HALMSTAD UNIVERSITY

Learning Systems (DT8008)

Regression

Dr. Mohamed-Rafik Bouguelia
mohbou@hh.se

Halmstad University

HALMSTAD UNIVERSITY

(I) Linear Regression

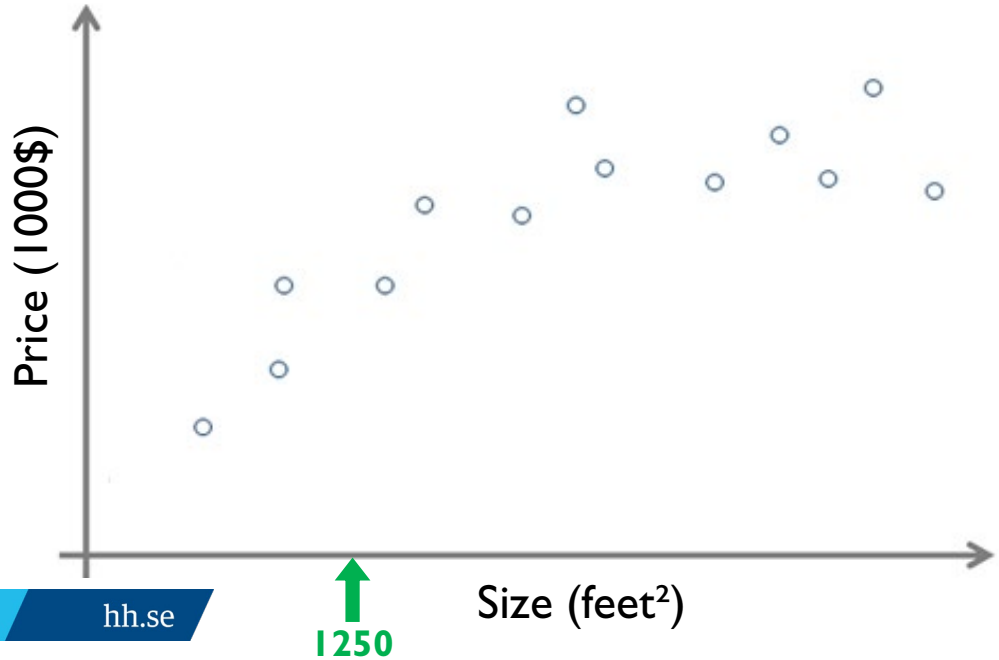
Example of Linear Regression with One Feature

Linear Regression with one feature

- Suppose that we are given the following dataset:

House size in feet² (x)	House price in 1000$ (y)
$x^{(1)} = 2104$	$y^{(1)} = 460$
$x^{(2)} = 1416$	$y^{(2)} = 230$
$x^{(3)} = 1534$	$y^{(3)} = 315$
$x^{(4)} = 852$	$y^{(4)} = 178$
...	...

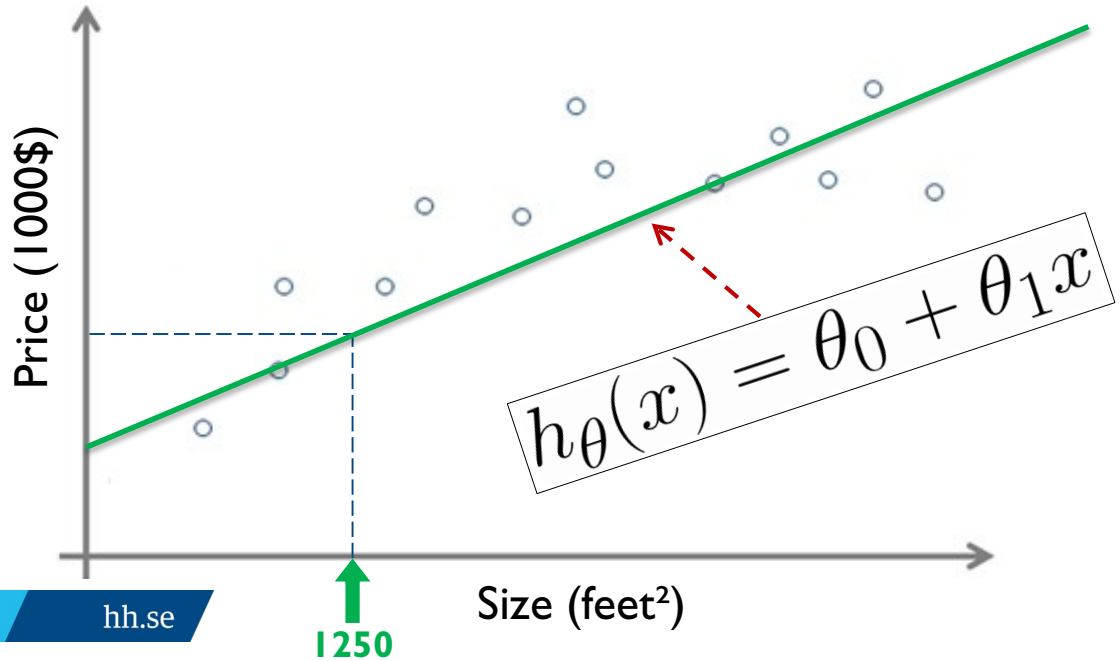
Question:
Given a new house with a size of **1250** feet², how do we predict its price ?



Linear Regression with one feature

- Suppose that we are given the following dataset:

House size in feet² (x)	House price in 1000$ (y)
$x^{(1)} = 2104$	$y^{(1)} = 460$
$x^{(2)} = 1416$	$y^{(2)} = 230$
$x^{(3)} = 1534$	$y^{(3)} = 315$
$x^{(4)} = 852$	$y^{(4)} = 178$
...	...



Question:
Given a new house with a size of **1250** feet², how do we predict its price ?

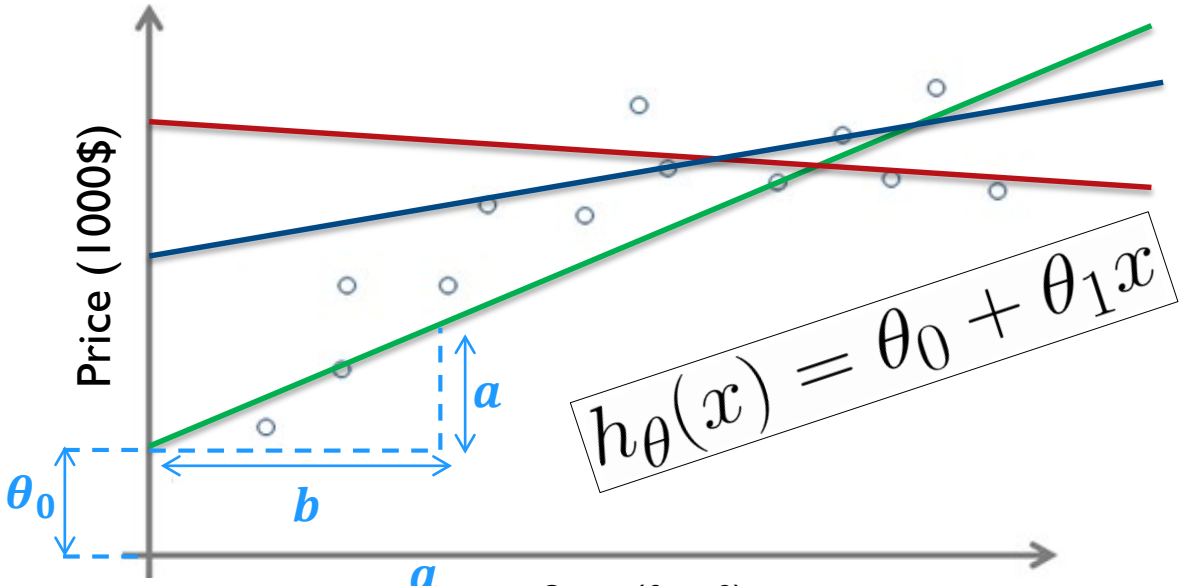
1. Assume that the relation between **size** and **price** is linear.
2. Find a line that fits the training dataset well.

Linear Regression with one feature

- Suppose that we are given the following dataset:

House size in feet² (x)	House price in 1000$ (y)
$x^{(1)} = 2104$	$y^{(1)} = 460$
$x^{(2)} = 1416$	$y^{(2)} = 230$
$x^{(3)} = 1534$	$y^{(3)} = 315$
$x^{(4)} = 852$	$y^{(4)} = 178$
...	...

Question:
Given a new house with a size of **1250** feet², how do we predict its price ?

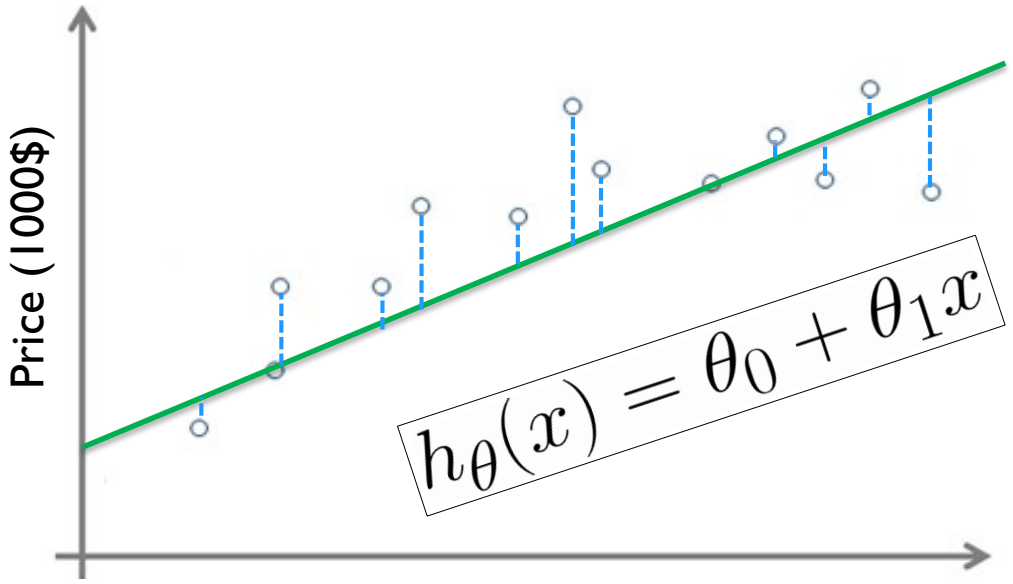


1. Assume that the relation between **size** and **price** is linear.
2. Find a line that fits the training dataset well.

? How do we find the best parameters θ_0 and θ_1 (i.e. the best fitting line)

Linear Regression with one feature

- Choose θ_0, θ_1 so that $h_\theta(x^{(i)})$ is close to $y^{(i)}$ for our training examples $(x^{(i)}, y^{(i)}), \forall i = 0 \ldots n$
- We want to find the parameters $\theta = (\theta_0, \theta_1)$ that minimizes the error function $E(\theta)$.

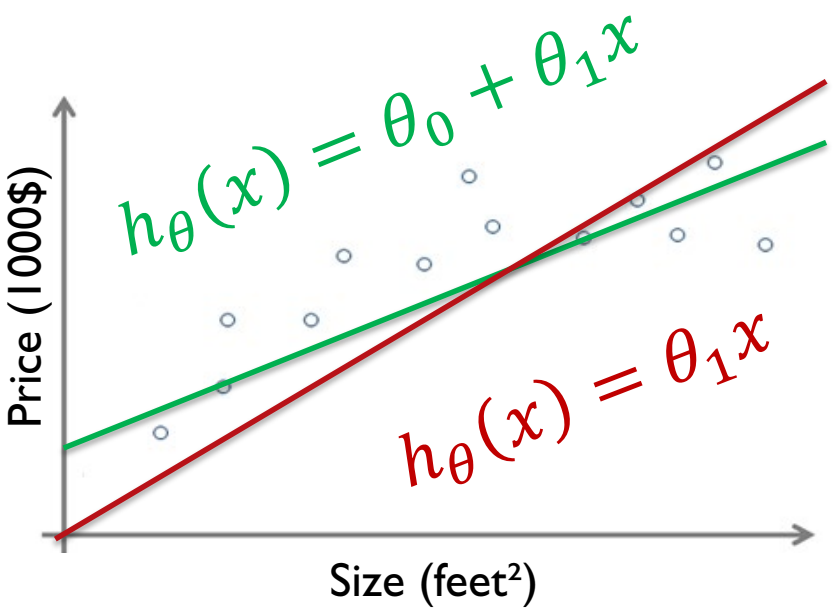


$$\min_{\theta_0,\theta_1} \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left[h_\theta(x^{(i)}) - y^{(i)}\right]^2}_{\text{Error function } E(\theta_0,\theta_1)}$$

(mean squared error cost function)

Linear Regression with one feature

- To simply the explanation, let's first assume that $\theta_0 = 0$, so our model h_θ is on the form: $h_\theta(x) = \theta_1 x$



