Universitat de les Illes Balears

Machine Learning

Lesson 2: Supervised Learning

Linear Models (LMS, Logistic Regression, Perceptron)

Remember: a simple example…

Size (feet2)	*Number of bedrooms*	*Number of floors*	*Age of home (years)*	*Price ($1000)*
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

… for dataset notation

Dataset = $\{(\mathbf{x}^{(i)}, y^{(i)}); i = 1, \dots m\}$

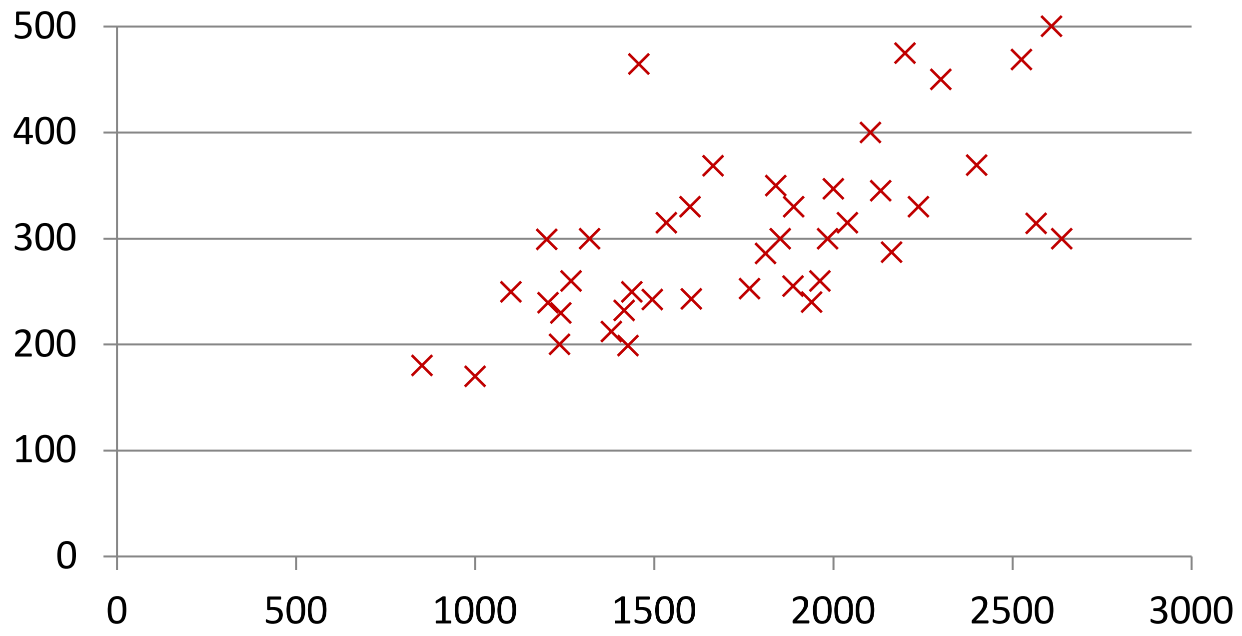
$(\mathbf{x}^{(i)}, y^{(i)})$ = *Training example*

$\mathbf{x}^{(i)}$= *"input" variable (features),* $\mathbf{x}^{(i)}$= $\left(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\right)$

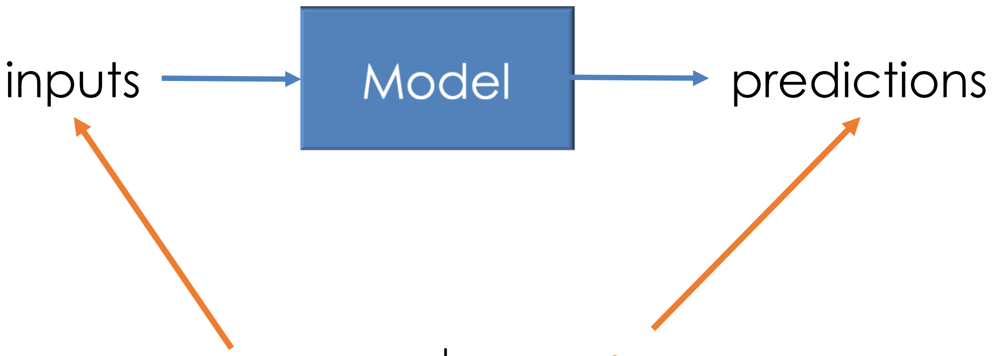
$y^{(i)}$ = *"output" variable (target),* $y \in \mathcal{Y}$

What price...?

Size (feet^2)	Price ($) in 1000's
2104	460
1416	232
1534	315
852	178
…	…



ML Model: regression

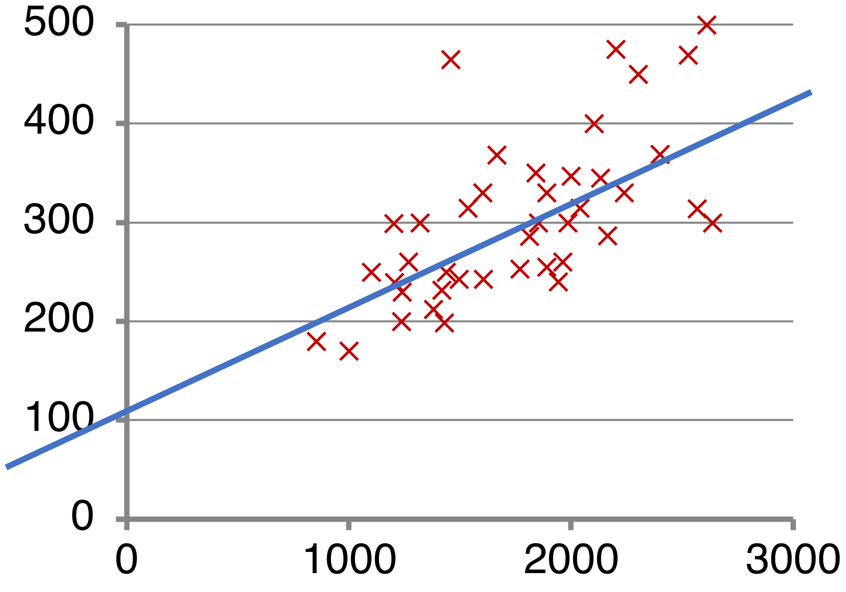


Size (feet^2)	Price ($) in 1000's
2104	460
1416	232
1534	315
852	178
…	…

Linear regression

Hypothesis: the model is linear

$$h_\theta(x) = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

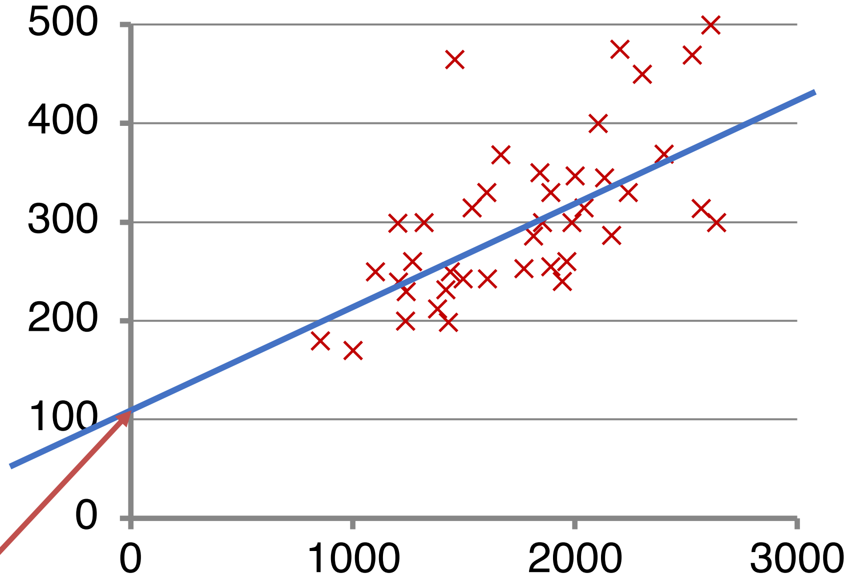


Case n=1:

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

Linear regression

Hypothesis: the model is linear



$$h_\theta(x) = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

is called the **bias** because it represents the intercept or prior (prejudice)

Linear regression

Hypothesis: the model is linear

$$h_\theta(x) = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

n = number of input dimensions

Generalization

(in matrix notation)

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

x_0=1 (Intercept term)

Loss function

Given a training set, how do we learn the parameters?

