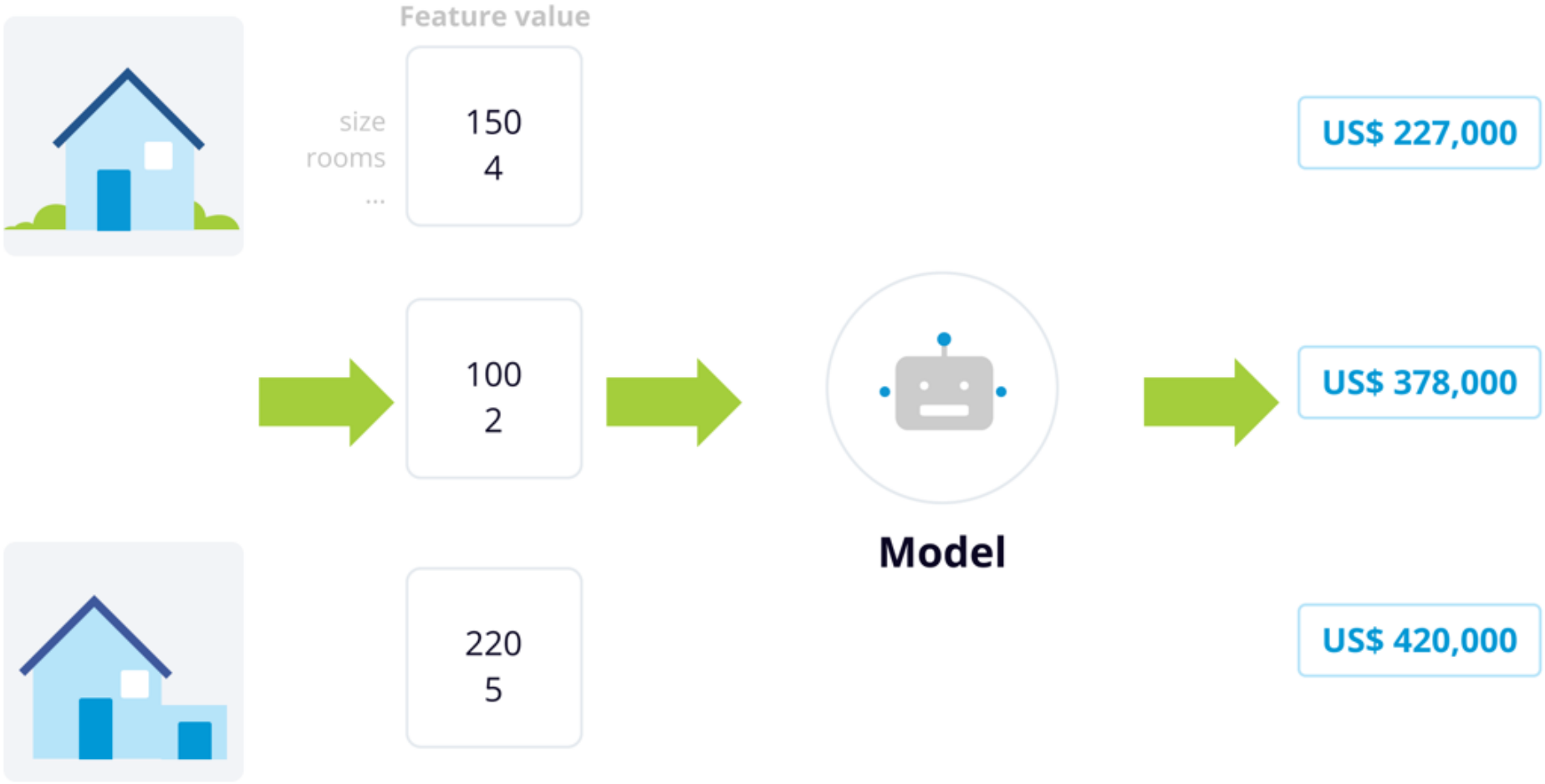
Linear Regression

Introduction

Example of a house price prediction problem.