In this case, we need to find the optimal value for θ_1

$$\underset{\theta_1}{\text{minimize}}\ \mathrm{E}(\theta_1)$$

Linear Regression with one feature

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

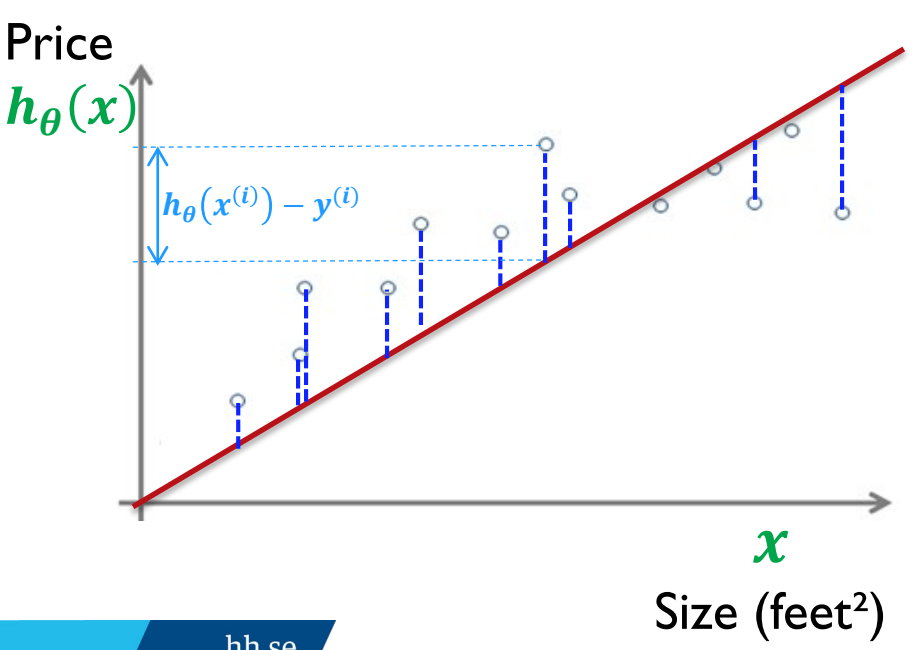
Hypothesis function (model)

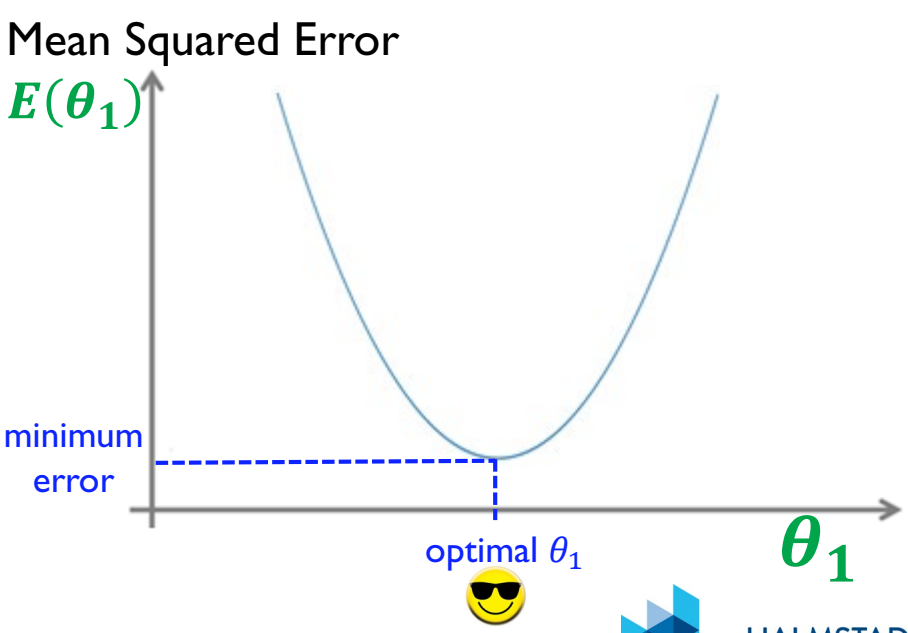
For a fixed θ_1, this is a function of the input x

$$E(\theta_1)\frac{1}{2n}\sum_{i=1}^{n}\left[h_\theta(x^{(i)}) - y^{(i)}\right]^2$$

Error (cost) function

Function of the parameter θ_1





Linear Regression with one feature

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

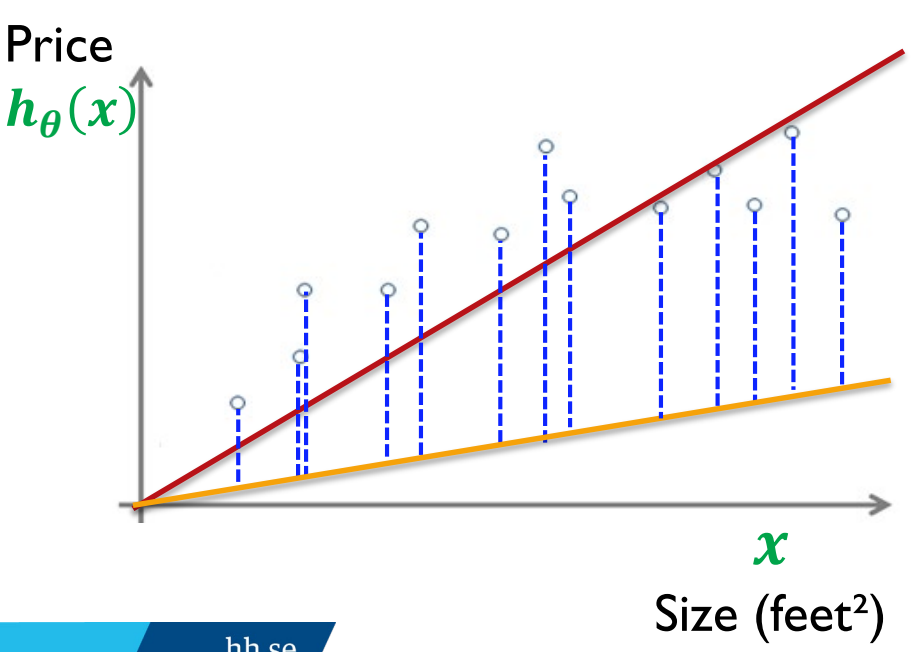
Hypothesis function (model)

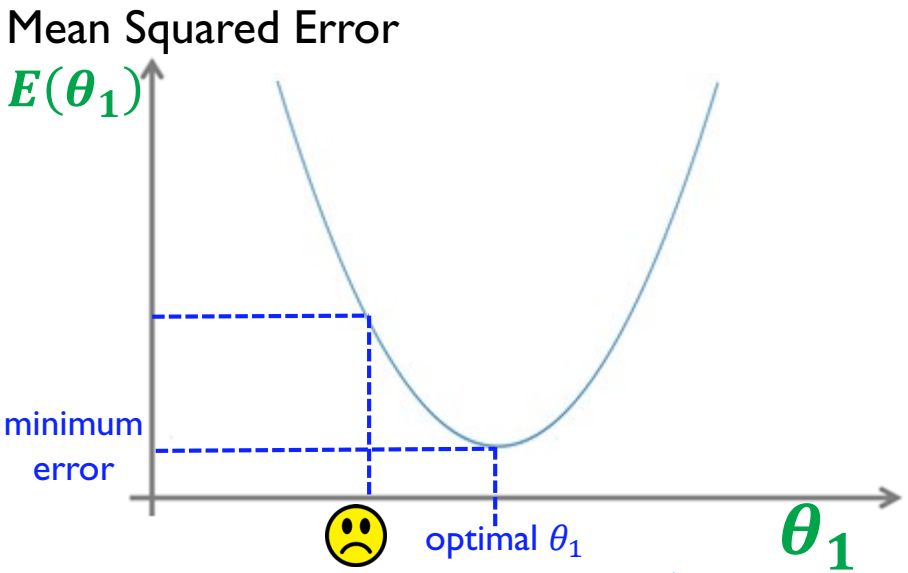
For a fixed θ_1, this is a function of the input x

$$E(\theta_1)\frac{1}{2n}\sum_{i=1}^{n}\left[h_\theta(x^{(i)}) - y^{(i)}\right]^2$$

Error (cost) function

Function of the parameter θ_1



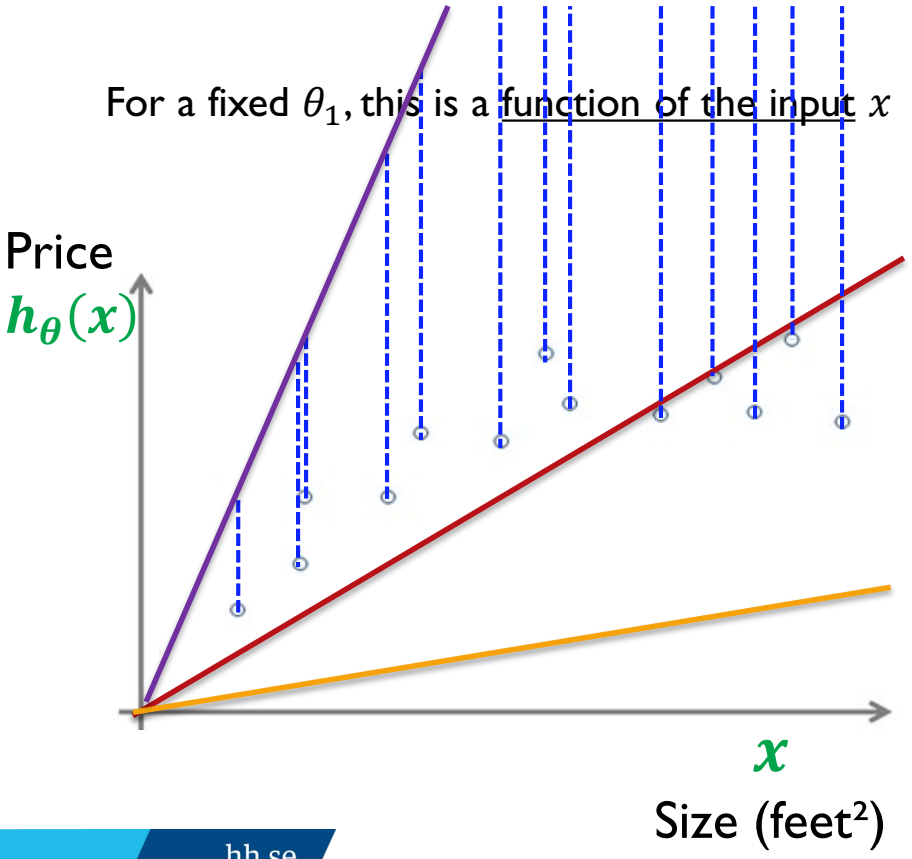


Linear Regression with one feature

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

Hypothesis function (model)

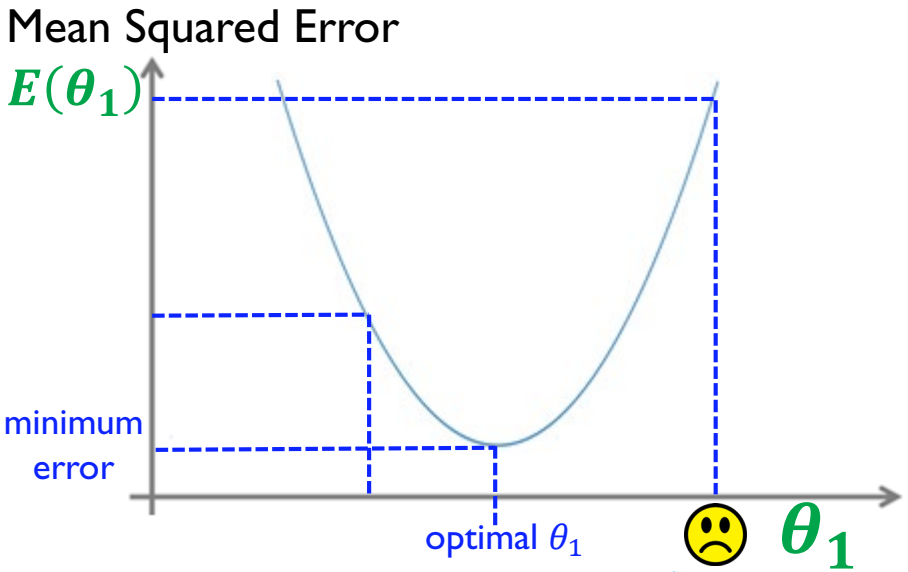
For a fixed θ_1, this is a function of the input x



$$E(\theta_1)\frac{1}{2n}\sum_{i=1}^{n}\left[h_\theta(x^{(i)}) - y^{(i)}\right]^2$$

Error (cost) function

Function of the parameter θ_1

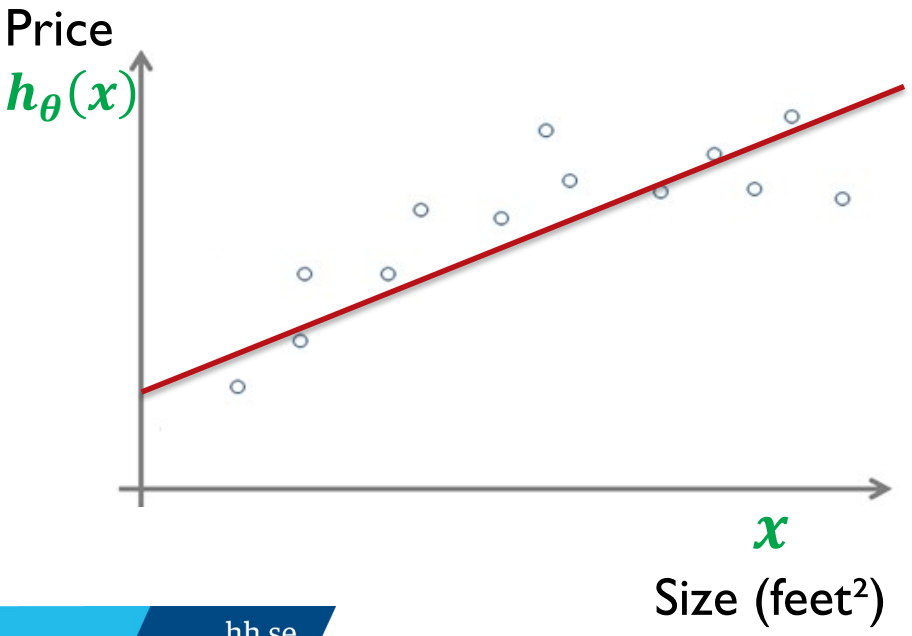


Linear Regression with one feature

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

Hypothesis function (model)