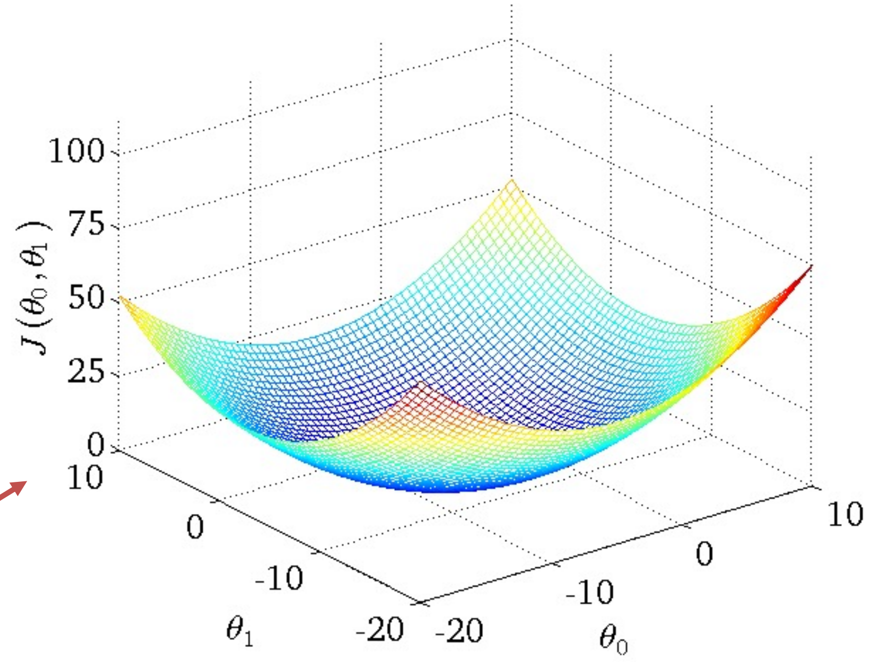
<u>Basic idea</u>: to make h_θ close to y

We've to define a function that measures, for each parameters' values, how close the $h_\theta(x^{(i)})$'s are to the corresponding $y^{(i)}$'s

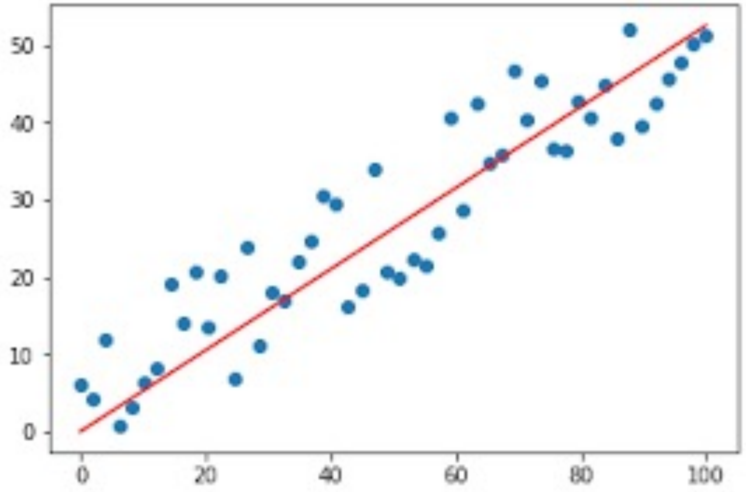
Quadratic loss function

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Case n=1:



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

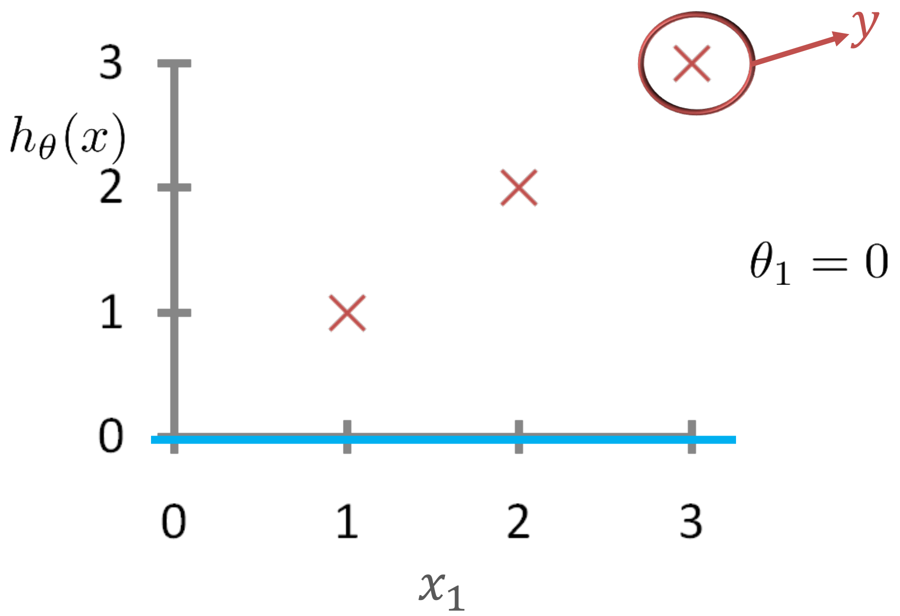


X	Y	$h(\theta_0=0, \theta_1=0.5)$	$MSE = (h(\theta=0.5) - y)^2$
1	1		
20	12		

Quadratic loss function: example

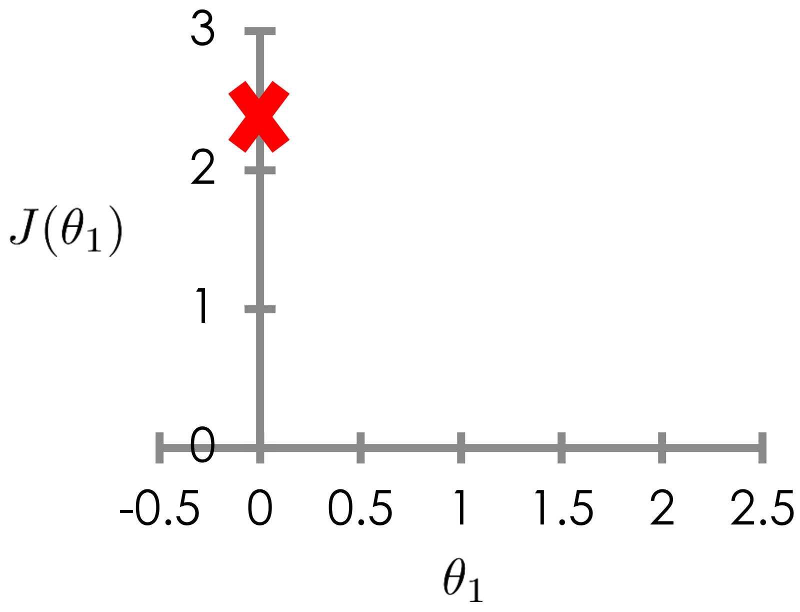
$h_\theta(x) = \theta_1 x_1$

(for a fixed θ_1, this is a function of x)



$J(\theta_1)$

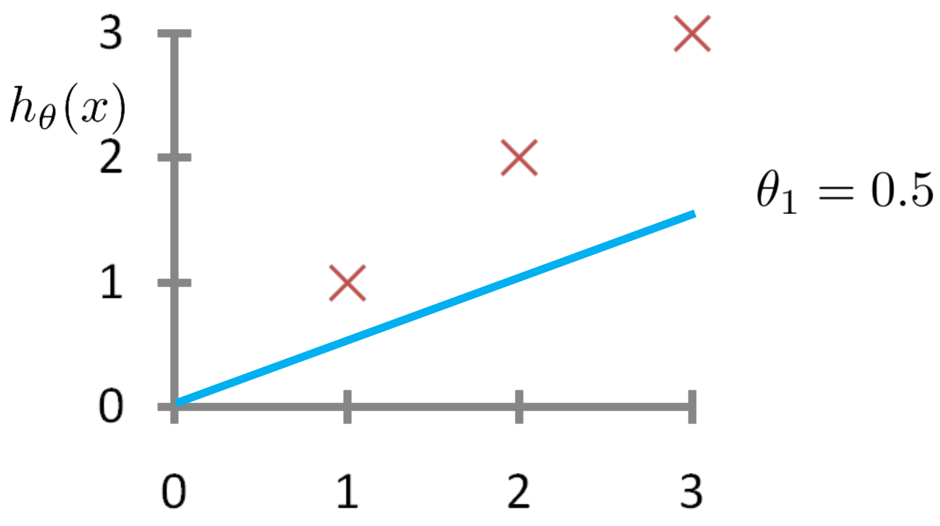
(function of the parameter θ_1)



Quadratic loss function: example

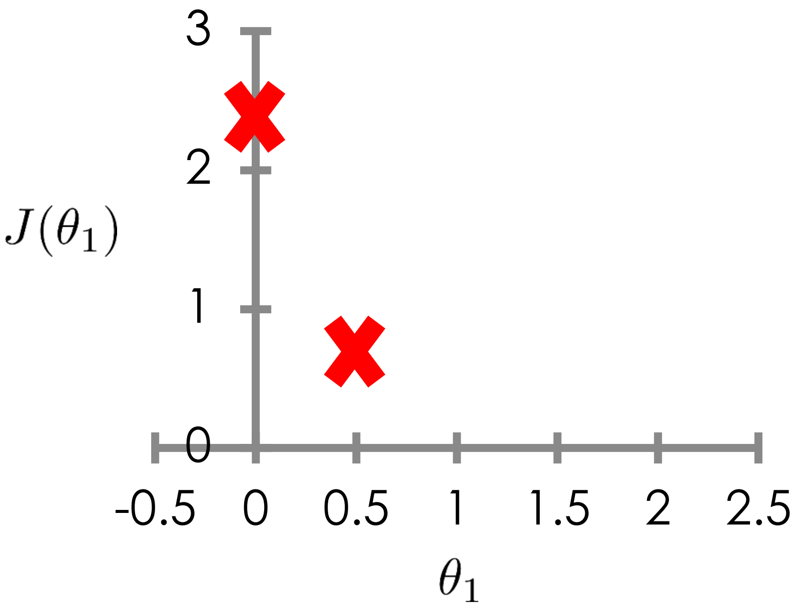
$h_\theta(x) = \theta_1 x_1$

(for a fixed θ_1, this is a function of x)



