{\partial \theta_j} (h_\Theta(x) - y)$

Chain rule of derivative

$= (h_\Theta(x) - y) . \frac{\partial}{\partial \theta_j} (\Theta_0 x_0 + \Theta_1 x_1 + \Theta_2 x_2 + \ldots. + \Theta_n x_n - y)$

$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$

$= (h_\Theta(x) - y) . x_j$

$\frac{\partial}{\partial \theta_j} (\Theta_j x_j) = x_j,$ other terms are 0

e.g., $\frac{\partial}{\partial \theta_0} (\Theta_0 x_0) = x_0,$ other terms are 0

Gradient Descent

- Hence each update is given by

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

- This is called the **LMS** update rule or the ***Least Mean Squares*** update rule.
  - Also known as the ***Widrow-Hoff*** learning rule.

Gradient Descent

- Has several properties:
  - Magnitude of update is proportional to the error (y – h(x))
    - What does this mean?
    - If we have a very good prediction i.e. h(x)$\approx y$, then the update is very small.
    - Conversely, if the prediction is very far off i.e. h(x)$\gg y$ or h(x)$\ll y$ then the update will be large.
- For learning over the complete training set, we iteratively update the parameters as below

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$

}

Gradient Descent Update: Batch vs Stochastic

- If we use all the examples in the training set ***at once***:
  - Batch gradient descent
- What if we update at every single data point?
  - Stochastic or incremental gradient descent

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}$$

}

}

Summary

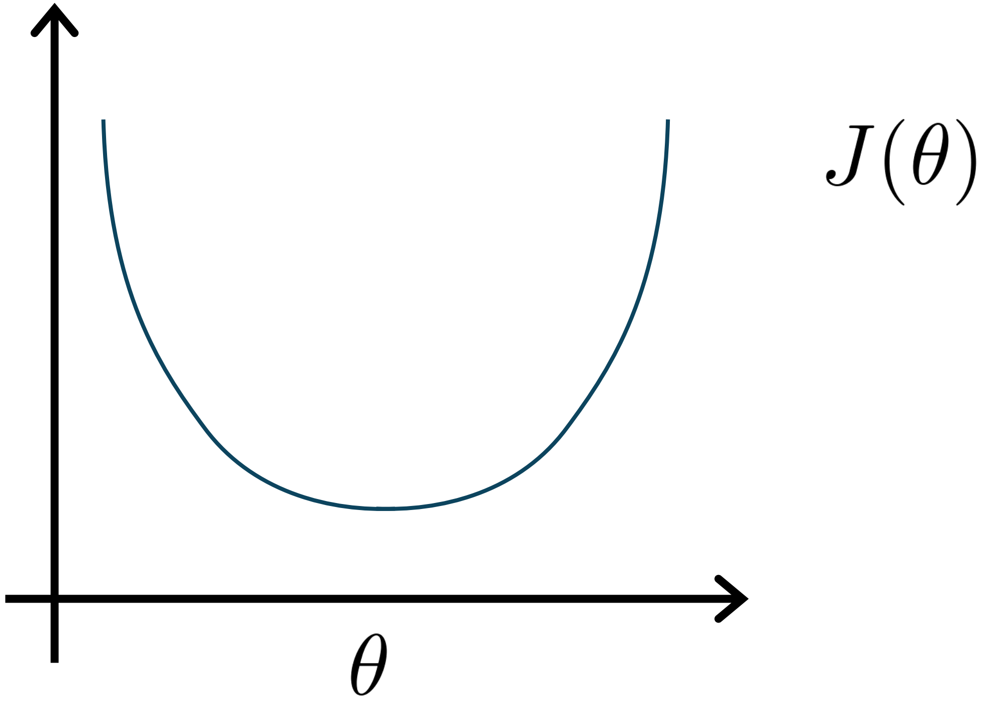
- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose, try α

$$\ldots, 0.001, \qquad , 0.01, \qquad , 0.1, \qquad , 1, \ldots$$

Normal Equations

Gradient Descent



Normal equation: Method to solve for θ analytically.

Normal Equations

- Steps:
  - Set the partial derivatives of the cost function J(θ) to zero
  - Then we can estimate the parameters as follows:

$$\nabla_\theta J(\theta) \;=\; X^T X\theta = X^T\vec{y}$$

$$\Theta = (X^T X)^{-1} X^T y$$

  - Where X is the input feature vector
  - y is the expected target value

Normal Equation Example

x_0	x_1 = Living area (ft^2)	x_2 =#bedrooms	y= Price(1000$s)
1	1643	4	256
1	1356	3	202
1	1678	3	287

- **X=** $\begin{bmatrix} 1 & 1643 & 4 \\ 1 & 1356 & 3 \\ 1 & 1678 & 3 \end{bmatrix}$

$$\boldsymbol{y} = \begin{bmatrix} 256 \\ 202 \\ 287 \end{bmatrix}$$

Dimension of X, mxn = m examples, n features
Dimension of y = mx1

$$\theta = (X^T X)^{-1} X^T y$$

m training examples, n features.