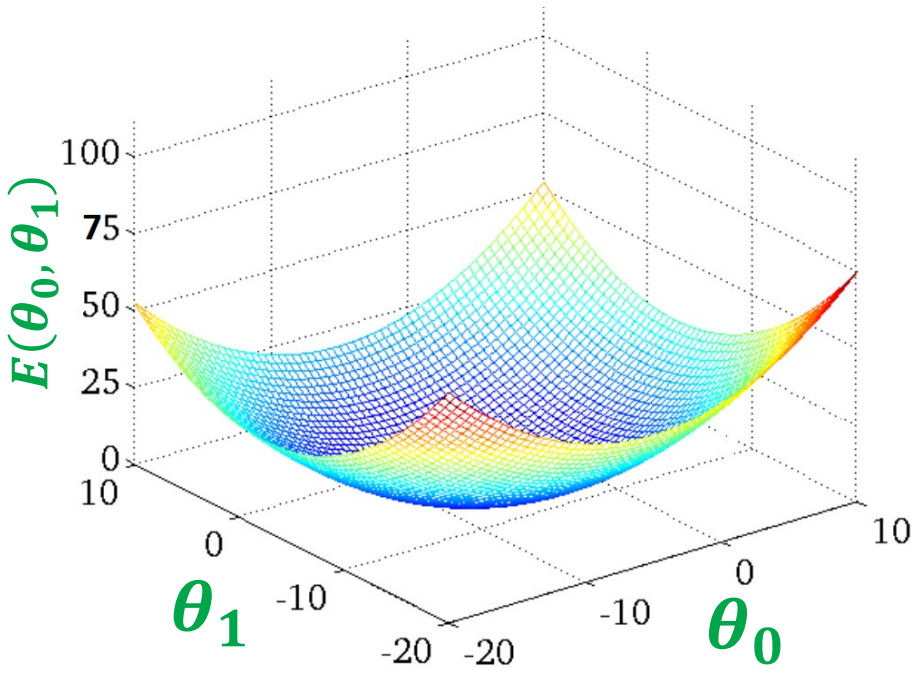
For fixed θ_0, θ_1, this is a function of the input x



$$E(\theta_0, \theta_1)\frac{1}{2n}\sum_{i=1}^{n}\left[h_\theta(x^{(i)}) - y^{(i)}\right]^2$$

Error (cost) function

Function of the parameters θ_0, θ_1



Linear Regression with one feature

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

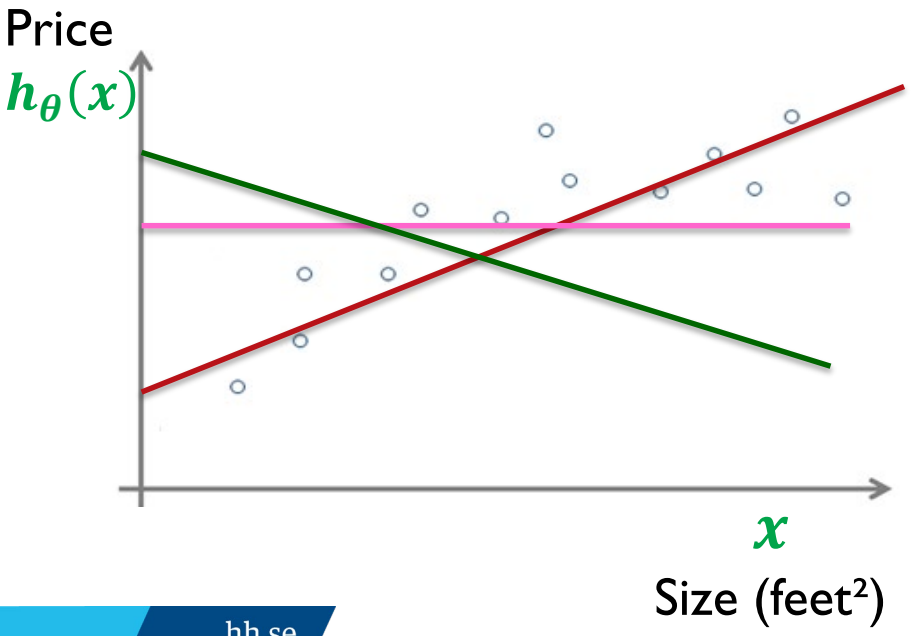
Hypothesis function (model)

For fixed θ_0, θ_1, this is a function of the input x

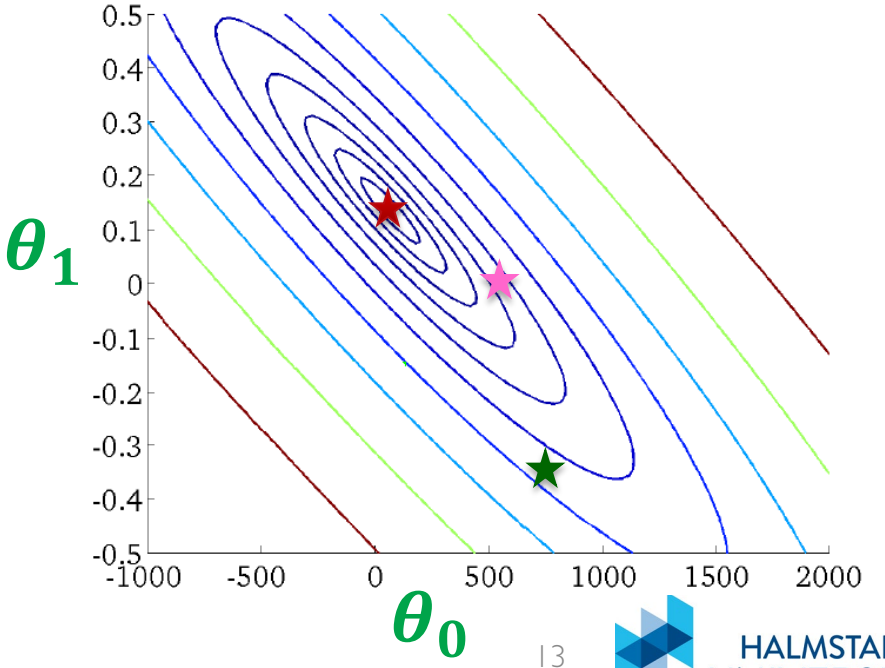
$$E(\theta_0, \theta_1) \frac{1}{2n} \sum_{i=1}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right]^2$$

Error (cost) function

Function of the parameters θ_0, θ_1



Contour plot of $E(\theta_0, \theta_1)$



Linear Regression with one feature

- To find the parameters that minimize the error function, we can use an optimization algorithm called: **Gradient Descent**.
- Gradient Descent is a general optimization algorithm, which is not only specific for this function error function.
- We will see later that for this specific error function, we can solve the minimization problem without the need to use the gradient descent algorithm.

Optimization problem: $\min_{\theta_0,\theta_1,\ldots} E(\theta_0, \theta_1, \ldots)$

Error/cost function: $E(\theta_0, \theta_1, \ldots) = \frac{1}{2n} \sum_{i=1}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right]^2$

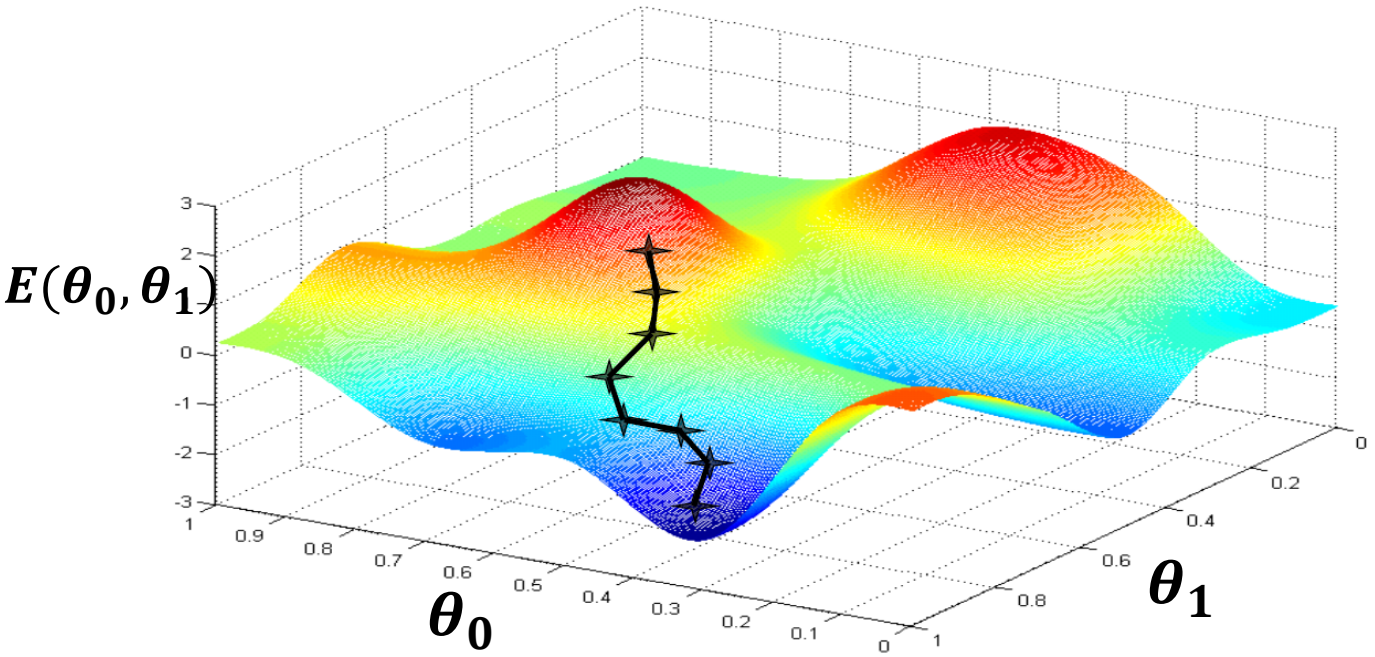
Hypothesis function: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots$

Gradient Descent

Algorithm for optimization

Gradient Descent – Basic idea

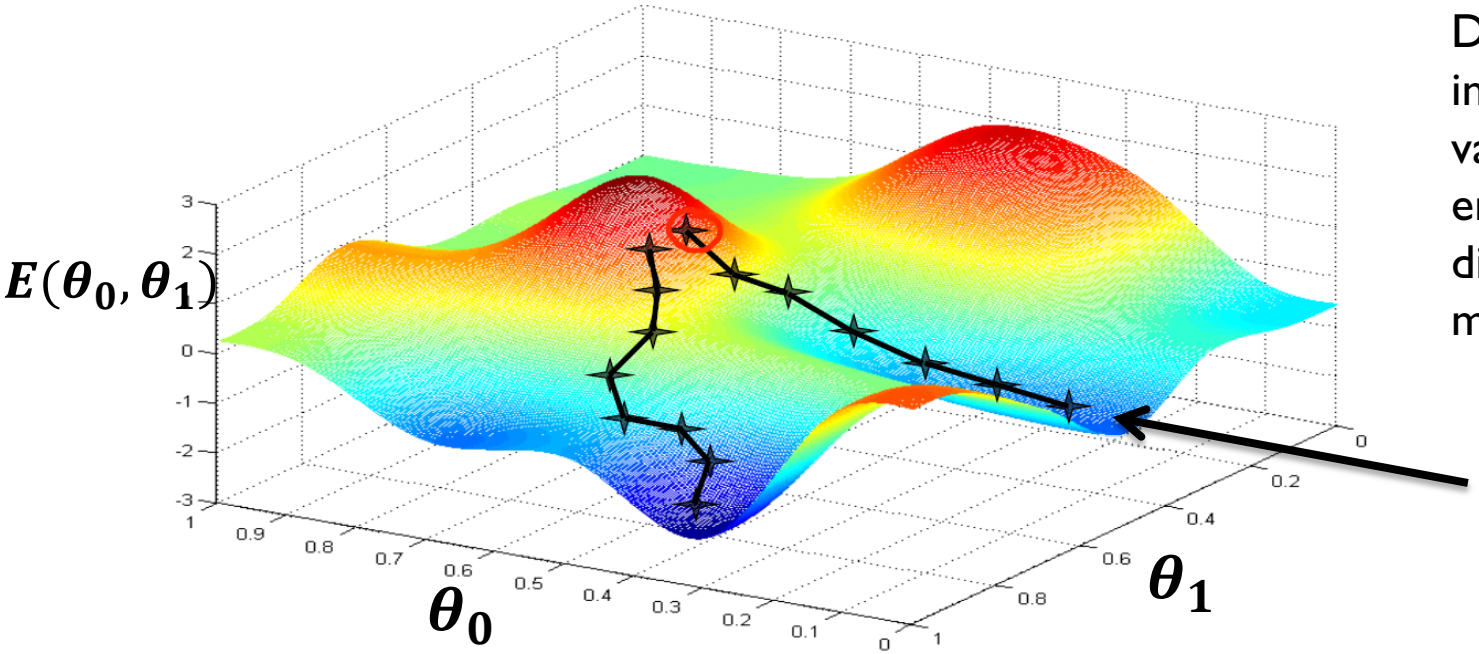
1. Start with some values for the parameters θ_0, θ_1
2. Keep updating θ_0, θ_1 to reduce $E(\theta_0, \theta_1)$ until we hopefully end up at a minimum.
   - At each update, how do we decide if we should increase or decrease each of the parameters?



Note:
For the purpose of explanation, the non-convex error function presented in this example is **not** the mean squared error (MSE). The MSE function is convex.