$J(\theta_1)$

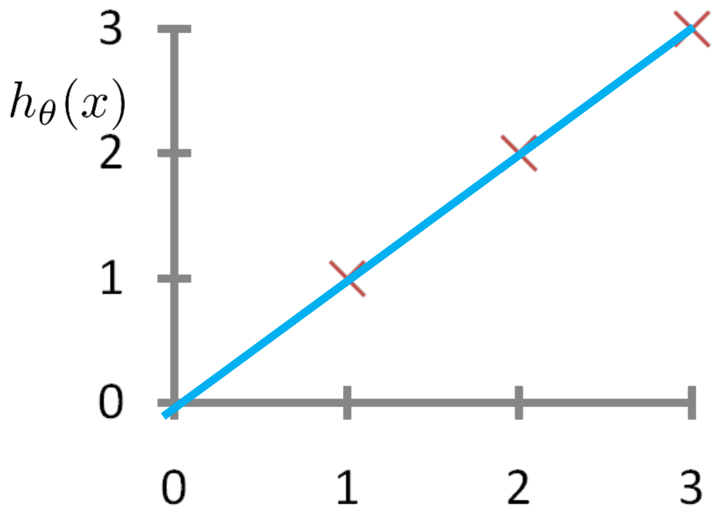
(function of the parameter θ_1)



Quadratic loss function: example

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

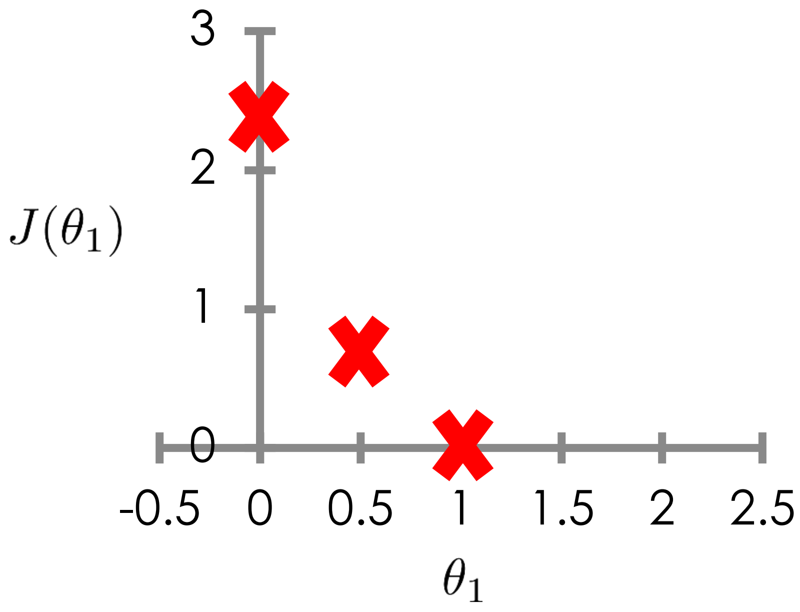
(for a fixed θ_1, this is a function of x)



$\theta_1 = 1$

$J(\theta_1)$

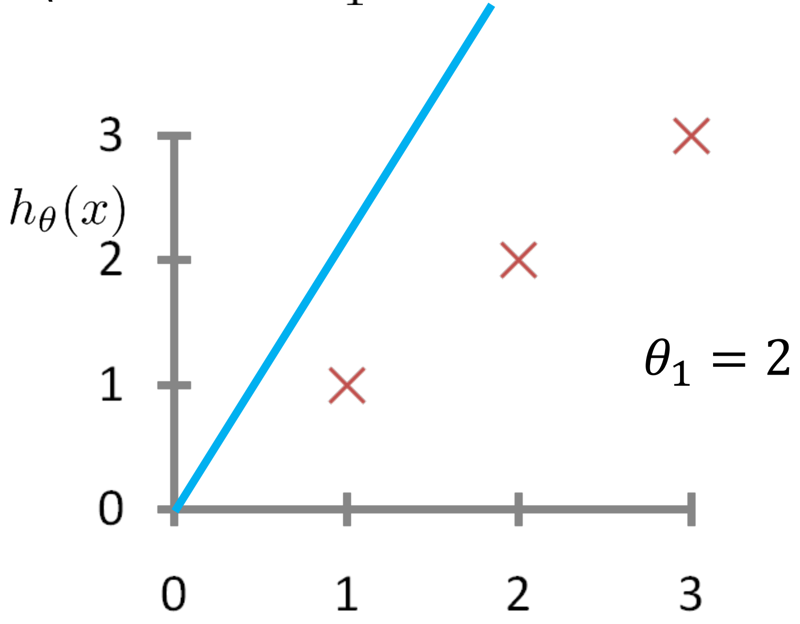
(function of the parameter θ_1)



Quadratic loss function: example

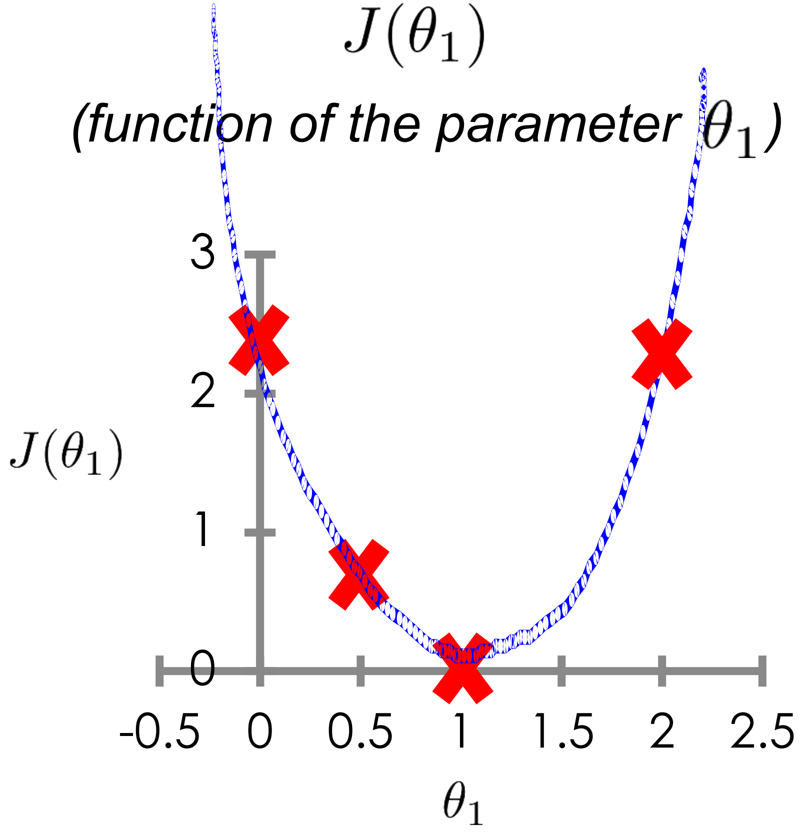
$h_\theta(x) = \theta_1 x_1$

(for a fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



Analytic Linear regression

Dataset = $X, \vec{y}$, where X is a matrix $m \times (n+1)$

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

$$\vec{y} = X\theta$$

$$X\theta = \vec{y}$$

$$X^T X\theta = X^T\vec{y}$$

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

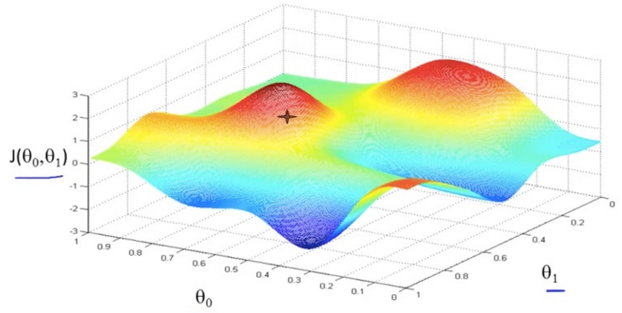
Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