Gradient Descent

- Need to choose
- Needs many iterations α
- Works well even when n is large

Normal Equation

- No need to choose
- Don't need to iterate α
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large.

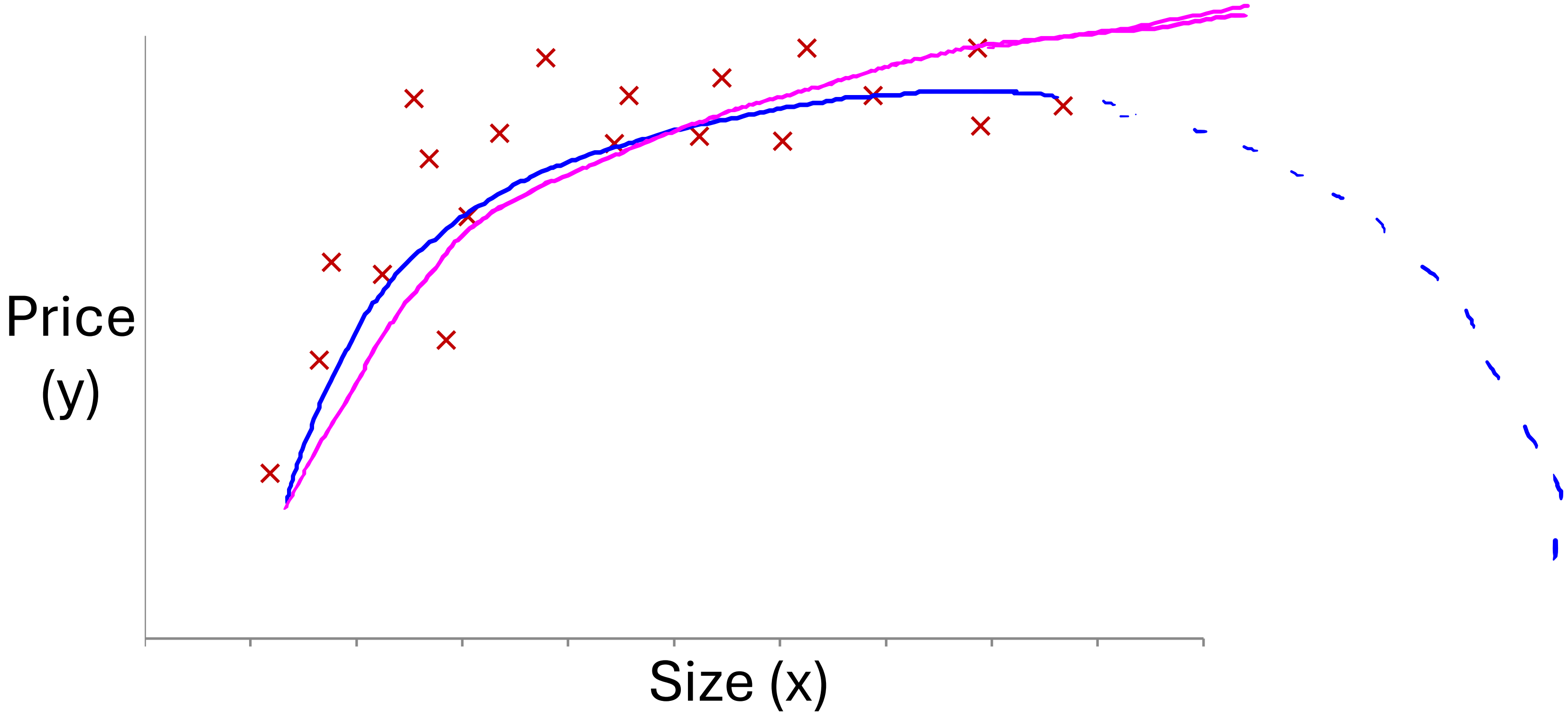
Basis Function: Extending Linear Regression to More Complex Models

- The inputs X for linear regression :
  - Original quantitative inputs

  - Transformation of quantitative inputs
    - e.g. log, exp, square root, square, etc.