Gradient Descent – Basic idea

1. Start with some values for the parameters θ_0, θ_1
2. Keep updating θ_0, θ_1 to reduce $E(\theta_0, \theta_1)$ until we hopefully end up at a minimum.
   - At each update, how do we decide if we should increase or decrease each of the parameters? ➔ next slide

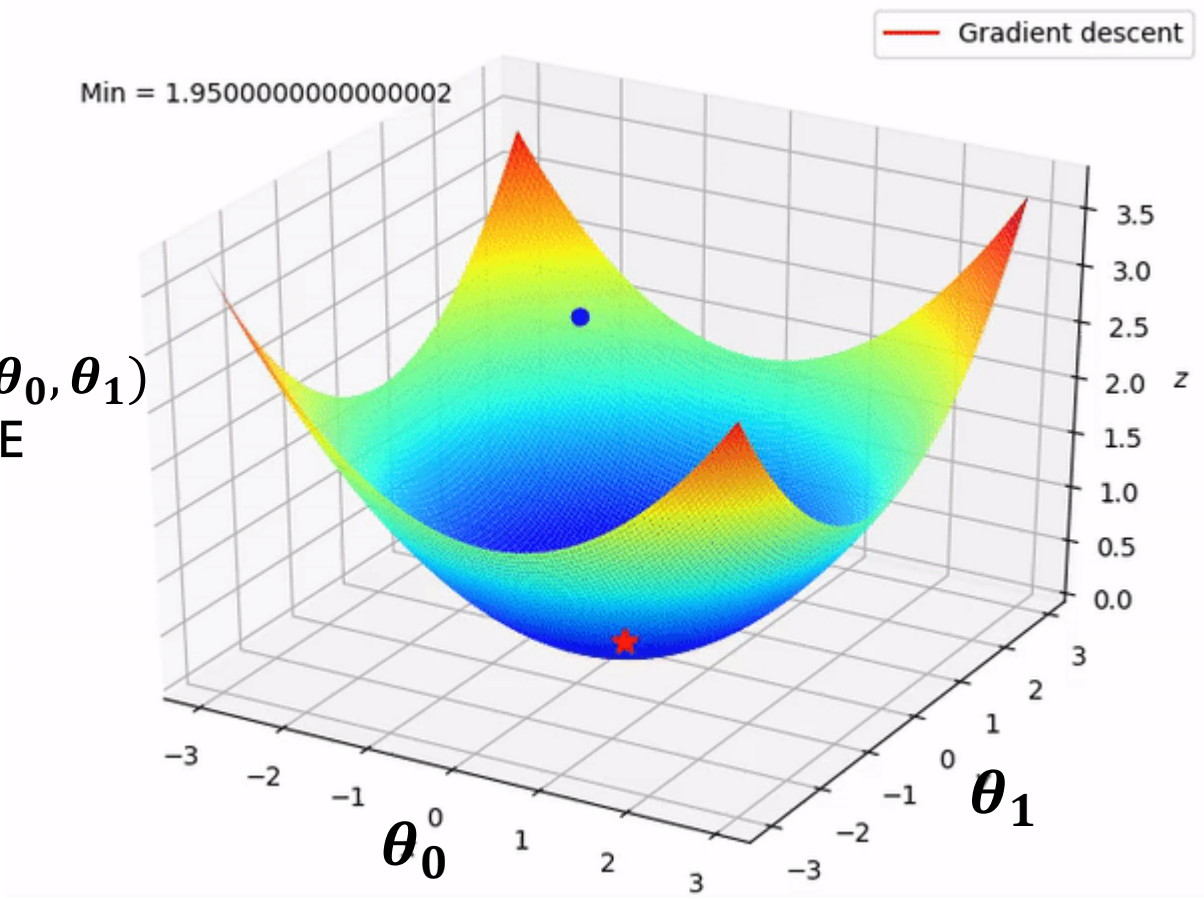


Depending on initial parameter values, we might end-up at a different (local) minimum.

Gradient Descent – Basic idea

Example with a convex error function (MSE).
Only one (global) minimum.

$E(\theta_0, \theta_1)$
MSE



Gradient Descent – Algorithm

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial E}{\partial \theta_j} \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

Learning rate $\alpha > 0$

Derivative of E with respect to θ_j

- At each iteration, we need to update θ_0 and θ_1 simultaneously (at the same time)

Correct (simultaneous update) :

$$\text{temp0} \leftarrow \theta_0 - \alpha \frac{\partial E}{\partial \theta_0}$$

$$\text{temp1} \leftarrow \theta_1 - \alpha \frac{\partial E}{\partial \theta_1}$$

$$\theta_0 \leftarrow \text{temp0}$$

$$\theta_1 \leftarrow \text{temp1}$$

Incorrect :

$$\text{temp0} \leftarrow \theta_0 - \alpha \frac{\partial E}{\partial \theta_0}$$

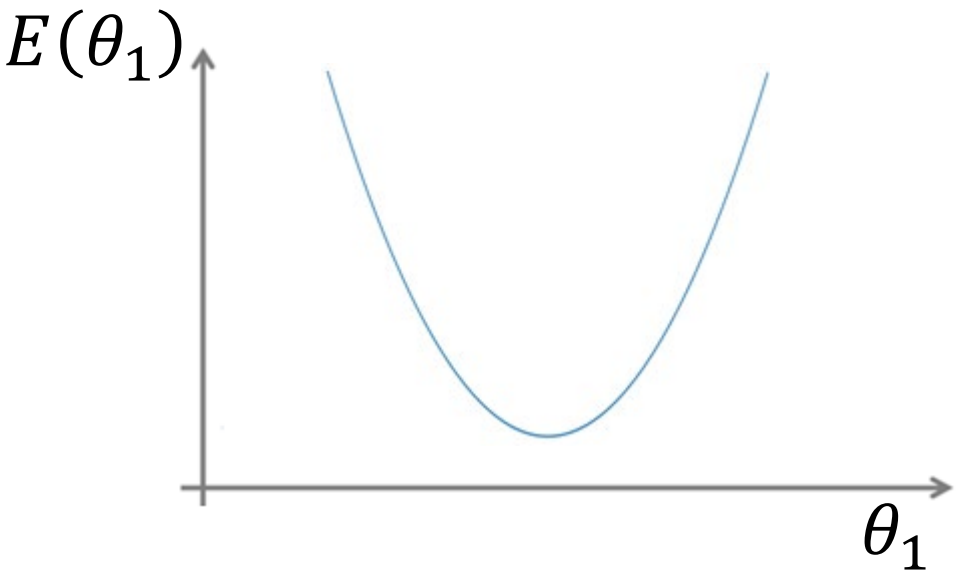
$$\theta_0 \leftarrow \text{temp0}$$

$$\text{temp1} \leftarrow \theta_1 - \alpha \frac{\partial E}{\partial \theta_1}$$

$$\theta_1 \leftarrow \text{temp1}$$

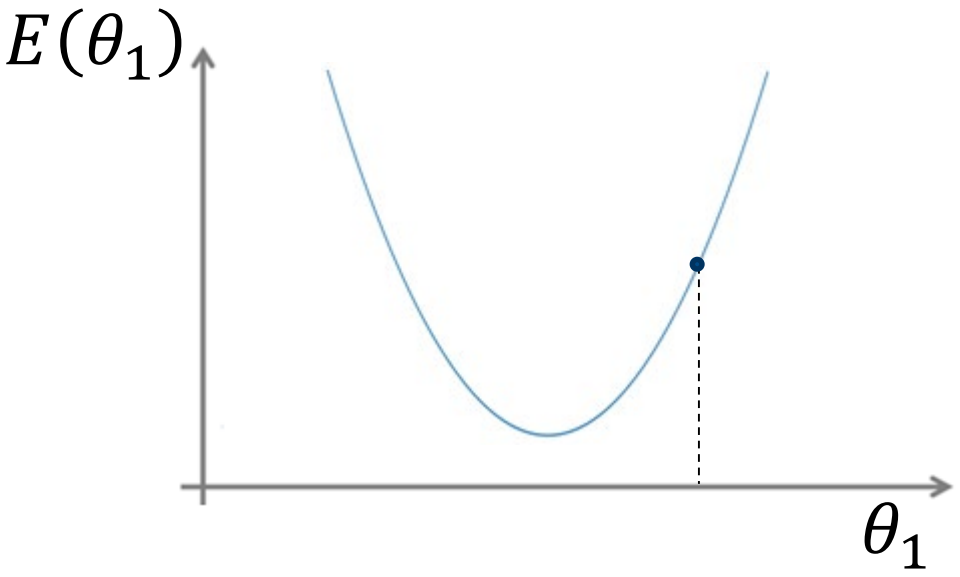
Gradient Descent – Algorithm

Assume for now that we have only one parameter θ_1



Gradient Descent – Algorithm

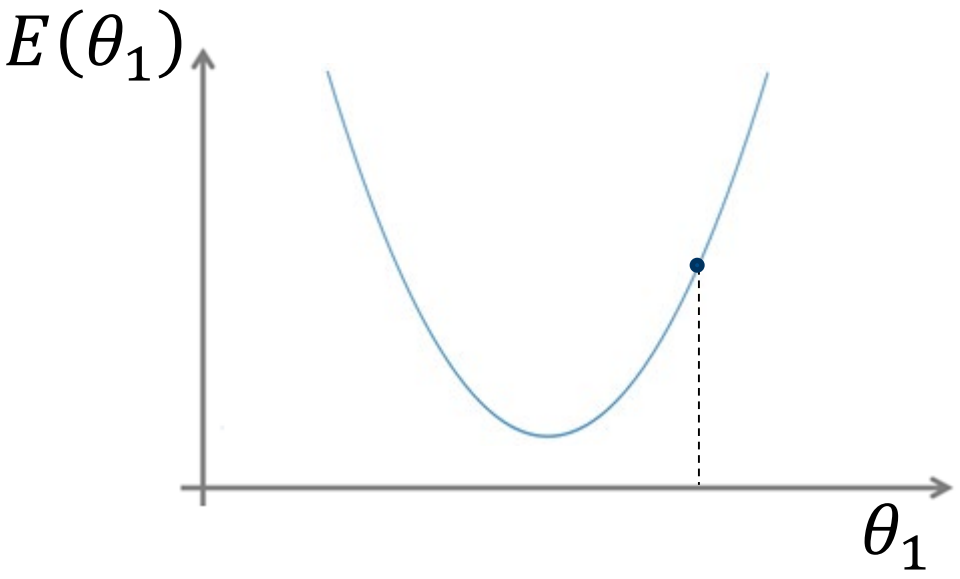
Assume for now that we have only one parameter θ_1



- Pick some initial value for θ_1

Gradient Descent – Algorithm

Assume for now that we have only one parameter θ_1

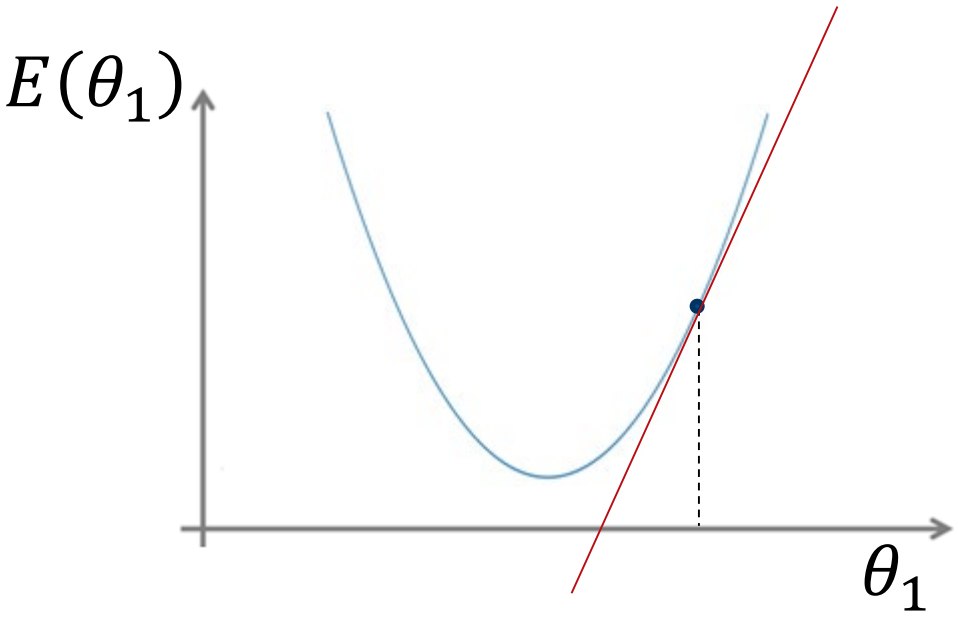


- Pick some initial value for θ_1
- We want to update θ_1 :

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{\partial E}{\partial \theta_1}$$

Gradient Descent – Algorithm

Assume for now that we have only one parameter θ_1



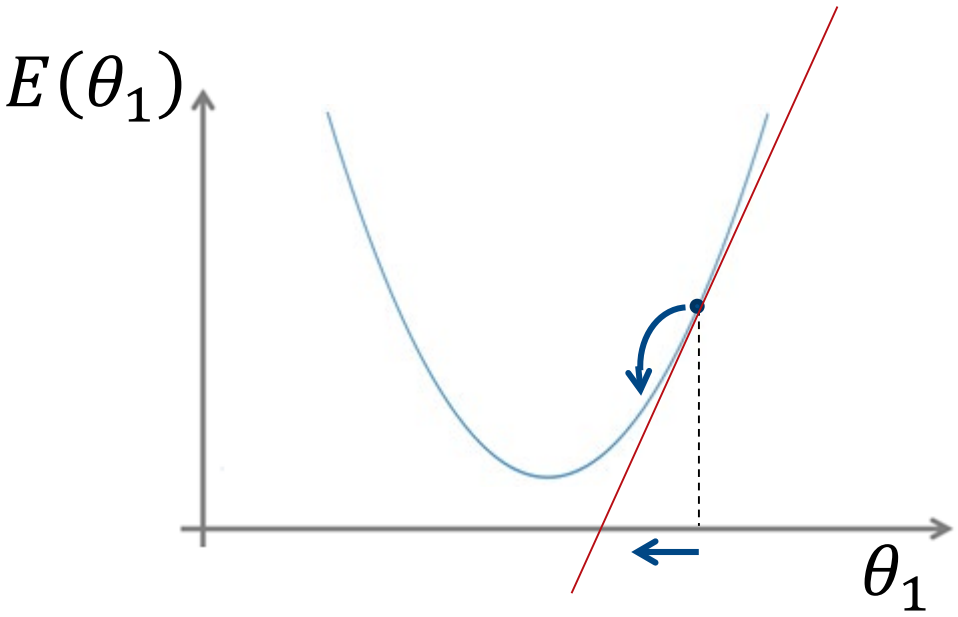
- Pick some initial value for θ_1
- We want to update θ_1 :

$$\theta_1 \leftarrow \theta_1 - \alpha \underbrace{\frac{\partial E}{\partial \theta_1}}$$

Derivative: Looks at the slope of the (red) line which is tangent to the function at that point.