full derivation

LMS learning algorithm

Loss function: Quadratic

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

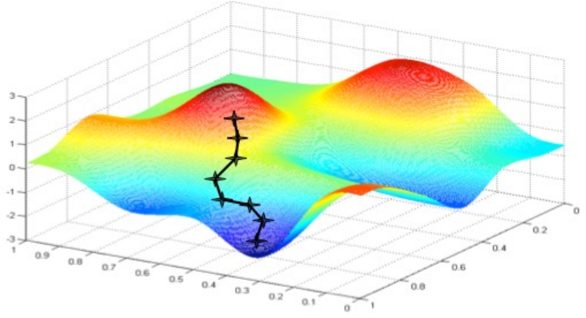


Goal: To minimize $J(\theta)$

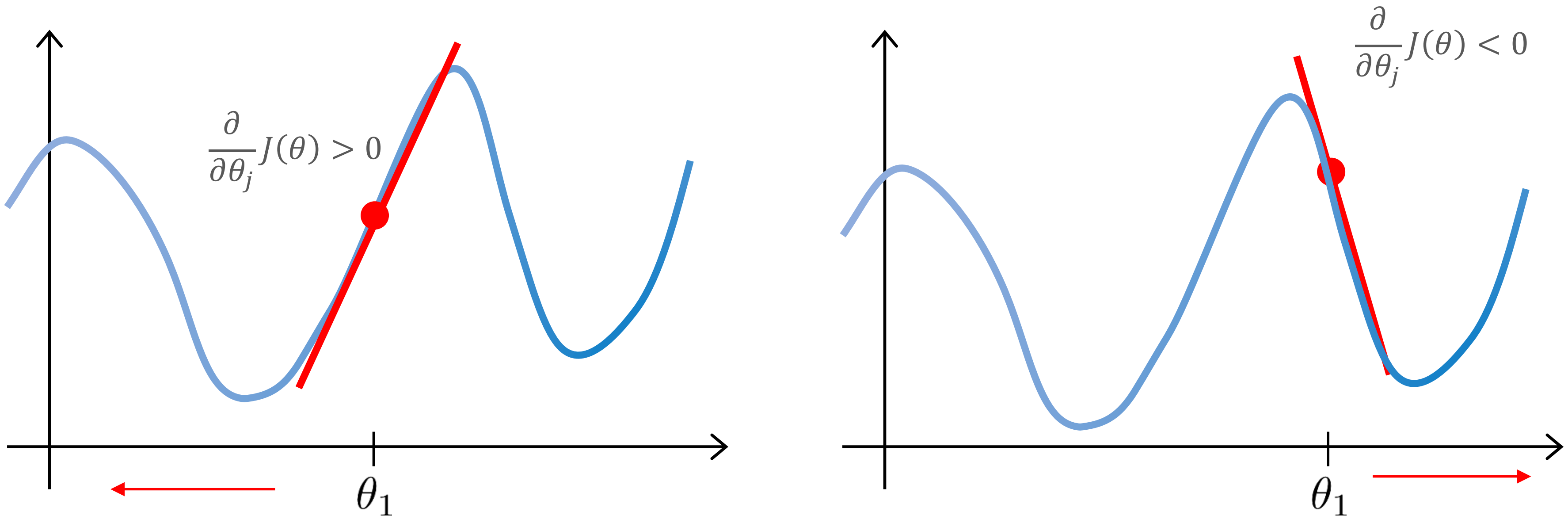
Gradient descent

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

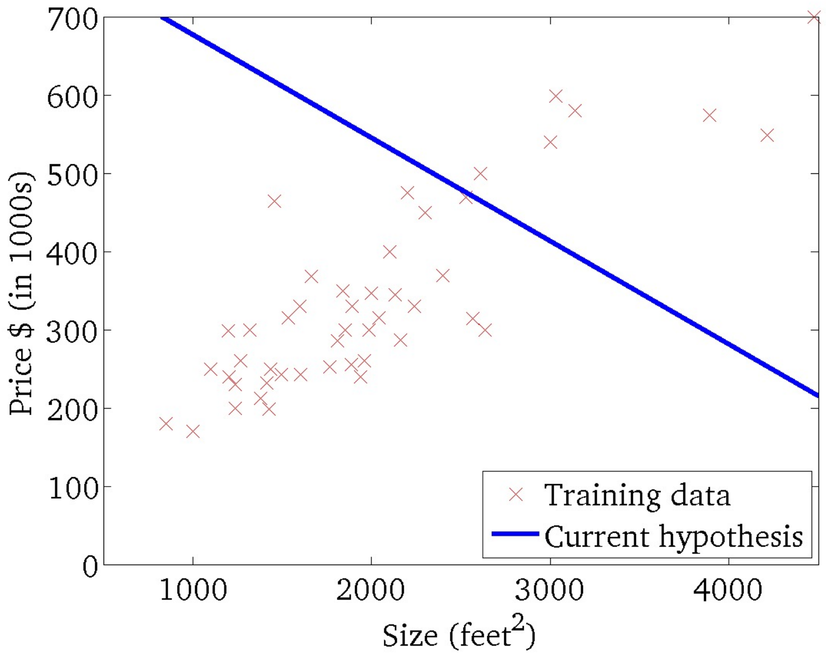
Learning rate

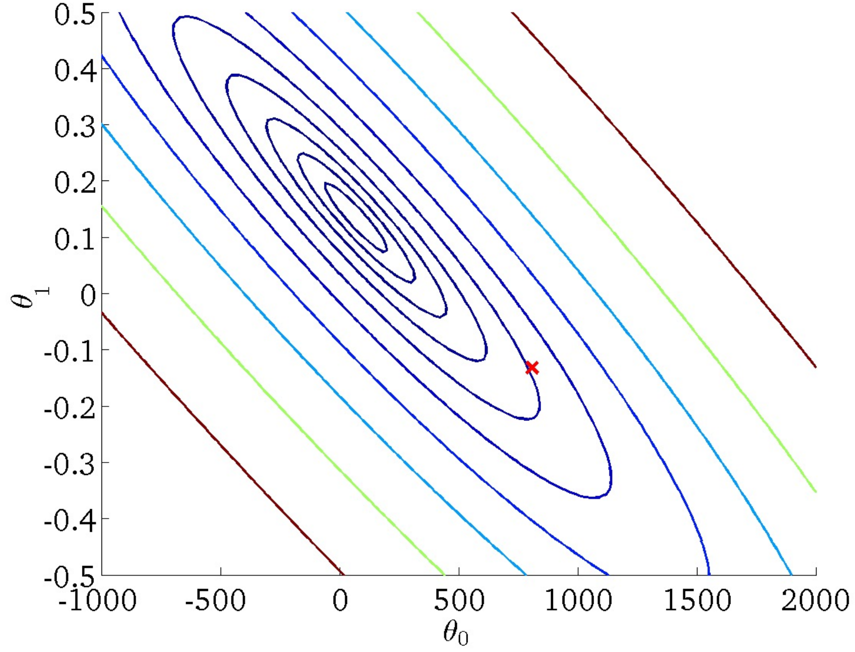


Gradient descent: basic idea

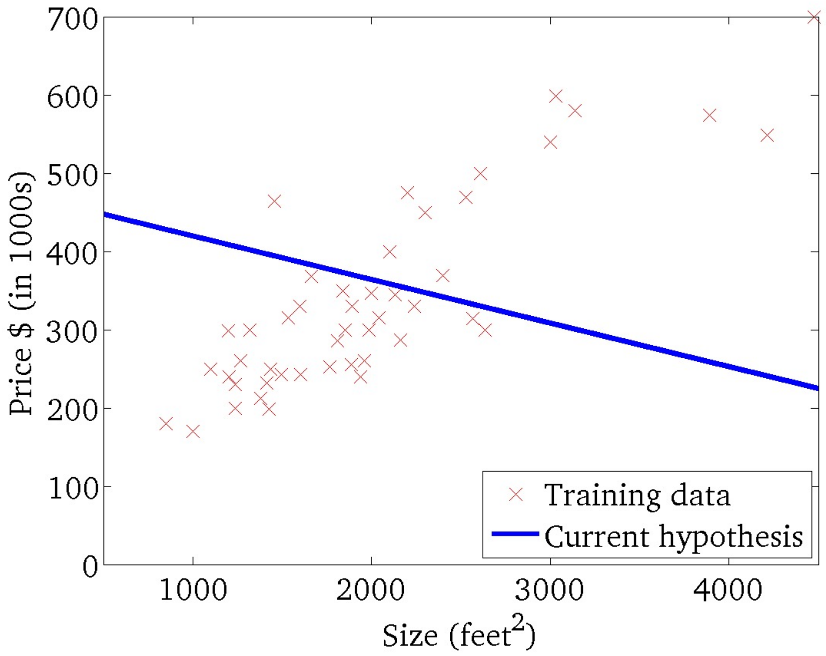


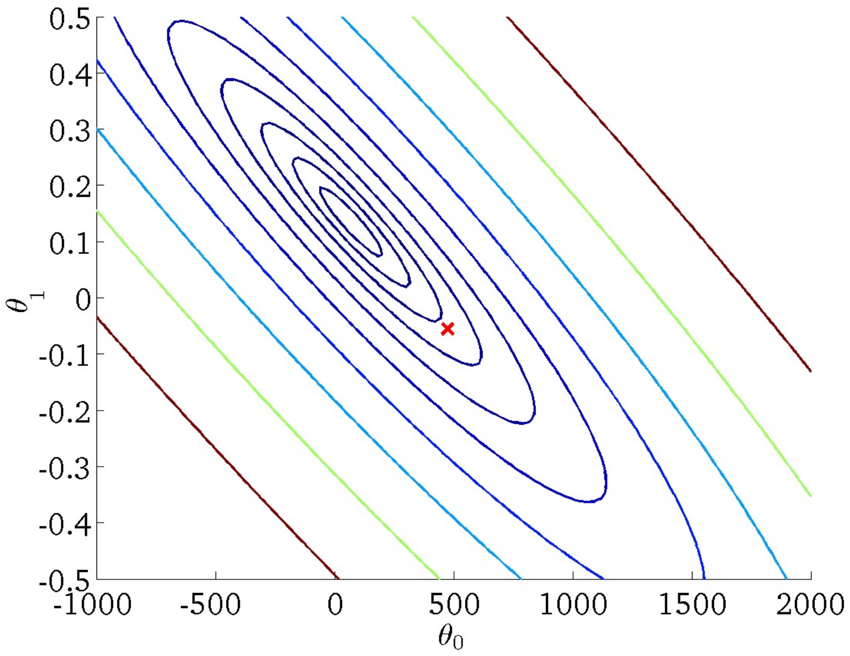
Gradient descent: Linear



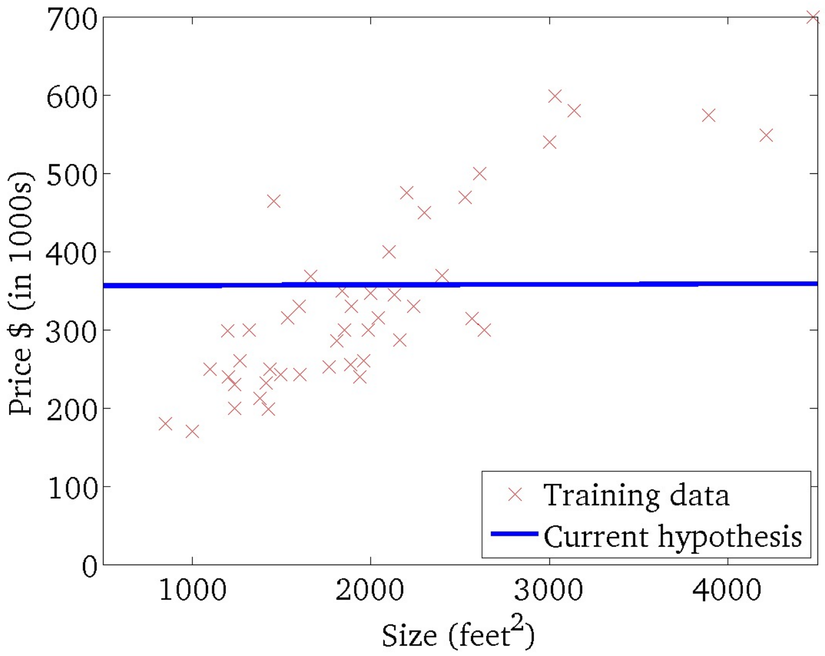


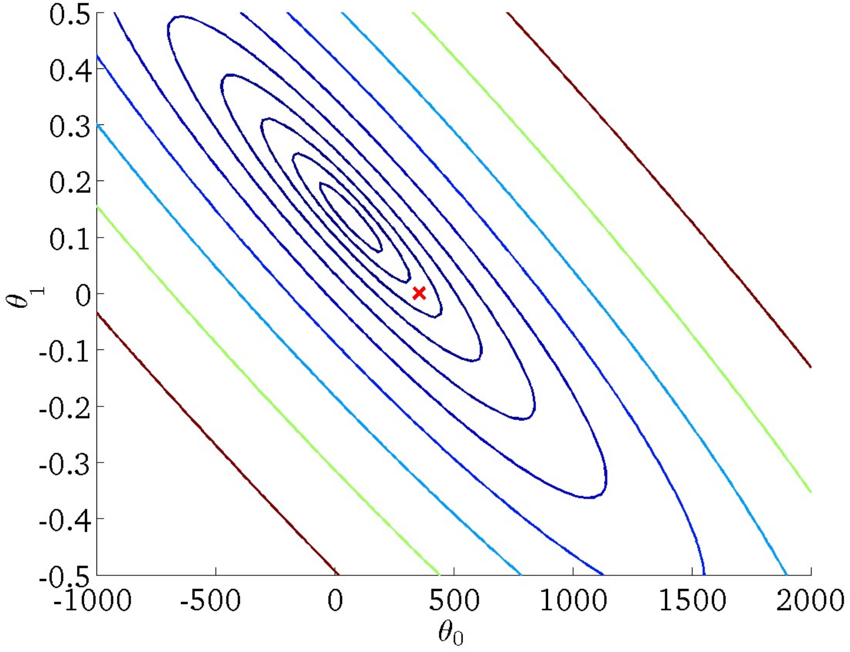
Gradient descent: Linear



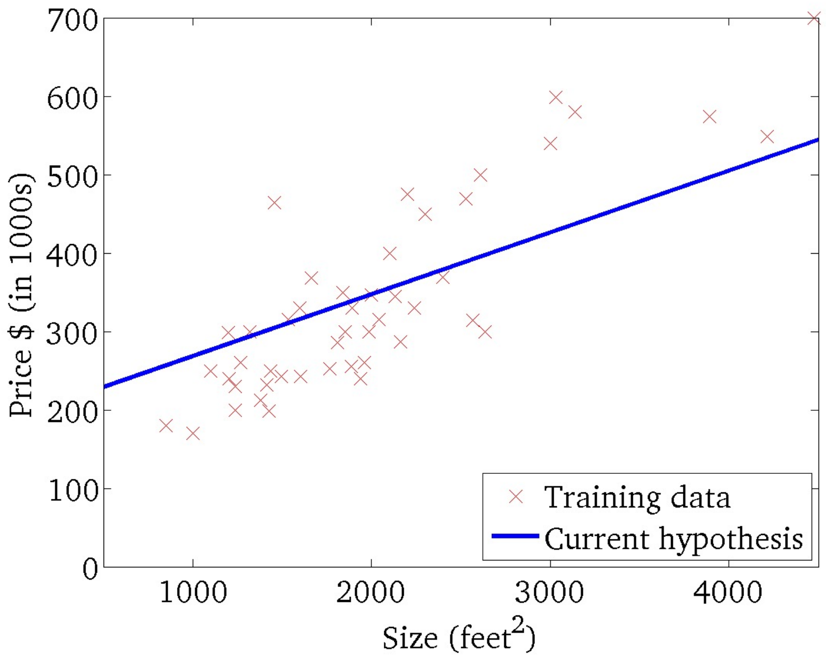


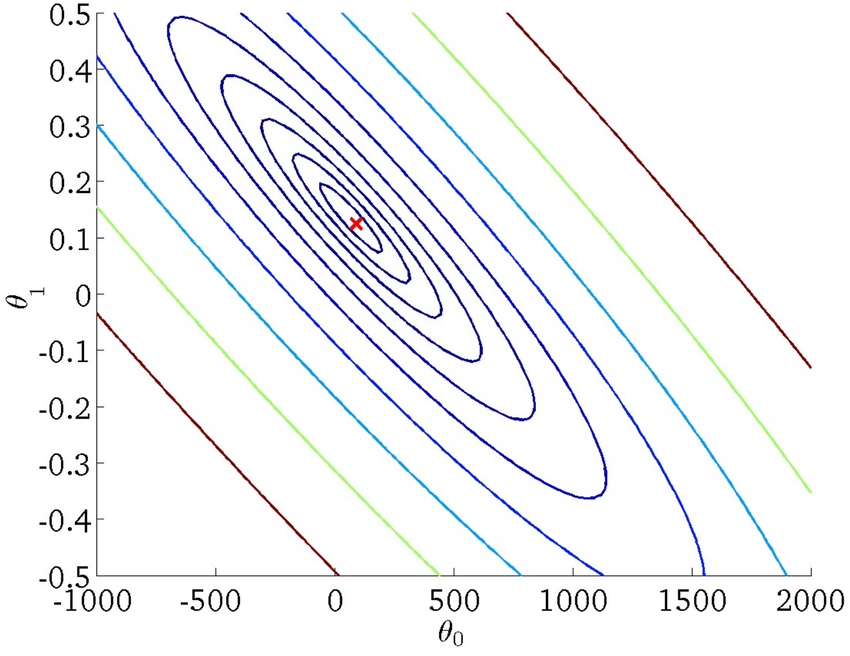
Gradient descent: Linear





Gradient descent: Linear





LMS algorithm: update rule

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta) \longrightarrow \theta_j := \theta_j - \alpha \cdot (h_\theta(x) - y) x_j$$

Execise: Obtain te gradient for the quadratic cost function of:

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

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