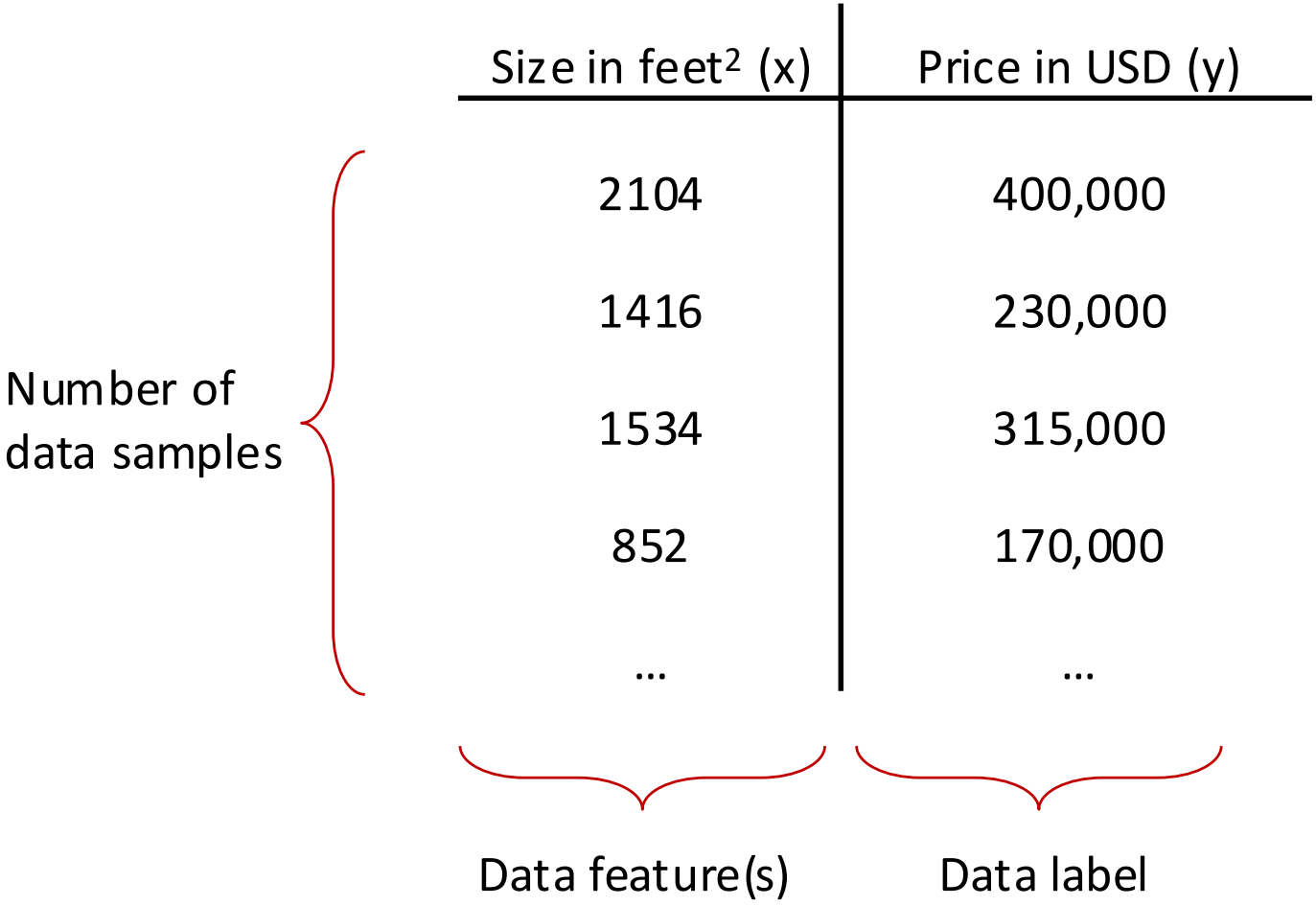
Introduction

Example of a house price prediction problem.