Gradient Descent – Algorithm

Assume for now that we have only one parameter θ_1



- Pick some initial value for θ_1
- We want to update θ_1 :

$$\theta_1 \leftarrow \theta_1 - \alpha \underbrace{\frac{\partial E}{\partial \theta_1}}_{\geq 0}$$

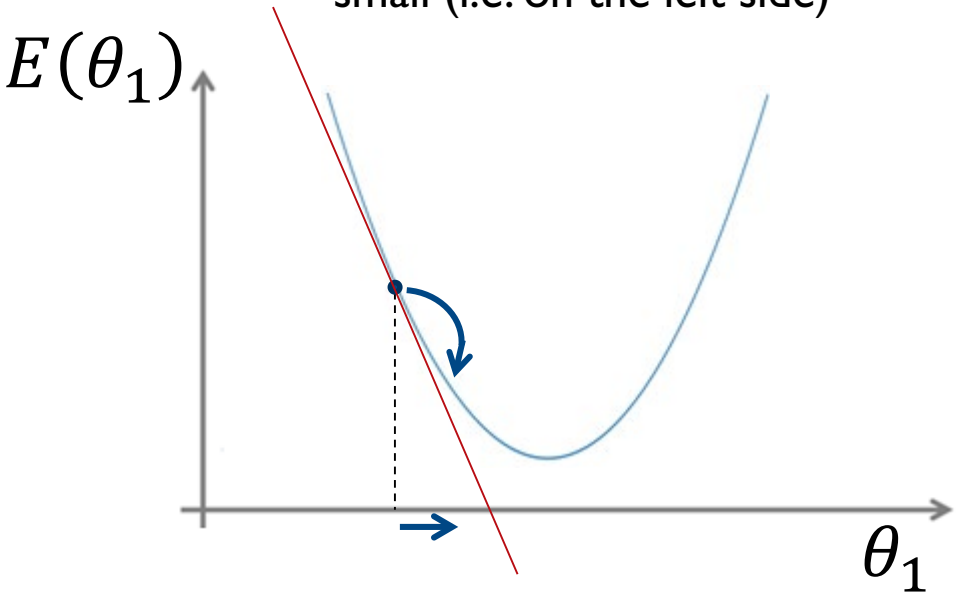
(positive slope)

In this case, since the derivative is positive and $\alpha \geq 0$, then θ_1 will **decrease**, and get's closer to the optimal value.

Gradient Descent – Algorithm

Assume for now that we have only one parameter θ_1

If our initial value of θ_1 was too small (i.e. on the left side)



- Pick some initial value for θ_1
- We want to update θ_1 :

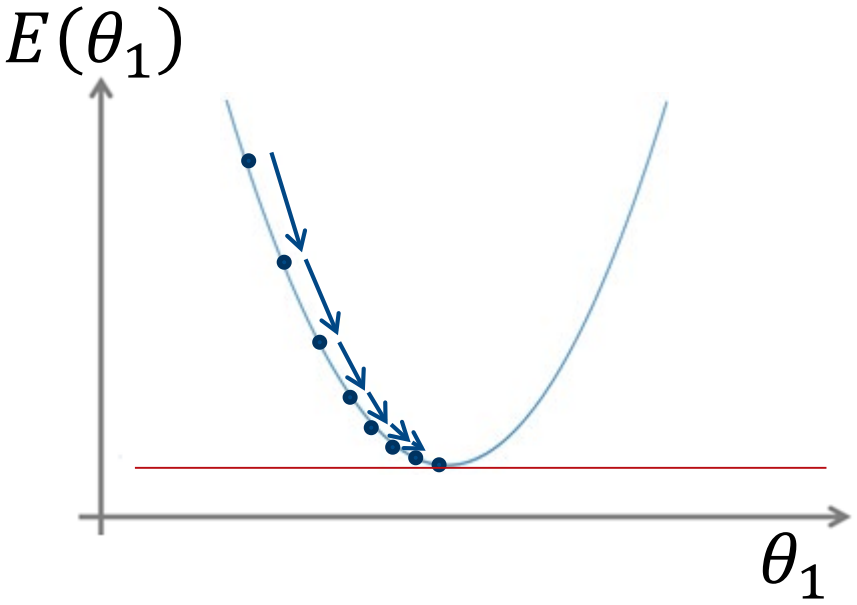
$$\theta_1 \leftarrow \theta_1 - \alpha \underbrace{\frac{\partial E}{\partial \theta_1}}_{\leq 0}$$

(negative slope)

In this case, since the derivative is negative and $\alpha \geq 0$, then θ_1 will **increase**, and get's closer to the optimal value.

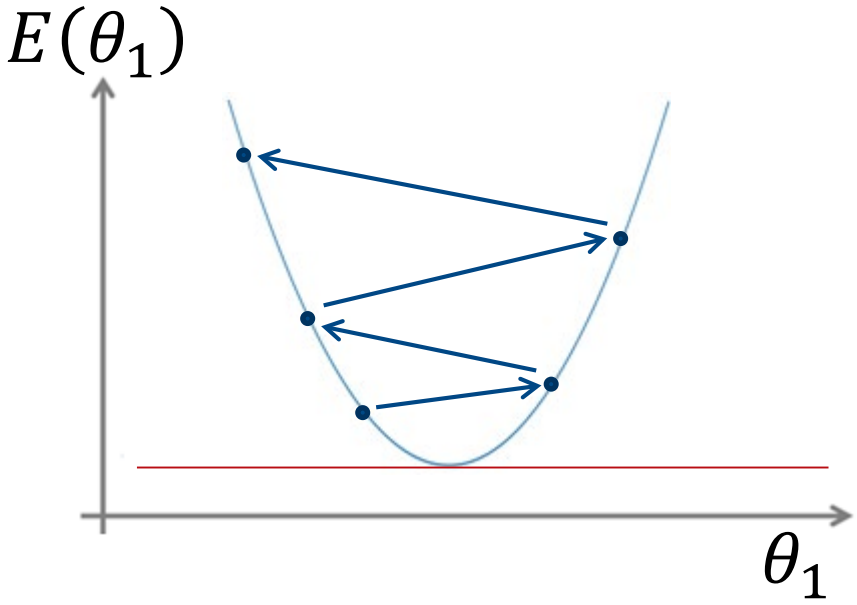
Gradient Descent – Algorithm

- Reasonably small value of α



Notice that as we approach a local minimum, gradient descent will automatically take smaller steps (**why?**). So, no need to decrease α over time.

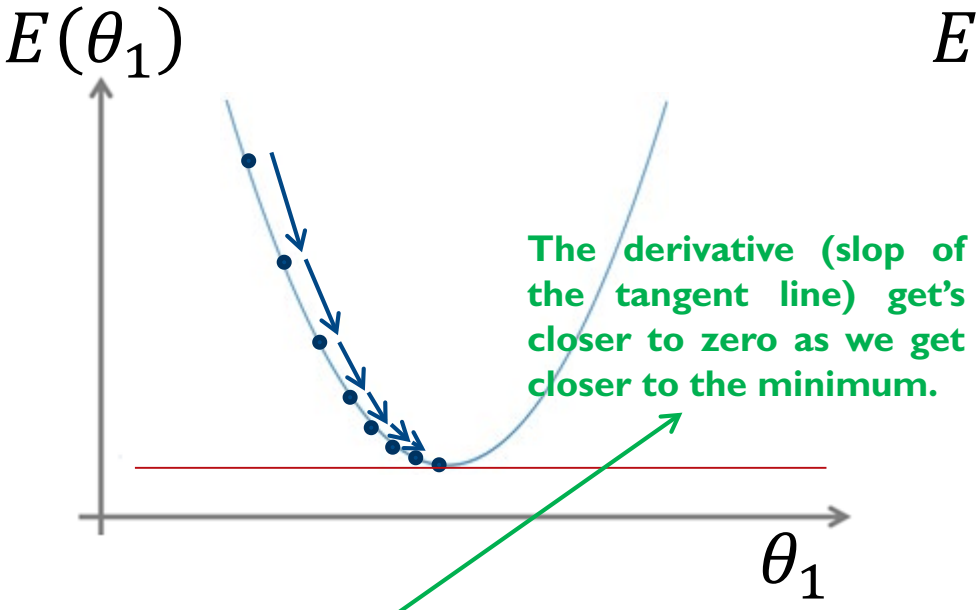
- Very large value of α



If α is too large, it may fail to converge, or may even diverge.

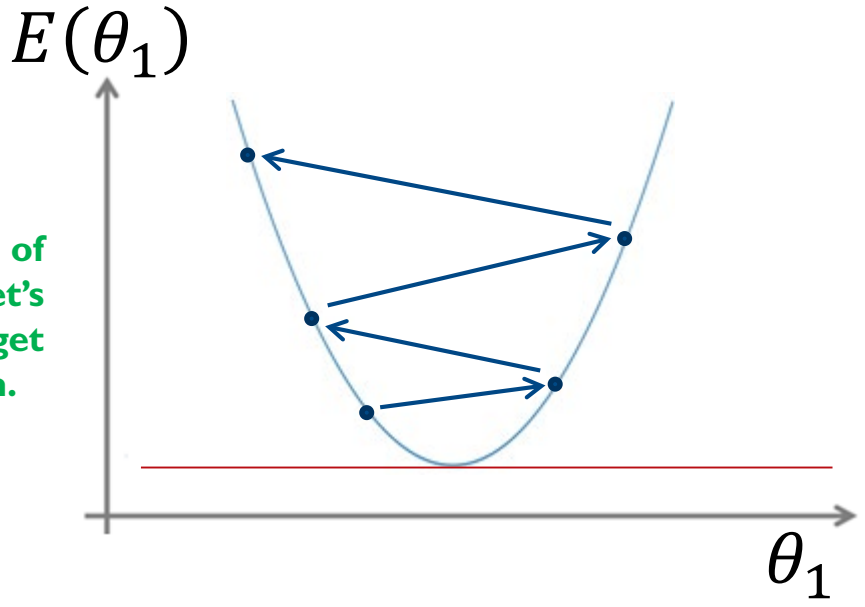
Gradient Descent – Algorithm

- Reasonably small value of α



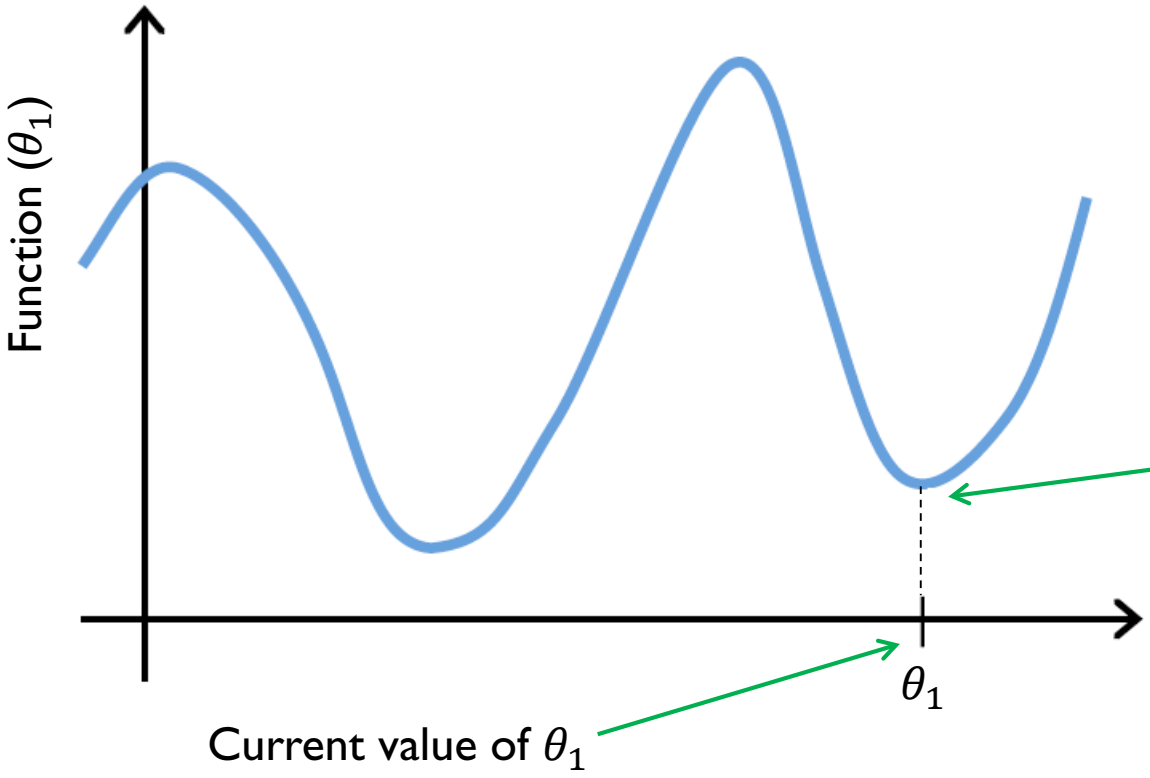
Notice that as we approach a local minimum, gradient descent will automatically take smaller steps (**why?**). So, no need to decrease α over time.

- Very large value of α



If α is too large, it may fail to converge, or may even diverge.

Gradient Descent – Local minimum



Assume that we have reached a local optimum (local minimum here).

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{\partial E}{\partial \theta_1}$$

The derivative will be equal to 0, so **convergence**. We get stuck at this local minimum.