This rule (also known as Widrow-Hoff learning rule) has several properties that seem natural and intuitive. For instance, the magnitude of the update is proportional to the error term $(y - h_\theta(x))$; thus, for instance, if we are encountering a training example on which our prediction nearly matches the actual value, then, we find that there is little need to change the parameters; in contrast, a larger change to the parameters will be made if our prediction has a large error.

Execise 2. Using the Gradient derived above, calculate the first iteration of the gradient descent

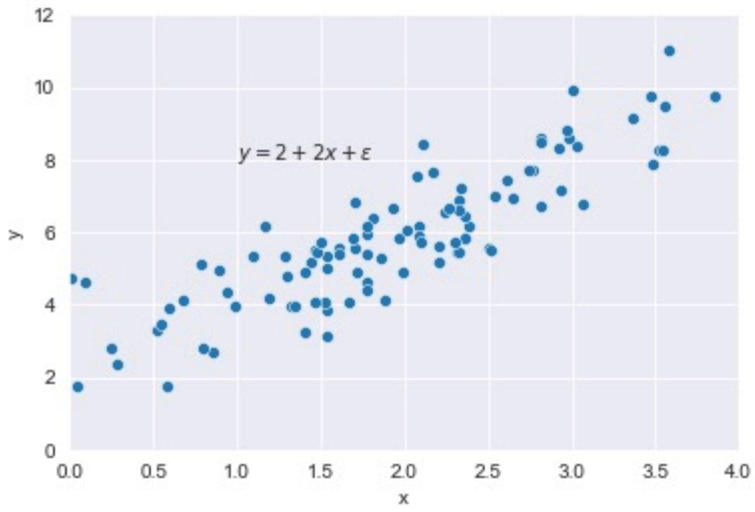
Initial Model

$$\hat{y} = \theta_0 + \theta_1 x$$

$$\theta_0 = 2$$
$$\theta_1 = 1$$
$$\alpha = 0.1$$

Grad Desc Algorithm

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



Sample	X_0	X_1	y_real	θ_0^j	θ_1^j	y_pred	error_term	Grad_ θ_0	Grad_ θ_1	Learn_rate	θ_0^{j+1}	θ_1^{j+1}
0	1.0	2.50	5.58	2	1							
1	1.0	1.86	5.30									
2	1.0	2.65	6.96									
3	1.0	3.52	8.24									
4	1.0	1.77	5.38									

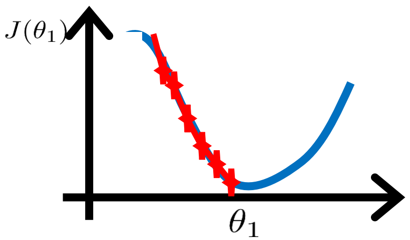
Learning rate

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

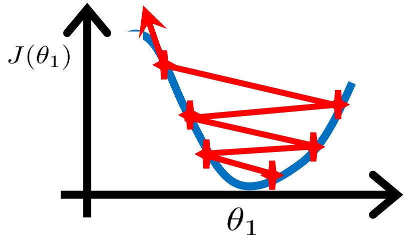
Learning rate

In Practice:

If a is too small, gradient descent can be slow.