  - Polynomial transformation
    - example: $y = b_0 + b_1 \times x_1 + b_2 \times x_2 + b_3 \times x_3$

Choice of Features



Blue curve fitting $$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

Purple curve fitting $$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$

Basis Function: Extending Linear Regression to More Complex Models

- Previous

- θ_i = the ***parameters*** or ***weights*** of the linear model characterizing X→Y

- A more generic model is φ

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

- Non-linear complex models

  - $h(x) = \sum_{i=0}^{n} \theta_i \underbrace{\varphi_i(x)}_{\text{Basis function}}$

Types of Basis Function

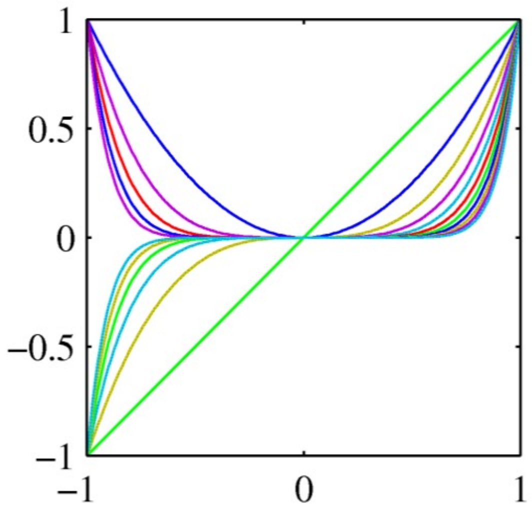
- Linear basis function: the simplest case

$$\varphi_i(\mathbf{x}) = x_i$$

- Polynomial basis function

$$\varphi_i(\mathbf{x}) = x^i$$

  - A small change in x affects all basis functions

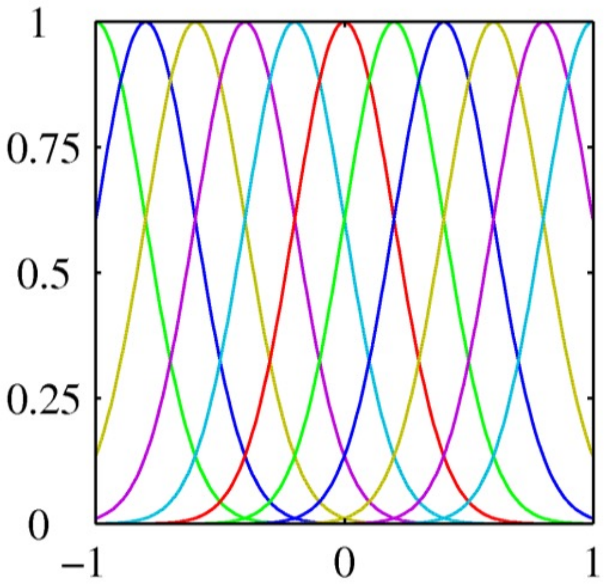


Types of Basis Function

- Gaussian basis functions

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

- A small change in x affects nearby basis functions. μ_j and s controls location and slope



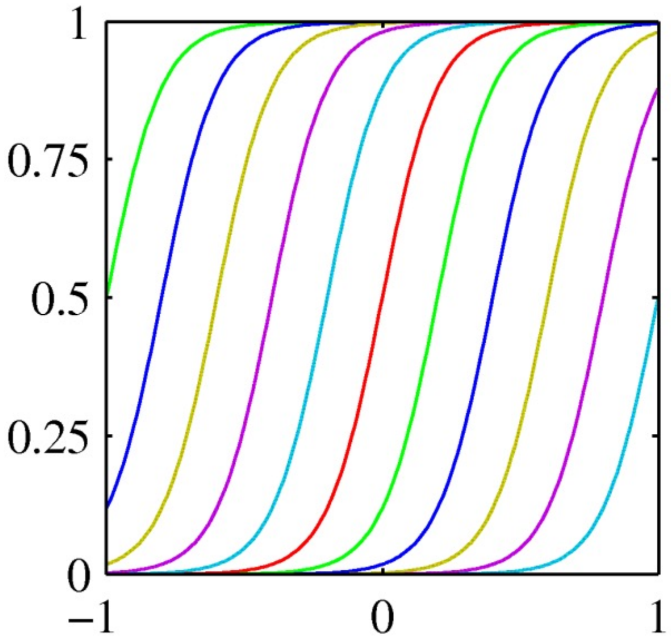
Types of Basis Function

- Sigmoidal basis functions

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

- A small change in x affects nearby basis functions. μ_j and s controls location and slope



Optional Reading

- Handout uploaded on Canvas