Using Gradient Descent for Linear Regression

(with one feature)

Gradient descent for linear regression

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial E}{\partial \theta_j} \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

$$E(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right]^2$$

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

- **Derivative of $E(\theta_0, \theta_1)$ with respect to θ_0**

$$\text{Let:} \quad g(\theta) = \left[h_\theta(x^{(i)}) - y^{(i)} \right] = \left[\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right]$$

$$\frac{\partial}{\partial \theta_0} E(\theta) = \frac{\partial}{\partial \theta_0} \frac{1}{2n} \sum_{i=0}^{n} g(\theta)^2$$

$$= \frac{1}{2n} \sum_{i=0}^{n} 2g(\theta) \frac{\partial g}{\partial \theta_0} \quad = \quad \boxed{\frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right]}$$

Gradient descent for linear regression

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial E}{\partial \theta_j} \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

$$E(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right]^2$$

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

- **Derivative of $E(\theta_0, \theta_1)$ with respect to θ_1**

$$\text{Let:} \quad g(\theta) = \left[h_\theta(x^{(i)}) - y^{(i)} \right] = \left[\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right]$$

$$\frac{\partial}{\partial \theta_1} E(\theta) = \frac{\partial}{\partial \theta_1} \frac{1}{2n} \sum_{i=0}^{n} g(\theta)^2$$

$$= \frac{1}{2n} \sum_{i=0}^{n} 2g(\theta) \frac{\partial g}{\partial \theta_1} \quad = \quad \boxed{\frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x^{(i)}}$$

Gradient descent for linear regression

- Pick some initial values for θ_0 and θ_1
- Repeat until convergence {

$$\text{temp0} \leftarrow \theta_0 - \alpha \frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right]$$

$$\text{temp1} \leftarrow \theta_1 - \alpha \frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x^{(i)}$$

$$\theta_0 \leftarrow \text{temp0}$$

$$\theta_1 \leftarrow \text{temp1}$$

}

- **Batch Gradient Descent** for linear regression with 1 feature.
- **"Batch"** = each step of the GD uses all training examples.

Gradient descent for linear regression

- The **batch** gradient descent uses all the training examples, to update the model parameters.

$$\theta_0 - \alpha \frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right]$$

$$\theta_1 - \alpha \frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x^{(i)}$$

- The **online** (also called **stochastic**) gradient descent updates the model parameters based on one training example at a time. This can be useful when:

$$\theta_0 - \alpha \left[h_\theta(x^{(i)}) - y^{(i)} \right]$$

$$\theta_1 - \alpha \left[h_\theta(x^{(i)}) - y^{(i)} \right] x^{(i)}$$

  - e.g. (1) you don't have all the training dataset beforehand. Your training examples arrive one by one over time, as a stream.
  - e.g. (2) your training dataset is very big (computationally expensive to use batch GD, or the dataset doesn't fit in memory).

Multivariate Linear Regression

(with multiple features)

Multivariate linear regression

Size (x_1)	Nb rooms (x_2)	Location (x_3)	Nb floors (x_4)	Age (x_5)	...	Price (y)
2104	6	2	2	45	...	460
1416	5	10	1	40	...	230
1534	5	3	2	30	...	315
852	4	2	1	35	...	178
...	...	...	...	...	...	...

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_1 x_2 + \theta_3 x_3 + \ldots + \theta_d x_d$$

- For convenience of notation, define $x_0 = 1$
  - Think of it as an additional feature which equals 1 for all data-points: ${x_0}^{(i)} = 0, \quad \forall i = 1 \ldots n$

$$h_\theta(x) = \theta^T x$$

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \ldots \\ x_d \end{bmatrix} \in \mathbb{R}^{d+1}, \quad \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \ldots \\ \theta_d \end{bmatrix} \in \mathbb{R}^{d+1}$$

Multivariate linear regression

Gradient descent

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x_j^{(i)}$$

}

Simultaneously update all θ_j for $j = 0, \ldots, d$

$$\theta_0 - \alpha \frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x_0^{(i)}$$

$= 1$

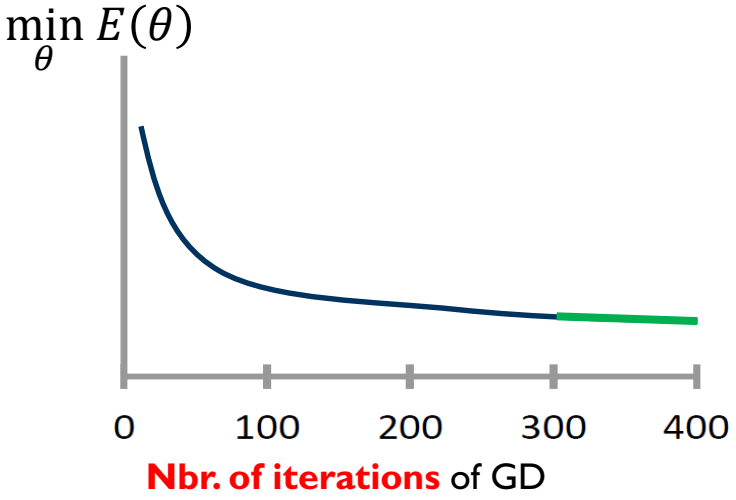
$$\theta_1 - \alpha \frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x_1^{(i)}$$

$$\theta_j - \alpha \frac{1}{n} \sum_{i=0}^{n} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x_2^{(i)}$$

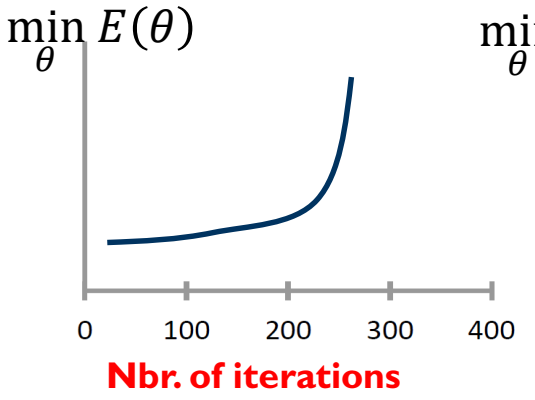
$\ldots$

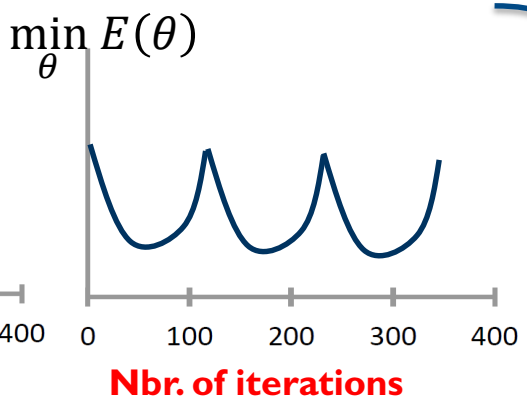
More about gradient descent

Convergence & Selecting α



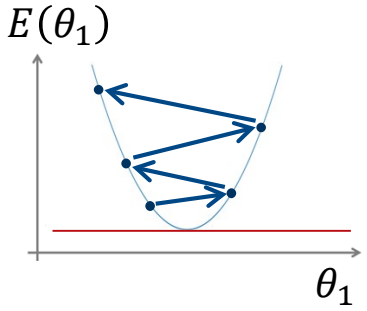
- For a sufficiently small α, the $E(\theta)$ (on the training set) should *decrease* at every iteration.
- One can consider convergence (thus stop) if $E(\theta)$ decreases by less than ϵ (small number e.g. 0.0001) in one iteration.





In these cases, you should use a smaller α

Note: if α is too small, GD can be slow to converge.



Linear Regression with the Normal Equation

(without using gradient descent)

Linear regression with the normal equation

- Method to solve for θ analytically.
- The derivative at the optimal point equals to 0. So, set the derivative to 0 $\frac{\partial}{\partial \theta_j} E(\theta) = \ \ldots = 0,$ and solve for $\theta_0, \theta_1, \ldots, \theta_d$

$$X = \begin{bmatrix} 1 & 2104 & 6 & 2 & 2 & 45 & \ldots \\ 1 & 1416 & 5 & 10 & 1 & 40 & \ldots \\ 1 & 1534 & 5 & 3 & 2 & 30 & \ldots \\ 1 & 852 & 4 & 2 & 1 & 35 & \ldots \\ \ldots & & & & & & \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 230 \\ 315 \\ 178 \\ \ldots \end{bmatrix}$$

- The solution will be:

$$\theta = \underbrace{(X^T X)^{-1}}_{\text{pseudo inverse}} X^T y$$

Linear regression with the normal equation

$$E(\theta) = \frac{1}{2n}\sum_{i=0}^{n}\left[h_\theta(x^{(i)}) - y^{(i)}\right]$$

$$X\theta = \begin{bmatrix} h_\theta(x^{(0)}) \\ h_\theta(x^{(1)}) \\ \cdots \\ h_\theta(x^{(n)}) \end{bmatrix}$$

$$\frac{\partial}{\partial\theta}E(\theta) = 0 \iff \frac{\partial}{\partial\theta}g(\theta) = 0,$$

$$\text{with: } g(\theta) = \sum_{i=0}^{n}\left[h_\theta(x^{(i)}) - y^{(i)}\right]$$

$$\begin{aligned} g(\theta) &= (X\theta - y)^T(X\theta - y) \\ &= ((X\theta)^T - y^T)(X\theta - y) \\ &= (X\theta)^T X\theta - (X\theta)^T y - y^T(X\theta) + y^T y \\ &= \theta^T X^T X\theta - 2(X\theta)^T y + y^T y \end{aligned}$$

$$\frac{\partial}{\partial\theta}g(\theta) = 2X^T X\theta - 2X^T y = 0 \iff X^T X\theta = X^T y$$

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

Gradient Descent vs. Normal Equation
(for linear regression)

Gradient Descent

- ➖ Need to choose α.
- ➖ Needs several iterations.
- ➕ Works well even when d is large.

Normal Equation

- ➕ No need to choose α.
- ➕ Don't needs to iterate.
- ➖ Need to compute $(X^T X)^{-1}$, which is slow when d is very large.
- Does not apply to other optimization problems.

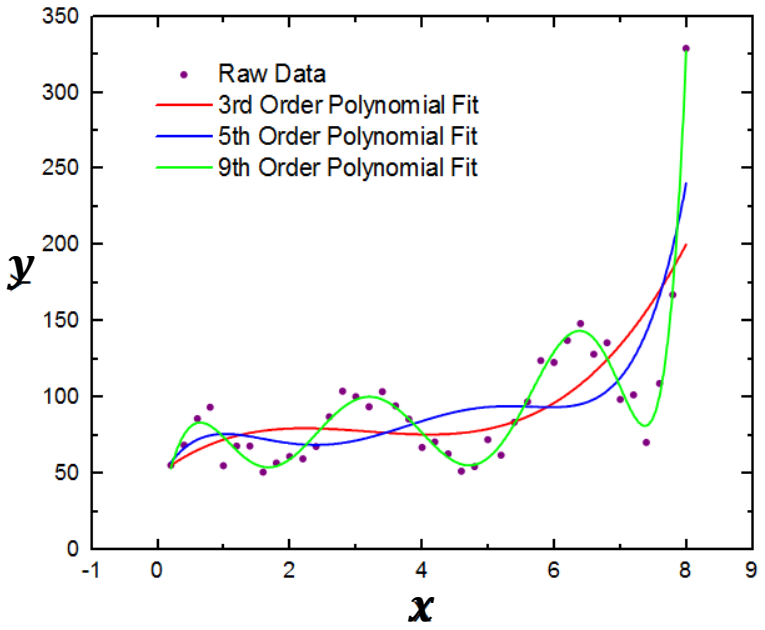
HALMSTAD UNIVERSITY

(2) Non-linear Regression

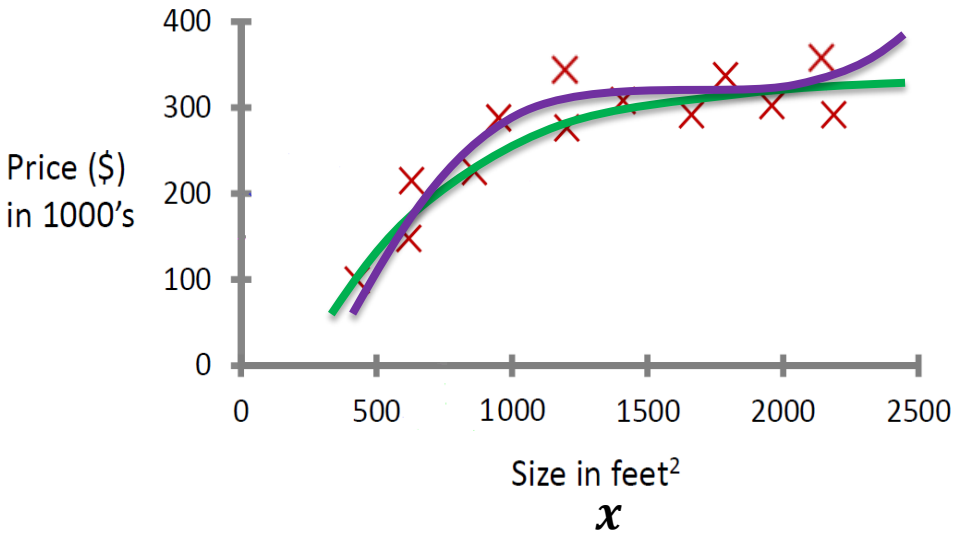
Non-linear regression

- The output $h_\theta(x)$ is a non-linear function
- e.g. polynomial function of degree > 1

- Examples with one variable:

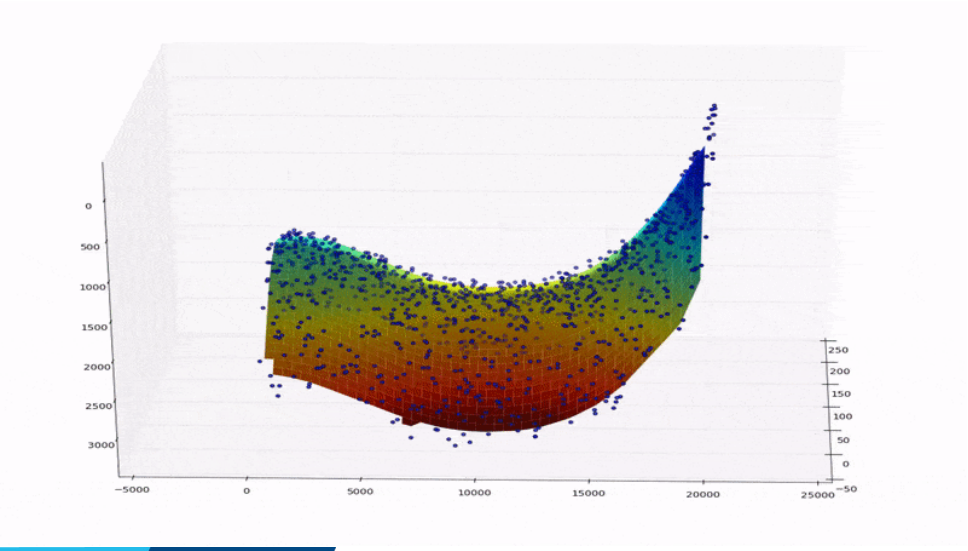


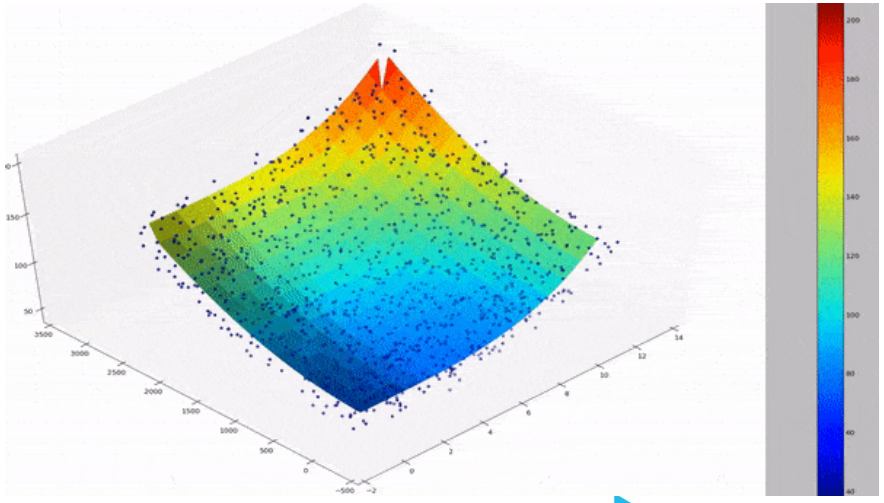
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$



Non-linear regression

- Examples of polynomial regression models with two features x_1, x_2
- A second order polynomial would be:
  - $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2$





Generalized Linear Model for Non-linear Regression

Generalized Linear Model

- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad z = \begin{bmatrix} 1 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

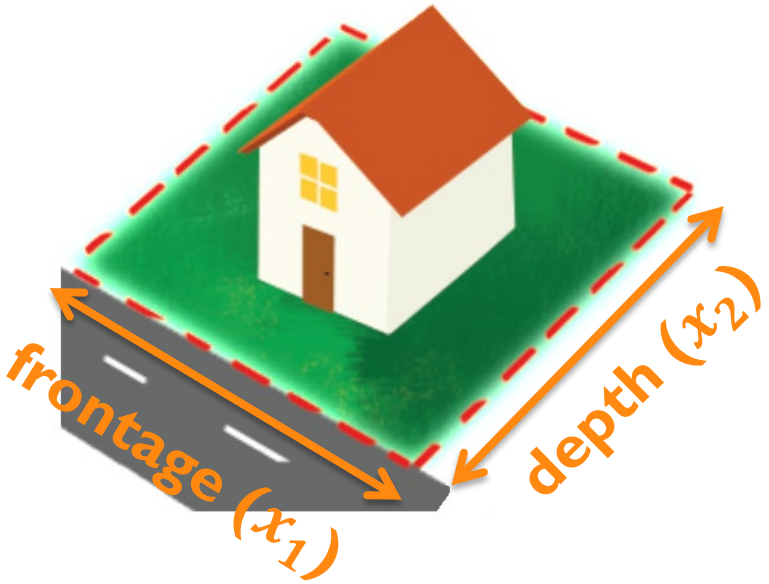
- $h_\theta(x) = \boldsymbol{\theta^T z} = \theta_0 + \theta_1 \overbrace{\boldsymbol{x_1}}^{z_1} + \theta_2 \overbrace{\boldsymbol{x_2}}^{z_2} + \theta_3 \overbrace{\boldsymbol{x_1 x_2}}^{z_3} + \theta_4 \overbrace{\boldsymbol{x_1^2}}^{z_4} + \theta_5 \overbrace{\boldsymbol{x_2^2}}^{z_5}$

- This can be seen as just creating and adding new features based on the two original features x_1, x_2

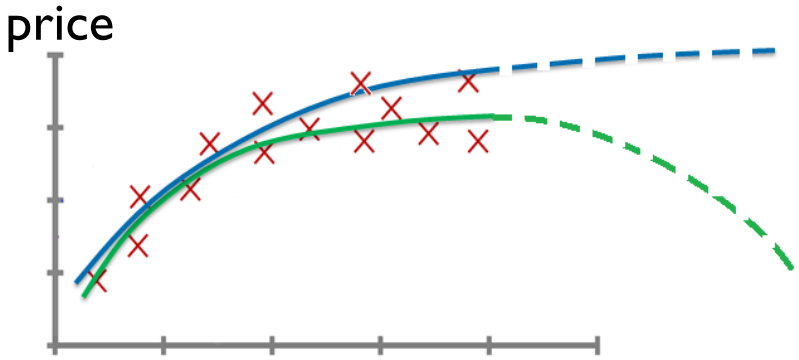
- We can still find the parameters θ of the **non-linear model** (in x) using a **linear model** based on z.
  - So, we can use the methods we studied previously in the linear regression lecture.

Generalized Linear Model

- It can be any kind of **new features** (also called: basis functions)
  - e.g. $\log(x_1)$, $\sqrt{x_2}$, $x_1 x_2$, x_1^2, x_2^3, ...
- Requires a good guess of relevant features for your problem …



- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
- New feature:
  - $land\_area = depth \times frontage = x_1 x_2$



- $h_\theta(x) = \theta_0 + \theta_1 x_1 x_2$
- $h_\theta(x) = \theta_0 + \theta_1 \sqrt{x_1 x_2}$

K-Nearest-Neighbors (KNN) for Non-linear Regression

K-Nearest-Neighbors Regression

- KNN is a *non-parametric* model
  - This does **not** mean parameter-free (e.g. K is a hyper-parameters).
    - **Parametric**: we select a hypothesis space and adjust a *fixed set of parameter* using the training data.
    - **Non-parametric:** the model is not characterized by a fixed set of parameters.

- In KNN, we just save/memorize the training dataset (there is no training as such).

- To make a prediction on a new data-point x, we look at the k most similar (i.e. closest/nearest) data-points from the training dataset. We can take:
  - the **average output** from these k examples,
  - or a **weighted average output** from these k examples.

- Need a distance measure (inverse of: similarity measure).

K-Nearest-Neighbors Regression

Let $u \in \mathbb{R}^d, v \in \mathbb{R}^d,$ be two vectors of the same dimension (d).

Euclidean distance: $\|u - v\| = \sqrt{\sum_{j=1}^{d}(u_j - v_j)^2}$

Let x be a new data-point, and $\{(x^{(l_1)}, y^{(l_1)}), \ldots, (x^{(l_k)}, y^{(l_k)})\}$ be the set of the k-nearest neighbors to x

Predicting using KNN

$$h(x) = \frac{1}{k}\sum_{i=1}^{k} y^{(l_i)}$$

Same as the weighted KNN with $w_{l_i} = 1$

Predicting using Weighted KNN

$$h(x) = \frac{\sum_{i=1}^{k} w_{l_i} y^{(l_i)}}{\sum_{i=1}^{k} w_{l_i}}$$

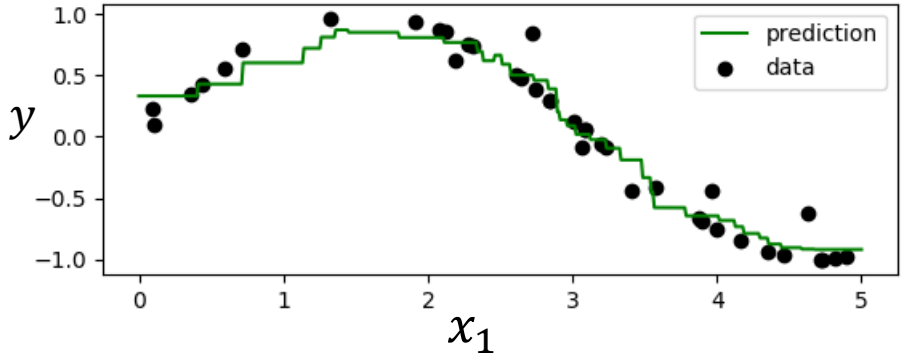
with: $w_{l_i} = \frac{1}{\|x - x^{(l_i)}\|}$

K-Nearest-Neighbors Regression

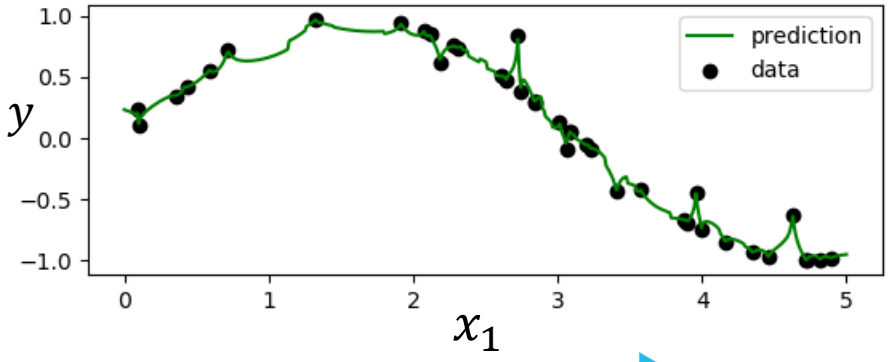
Algorithm

- Given:
  - Training dataset $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$
  - Testing data-point x to predict its output.
- Algorithm:
  - Compute $||x - x^{(i)}||$ for $i = 1, \ldots, n$
  - Select the k training examples closest to x (with the smallest distance to x)
    - $\{(x^{(l_1)}, y^{(l_1)}), (x^{(l_2)}, y^{(l_2)}), \ldots, (x^{(l_k)}, y^{(l_k)})\}$
  - Output the mean (or weighted mean) of $y^{l_1}, y^{l_2}, \ldots, y^{l_k}$

KNN with uniform weights (k=5)



Weighted KNN (k=5)



Kernel Regression (Non-linear Regression)

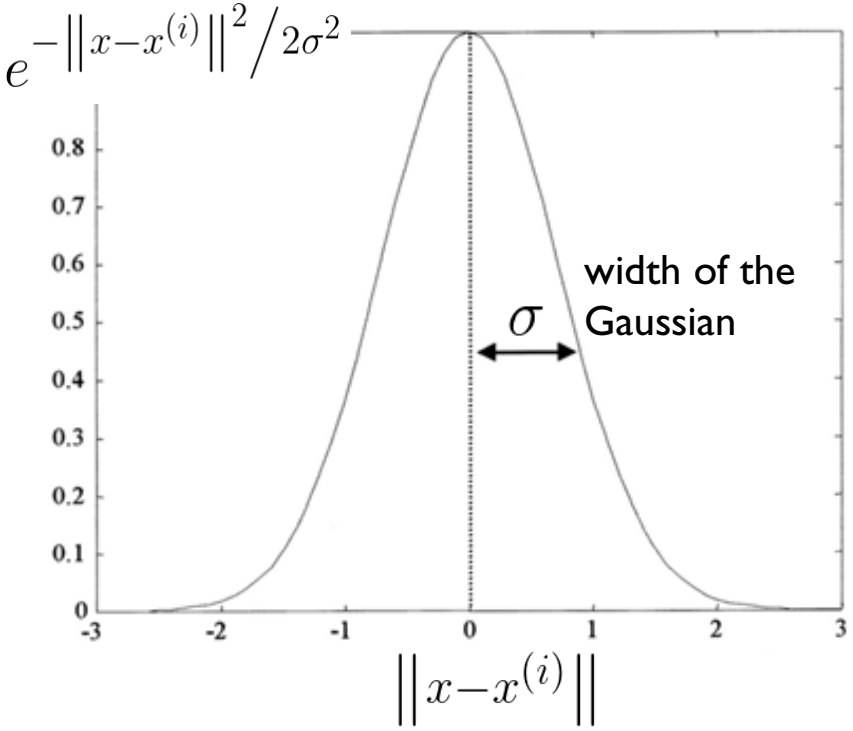
Kernel Regression

- Very similar to the weighted KNN method.
- A common kernel regression model is the Nadaraya-Watson estimator, with a **Gaussian kernel function**.