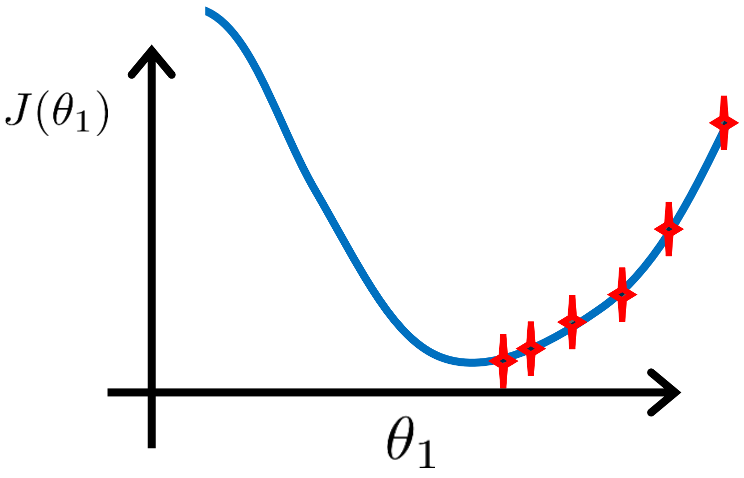


If a is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge



Gradient descent can be susceptible to **local minima** in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate a is not too large) to the global minimum.

As we approach a local minimum, gradient descent will automatically take smaller steps, being [0,1] smooths the derivative by reducing the jump stride.



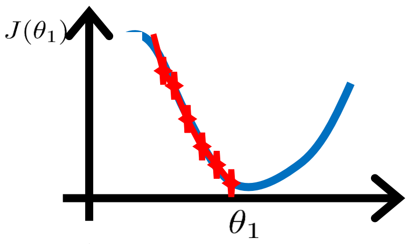
Learning rate

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

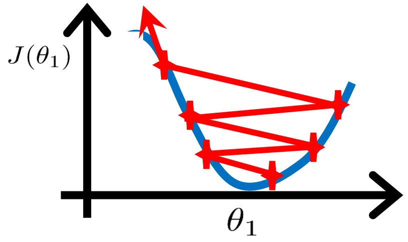
Learning rate

In Practice:

If a is too small, gradient descent can be slow.

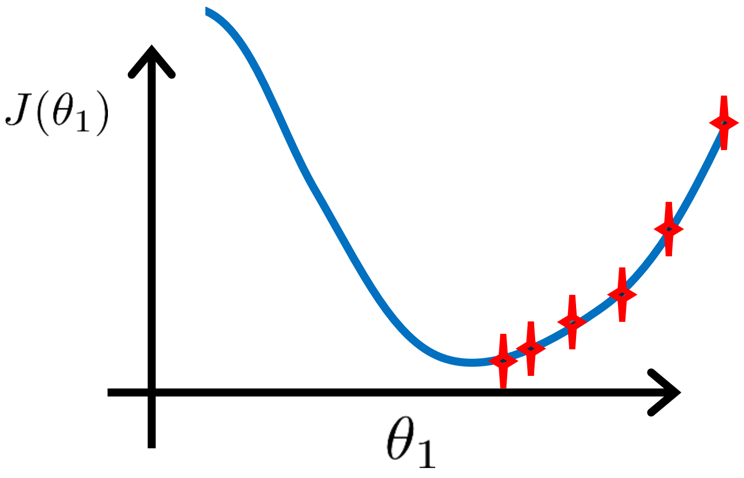


If a is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge



Gradient descent can be susceptible to **local minima** in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate a is not too large) to the global minimum.

As we approach a local minimum, gradient descent will automatically take smaller steps, being [0,1] smooths the derivative by reducing the jump stride.



LMS algorithm: incremental rule

Loop{

for i=1 to m, {

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

}

}

When the training set is large, **stochastic gradient descent** is often preferred over batch gradient descent

Whereas batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if m is large—stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at. Often, stochastic gradient descent gets θ "close" to the minimum much faster than batch gradient descent. (Note however that it may never "converge" to the minimum, and the parameters will keep oscillating around the global minimum; but in practice, most of the values near the minimum will be reasonably good approximations to the true minimum.

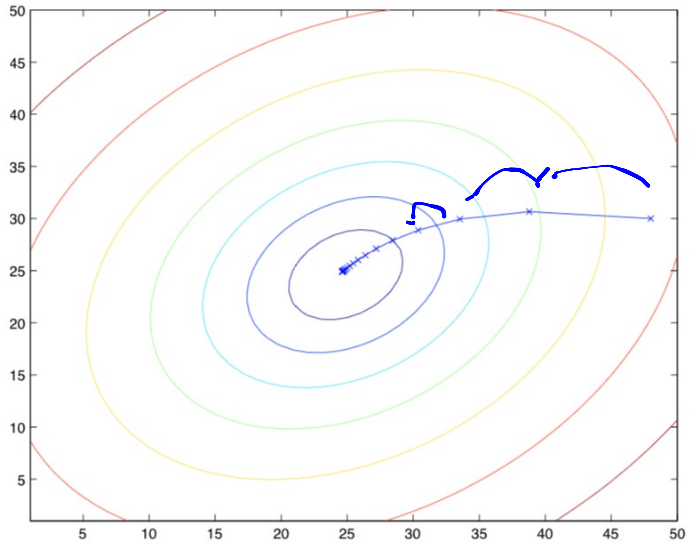
LMS algorithm: batch rule

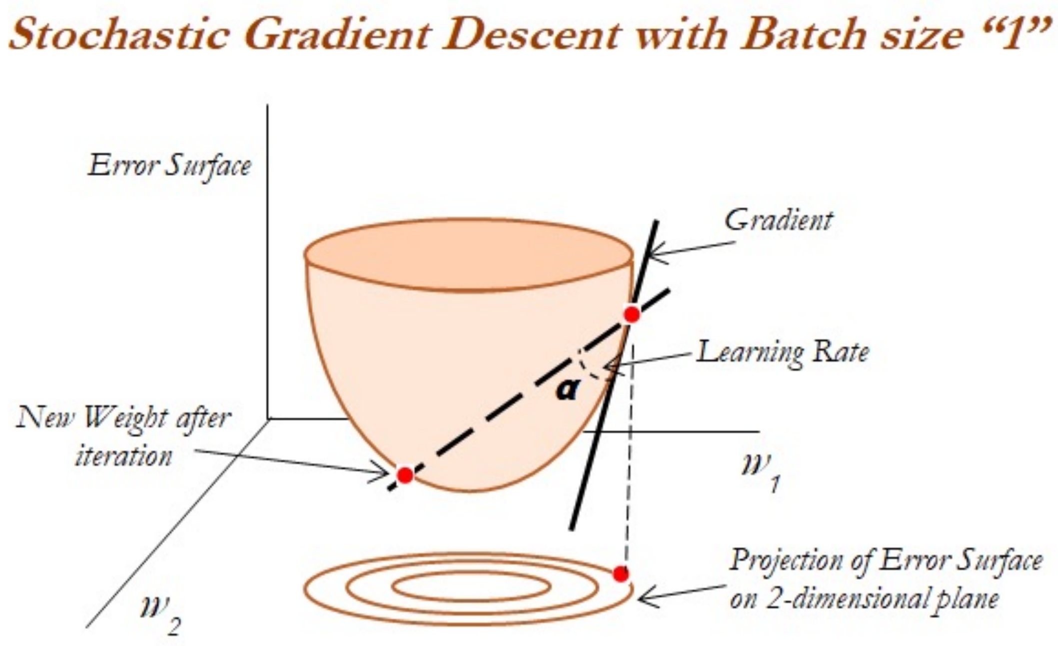
epoch

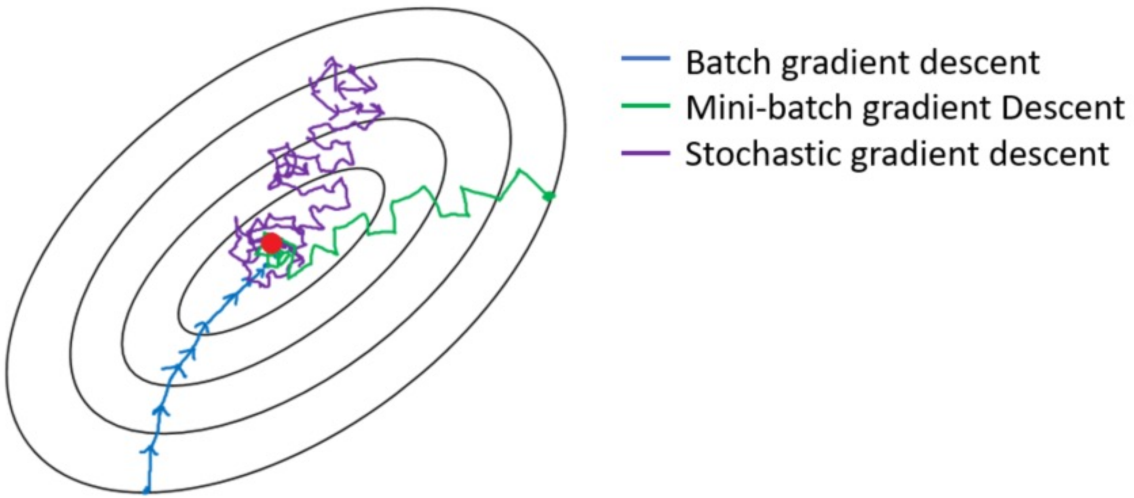
Repeat until convergence {

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

} (for every j)