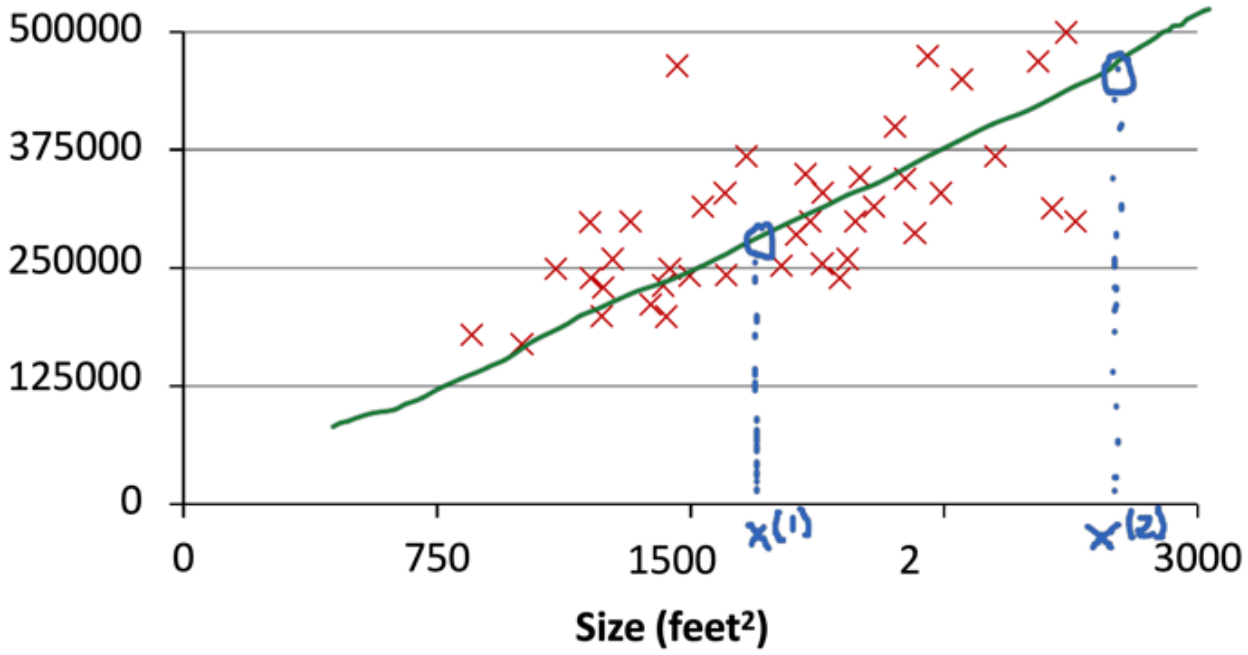
Problem Setup

Example of a dataset about house pricing.



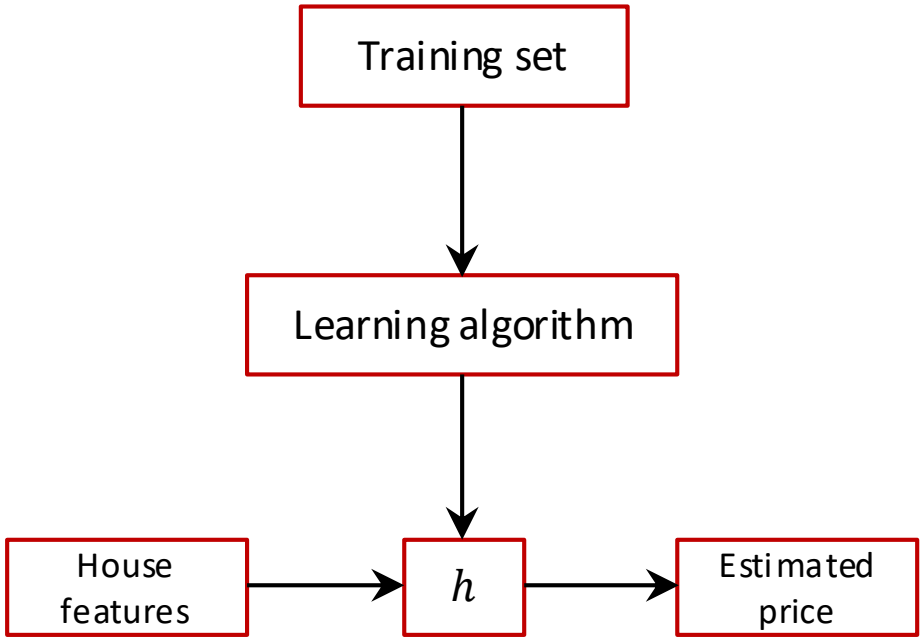
Problem Setup

Objective: find a line (i.e., a function) that can learn the relation between data features and data labels.



Question: What desired properties should the fitting function possess?

Problem Setup



How do we represent h?

- We represent hypothesis about the data using parameters $\theta = (\theta_0, \theta_1)$.
- If the data is correctly predicted according to the hypothesis h_θ, then $y \approx h_\theta(x) = \theta_0 + \theta_1 x$
- The learning algorithm finds the best hypothesis h_θ for the training set.
- We can then estimate the values of y for the test set using that h_θ
- If $h_\theta(x)$ is a linear function of a real number x of , this procedure is called linear regression.