$$k(u, v) = e^{-\|u-v\|^2 \ / \ 2\sigma^2}$$

$$\text{with } \gamma = \frac{1}{\sigma^2},\ k(u, v) = e^{-\gamma \ \|u-v\|^2}$$

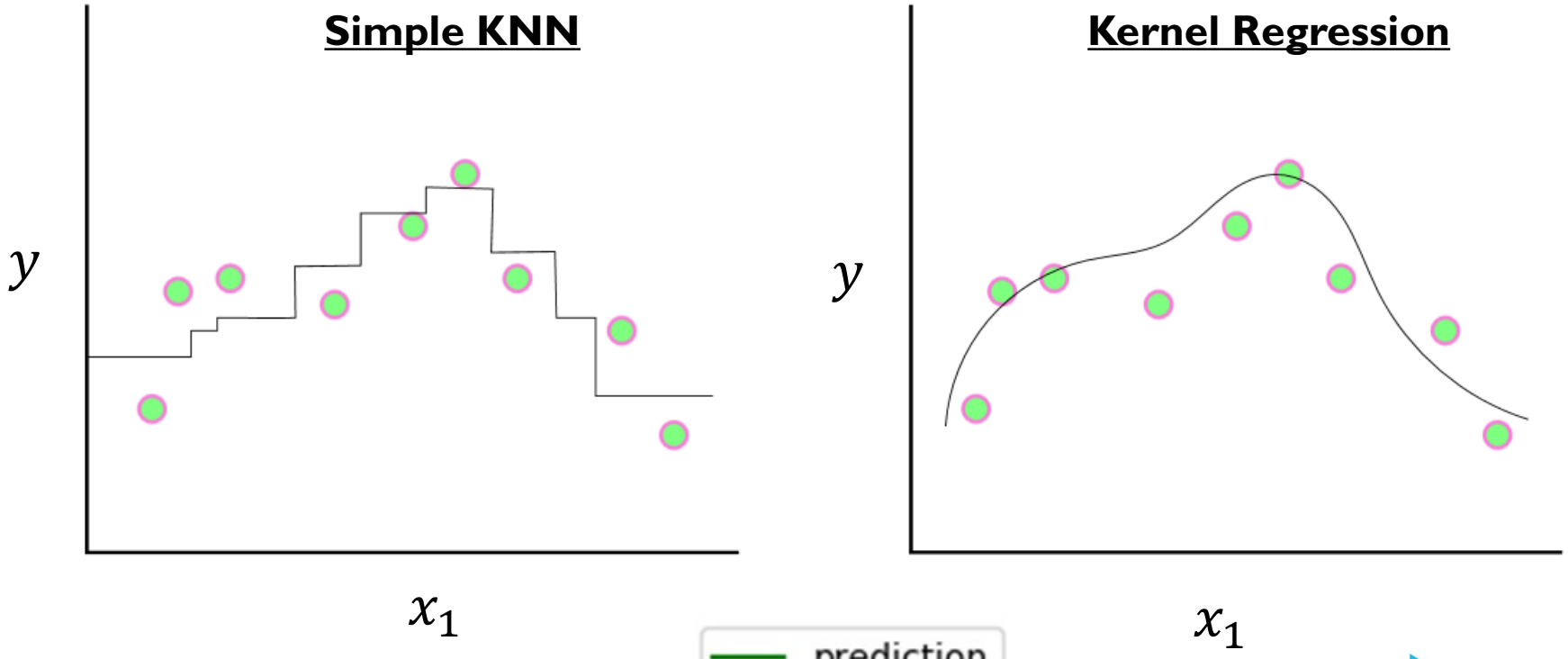
$$h(x) = \frac{1}{\sum_{i=1}^{n} k(x, x^{(i)})} \sum_{i=1}^{n} k(x, x^{(i)}) \ y^{(i)}$$



- Data-points closer to x have a larger weight, i.e. influences the prediction more.
- Larger values of σ implies that more data-points will influence the prediction.
- Too small σ may lead to overfitting. Too large σ may lead to underfitting.

Kernel Regression

- Example with KNN Regression and Kernel Regression



HALMSTAD UNIVERSITY

Features Normalization / Scaling

Features Normalization

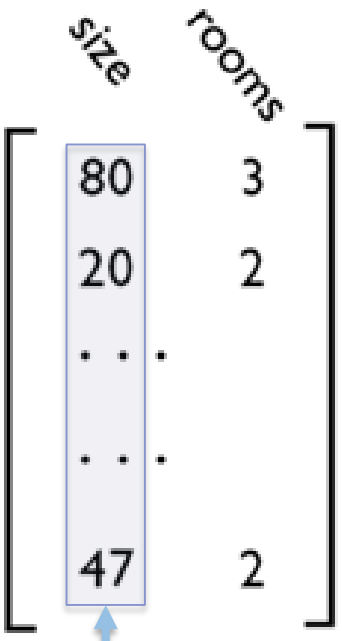
- Feature normalization is a preprocessing step used to normalize the range of the features.

- It is important when the features have very different scales.
  - For example, if the values of **feature 1** are $\in [0, 1]$ but the values of **feature 2** are $\in [120, 190]$, then normalizing the features is important.

- Motivation:
  - Suppose that some ML algorithm computes the Euclidean distance between two points. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

$$\left\| \begin{bmatrix} 0.11 \\ 182 \end{bmatrix} - \begin{bmatrix} 0.52 \\ 179 \end{bmatrix} \right\| = 3.027$$

min-max Features scaling

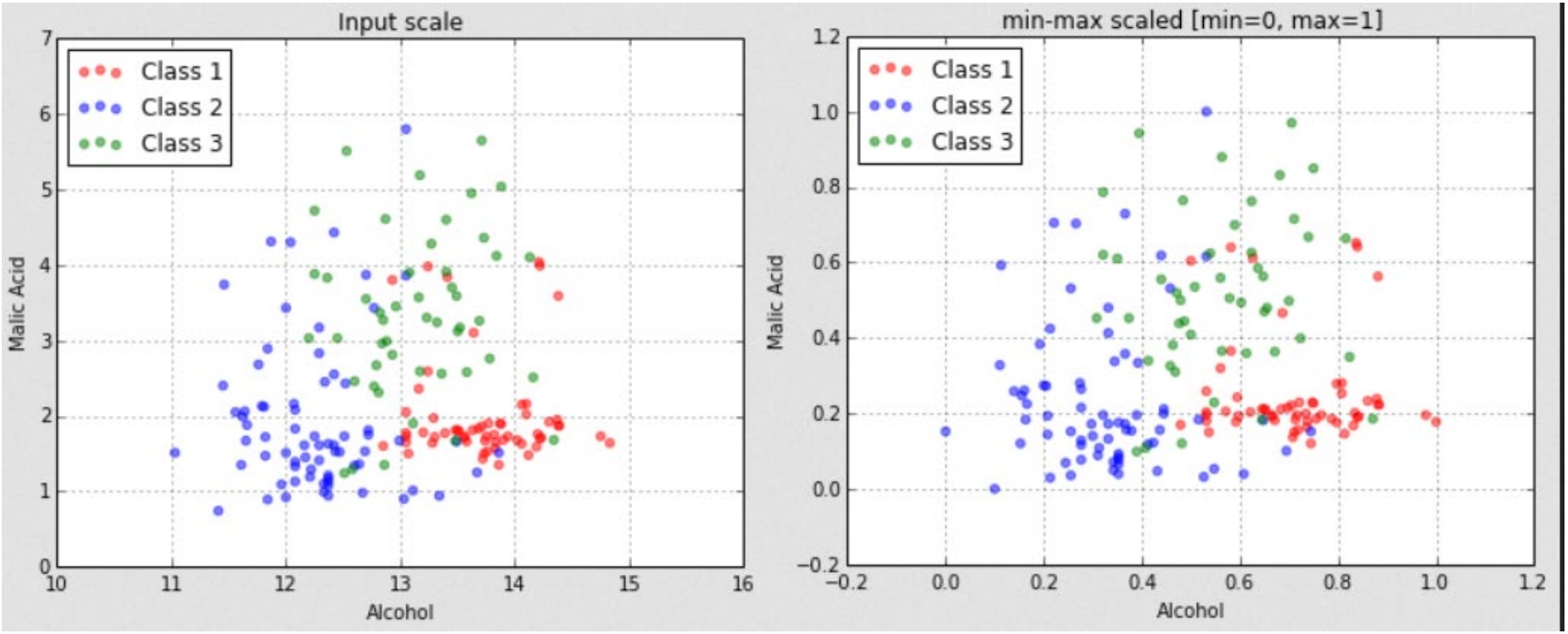


$$v'_j = \frac{v_j - \min(v_j)}{\max(v_j) - \min(v_j)}$$

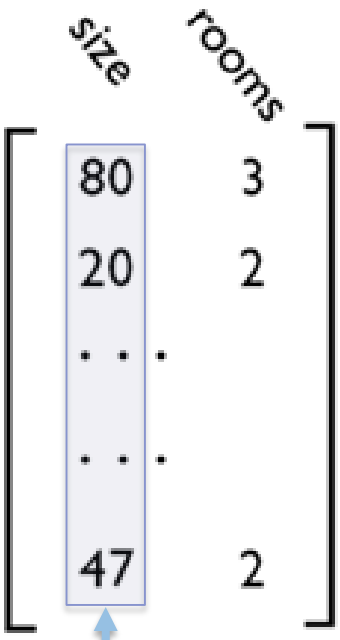
- v_j is a column (corresponding to feature j) from the data matrix X.
- v'_j are the normalized values of feature j. These values will be $\in [0, 1]$

min-max Features scaling

- Before Features Scaling
- After Features Scaling



Features Standardization

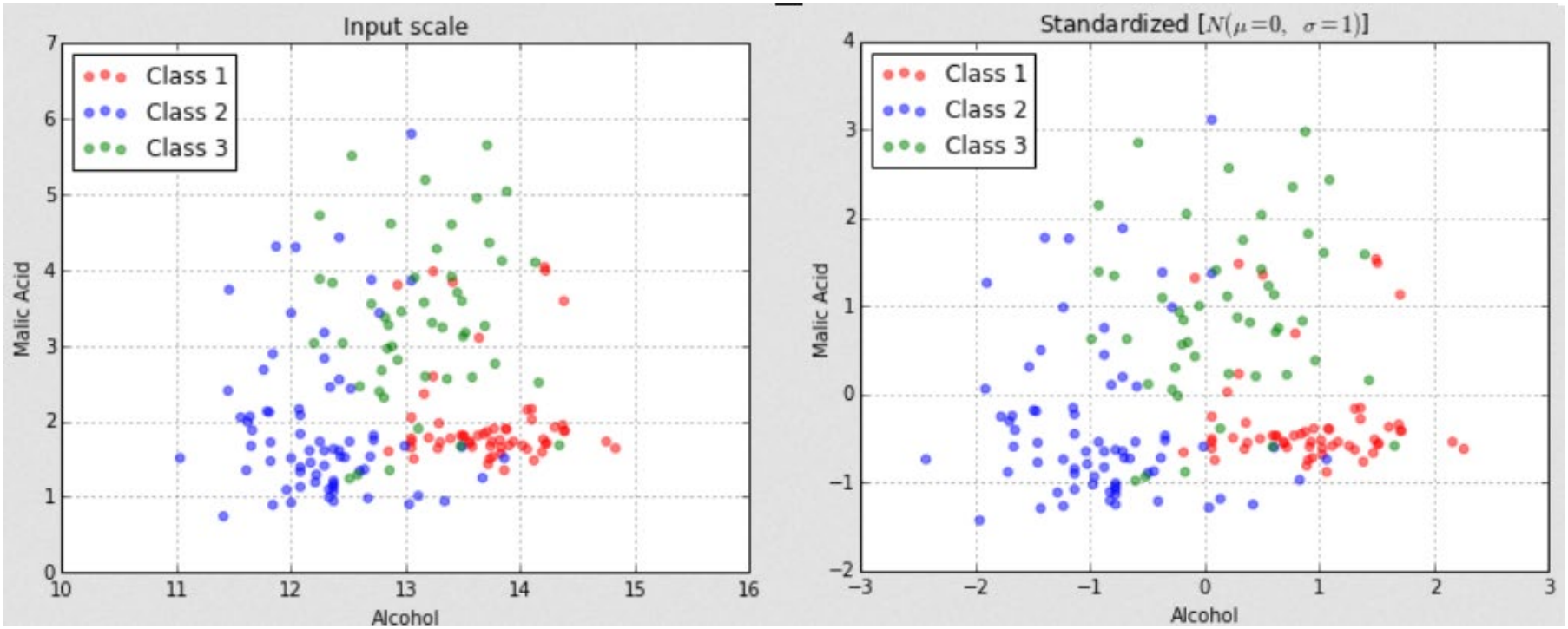


$$v'_j = \frac{v_j - \text{mean}(v_j)}{\text{stdev}(v_j)}$$

- v_j is a column (corresponding to feature j) from the data matrix X.
- v'_j are the normalized values of feature j. These values will be $\in [0, 1]$.
- To normalize, we just subtract the mean and divide by the standard deviation.

Features Standardization

- Before
- After



- NOTE: do not rescale or standardize the output (target variable).

```
>>> X
array([[  2,  46, 497],
       [  7,  76, 326],
       [  4,  29, 683],
       [  8,  55, 123]])
>>>
>>> X.min(axis=0)
array([  2,  29, 123])
>>>
>>> X.max(axis=0)
array([  8,  76, 683])
>>>
>>> X.mean(axis=0)
array([  5.25,  51.5 , 407.25])
>>>
>>> X.std(axis=0)
array([  2.384848  ,  16.94845126, 207.05841567])
>>>
>>> X_scaled = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
>>> X_scaled
array([[0.        , 0.36170213, 0.66785714],
       [0.83333333, 1.        , 0.3625    ],
       [0.33333333, 0.        , 1.        ],
       [1.        , 0.55319149, 0.        ]])
>>>
>>> X_standardized = (X - X.mean(axis=0)) / X.std(axis=0)
>>> X_standardized
array([[-1.36277029, -0.32451343,  0.43345256],
       [ 0.73379939,  1.44555981, -0.39240134],
       [-0.52414242, -1.32755493,  1.33174978],
       [ 1.15311332,  0.20650854, -1.372801  ]])
>>>
```

$$v'_j = \frac{v_j - \min(v_j)}{\max(v_j) - \min(v_j)}$$

$$v'_j = \frac{v_j - \text{mean}(v_j)}{\text{stdev}(v_j)}$$