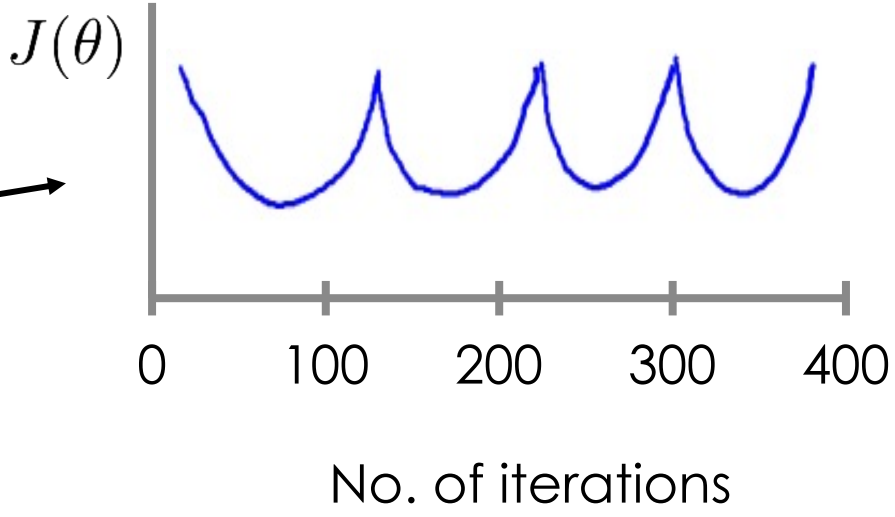
Gradient descent
minimize an objective function *J(θ)* by updating the parameters in the opposite direction of the gradient of the objective function.

The learning rate determines the size of the steps taken to reach the minimum

- Batch gradient descent: all training observations utilized in each iteration
- SGD: one observation per iteration
- Mini batch gradient descent: size of about 50 training observations for each iteration

LMS in practice

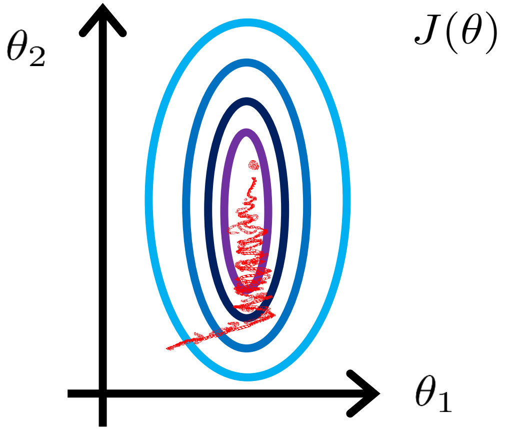
It is necessary to display the cost function

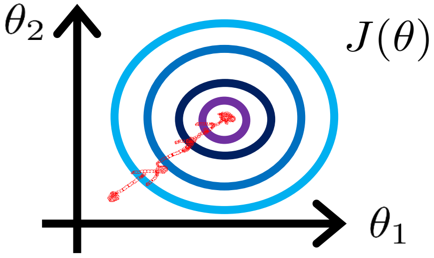


Simultaneous update

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

Data scaling / normalization





Summary of Linear models

Linear regression

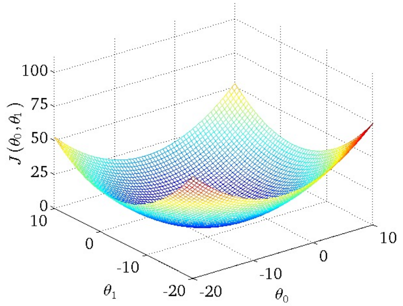
Hypothesis:

the model is linear

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

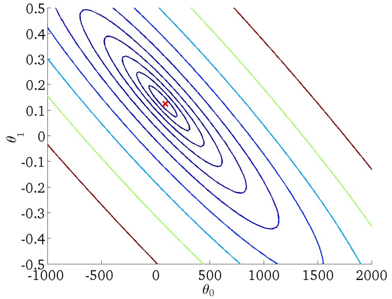
Gradient descent

Idea: *to make h_θ close to y by means minimizing a cost function*



LMS

Using the gradient to find the minimum (batch & incremental)



Linear regression revisited

Hypothesis: the model is linear

n = number of input dimensions

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

Generalization

(in matrix notation)

Case n=1:

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

x_0=1 (Intercept term)

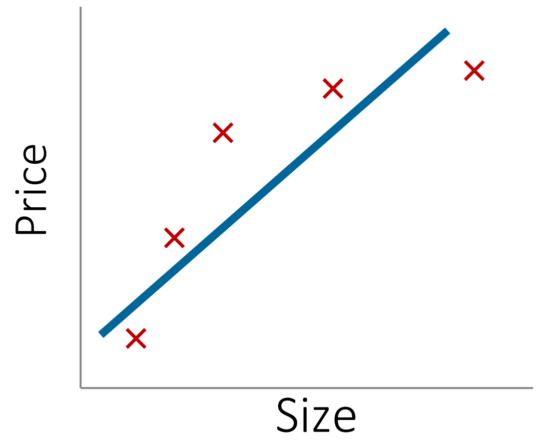
$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

Bias-Variance Tradeoff

A learning algorithm is **biased** for a particular input (x) if, when trained on different data sets, it is systematically incorrect when predicting the correct output for x.

Therefore, the **bias** is an error from erroneous assumptions in the learning algorithm.

High bias can cause an algorithm to miss the relevant relations between features and target outputs (**underfitting**).



$$\theta_0 + \theta_1 x$$

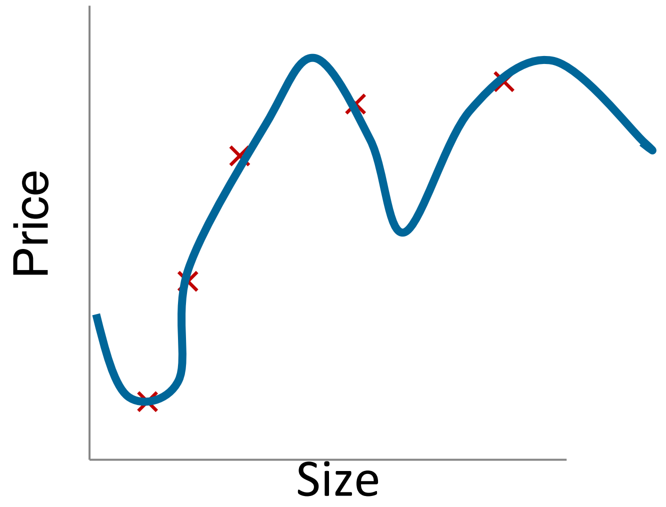
Bias
(Underfitting)

Bias-**Variance** Tradeoff

A learning algorithm has high **variance** for a particular input x if it predicts different output values when trained on different training sets.

The **variance** is an error from sensitivity to small fluctuations in the training set.

High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (**overfitting**).



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

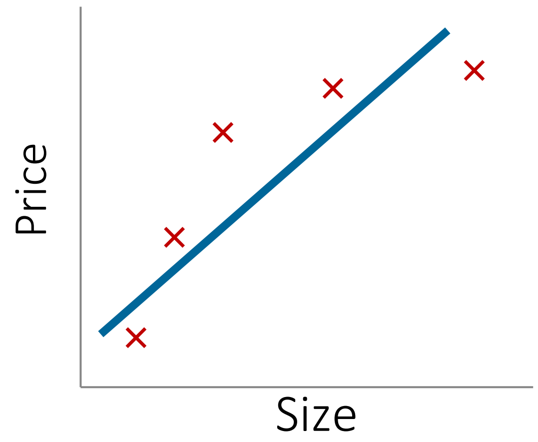
Variance
(Overfitting)

Bias-Variance Tradeoff

A learning algorithm is **biased** for a particular input (x) if, when trained on different data sets, it is systematically incorrect when predicting the correct output for x.

Therefore, the **bias** is an error from erroneous assumptions in the learning algorithm.

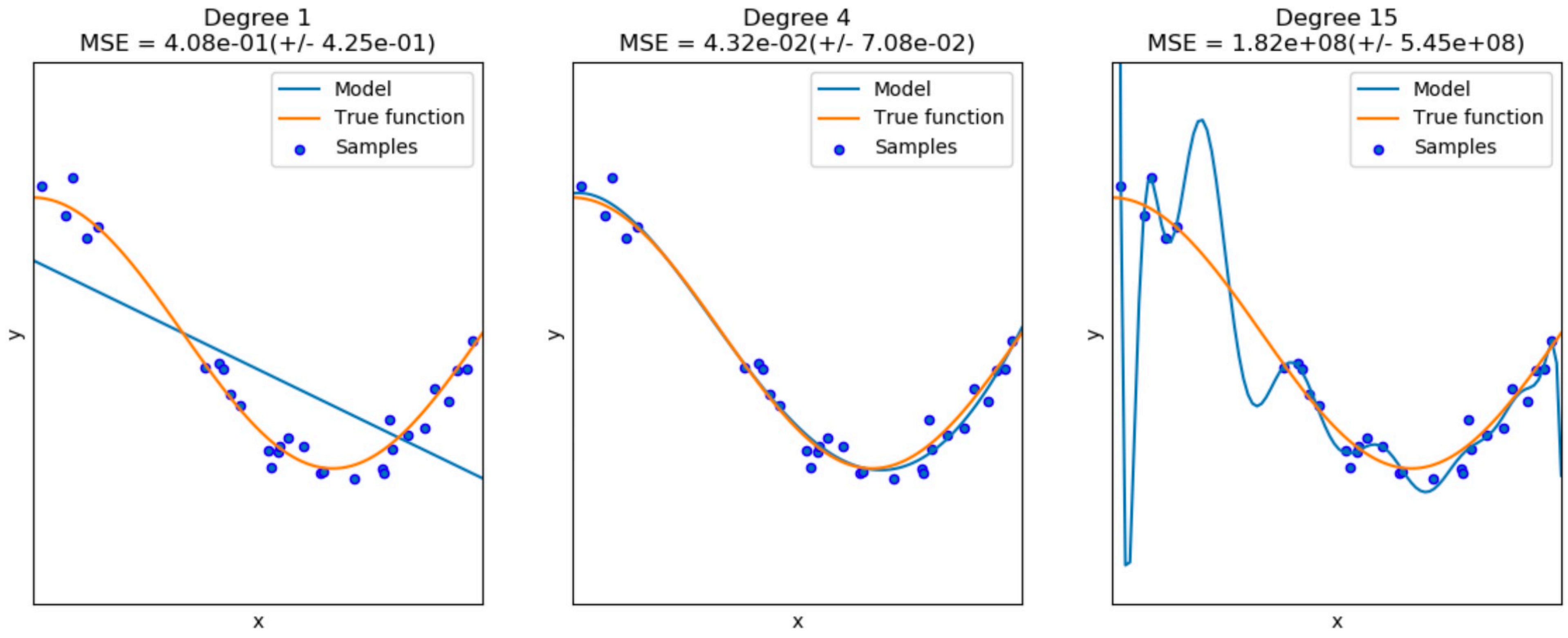
High bias can cause an algorithm to miss the relevant relations between features and target outputs (**underfitting**).



$$\theta_0 + \theta_1 x$$

Bias
(Underfitting)

Bias-Variance Tradeoff



Bias-Variance Tradeoff

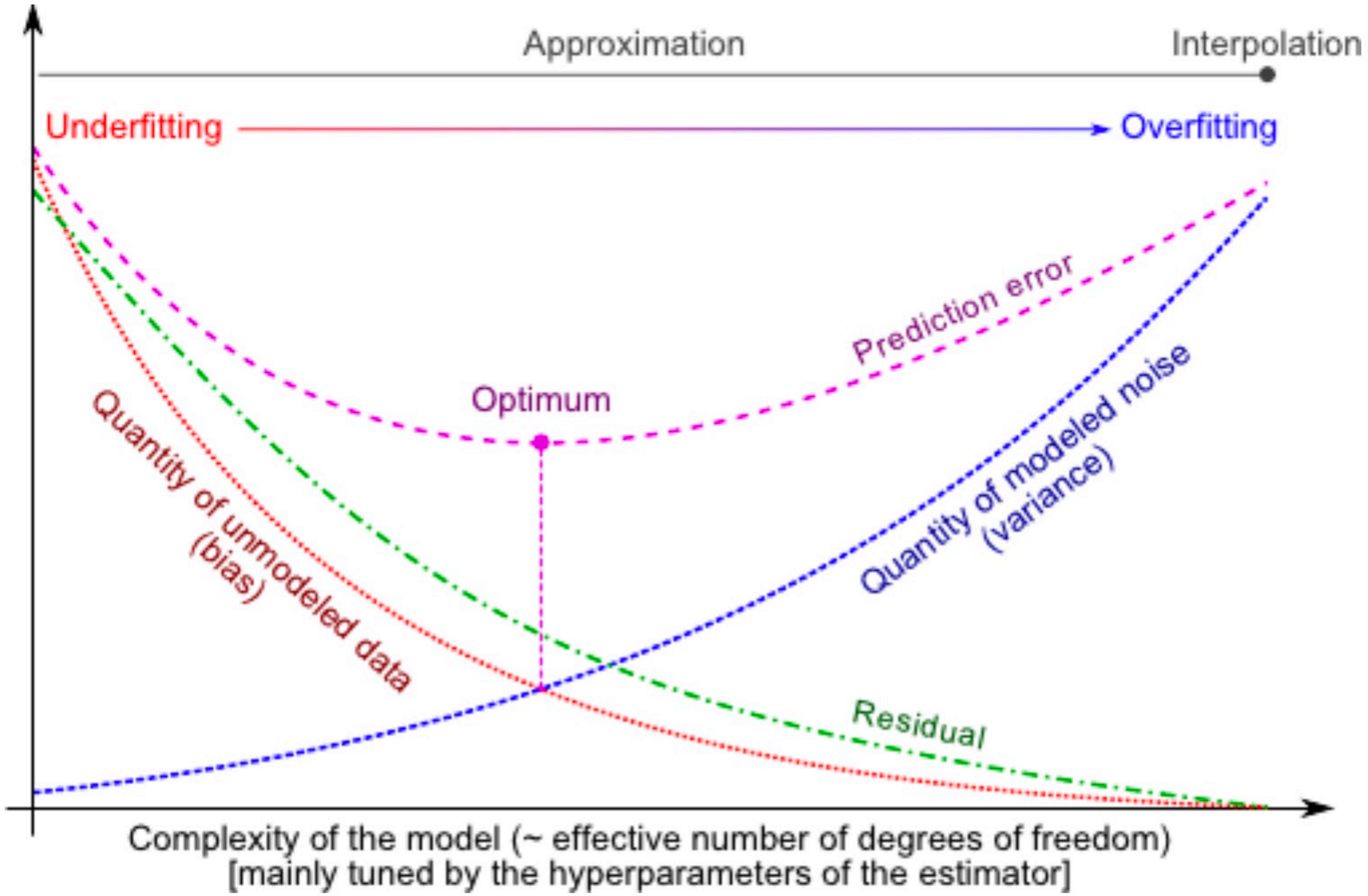
$$Err(x) = \mathbb{E}\left[\left(y_{test} - \hat{y}\right)^2\right] = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]$$
$$+ \sigma_e^2$$

Bies: La diferencia entre el valor predicho por el modelo y el valor real.

Varianza: El error producido debido a la sensibilidad del modelo con respecto a los datos de entreno.

Error irreductible: ruido que no conseguiremos modelizar

Bias-Variance Tradeoff



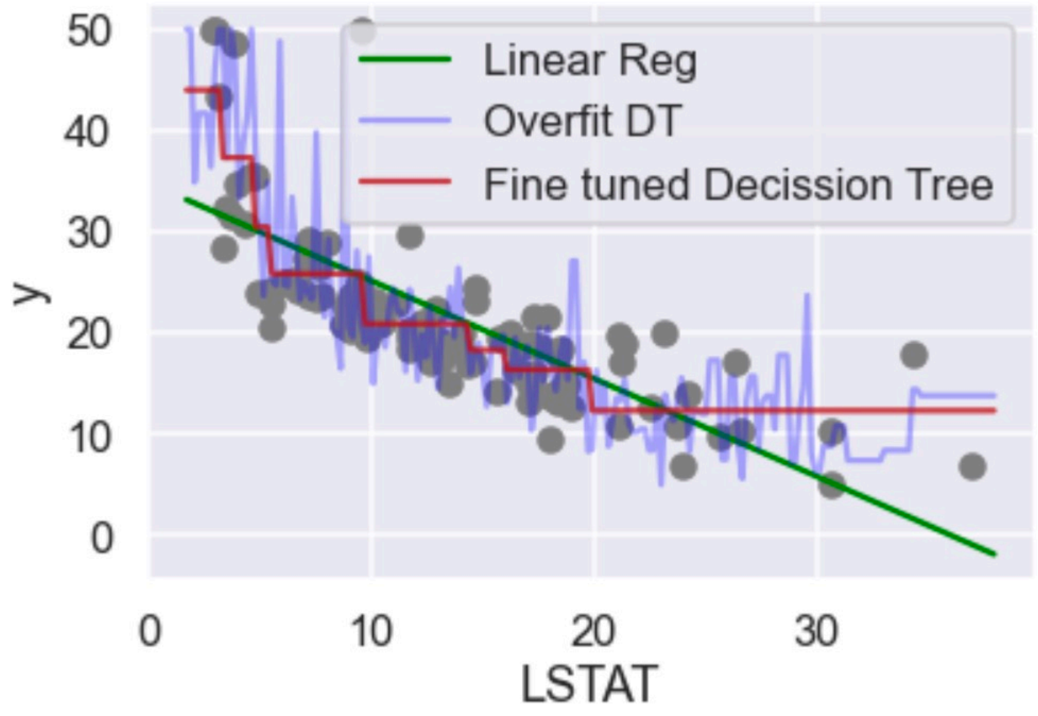
Bias-Variance Tradeoff

Alto bies: Simplificaciones sobre la forma de los datos, *menos flexibles*: Linear models

Baja Varianza: Pequeños cambios en las predicciones: Linear models

Bajo bies: Pocos supuestos sobre la forma de los datos, *más flexibles*: Decission trees, knn, ...

Alta Varianza: Grandes cambios en las prediccinoes: Decission trees, knn, ...



Bias-Variance Tradeoff

La única forma de comprobar que nuestro modelo generaliza bien es guardando una parte de los datos para medir el error *out-of-sample*, es decir, el error sobre los datos que no ha visto el modelo durante la fase de entreno.

Con tal de poder elegir el mejor modelo y sus parametros se dividirá inicialmente el dataset en dos subsets:

•subset de entrenamiento: normalmente se utiliza el 80% del tamaño de la muestra

•subset de testeo: el 20% restante.