Problem Setup

Size in feet^2 (x)	Price in USD (y)
2104	400,000
1416	230,000
1534	315,000
852	170,000
…	…

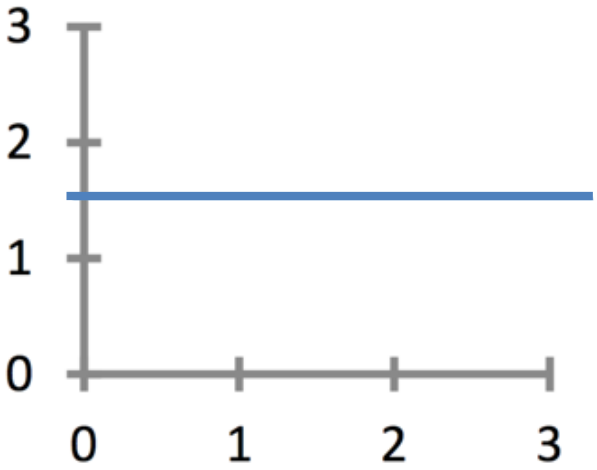
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$
where θ_0, θ_1 are learnable parameters.

Questions:

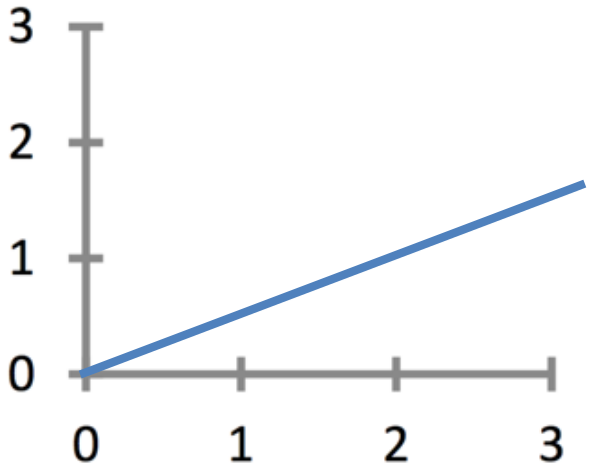
- What is the role of θ_0 and θ_1?
- How to choose good values of θ_0 and θ_1?

Observation

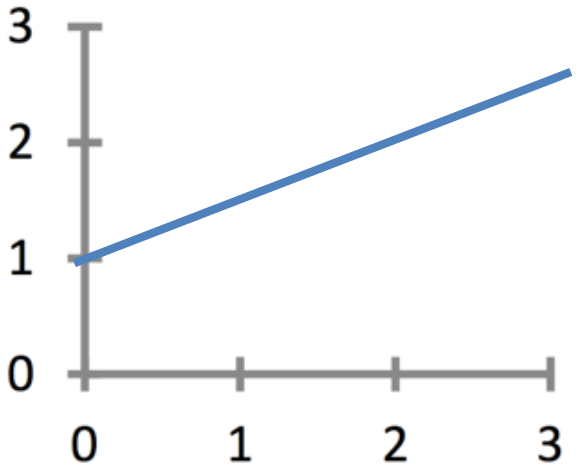
Fitting function $h_\theta(x) = \theta_0 + \theta_1 x$



$\theta_0 = 1.5$
$\theta_1 = 0$



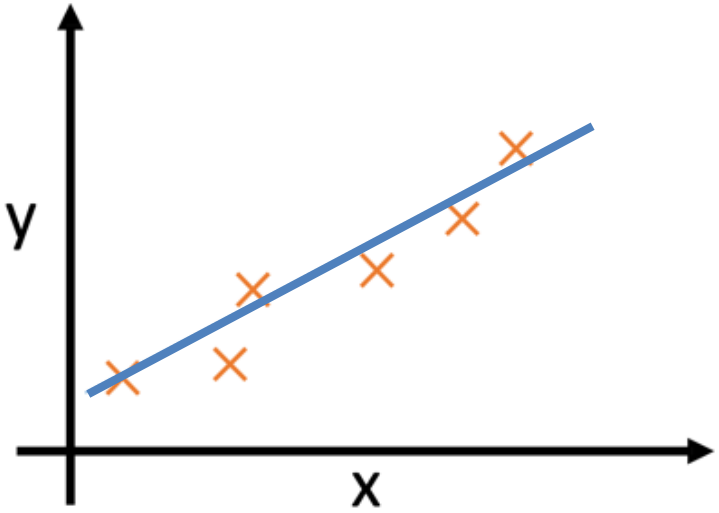
$\theta_0 = 0$
$\theta_1 = 0.5$



$\theta_0 = 1$
$\theta_1 = 0.5$

Visualization of the fitting function with some random values of θ_0 and θ_1

Observation

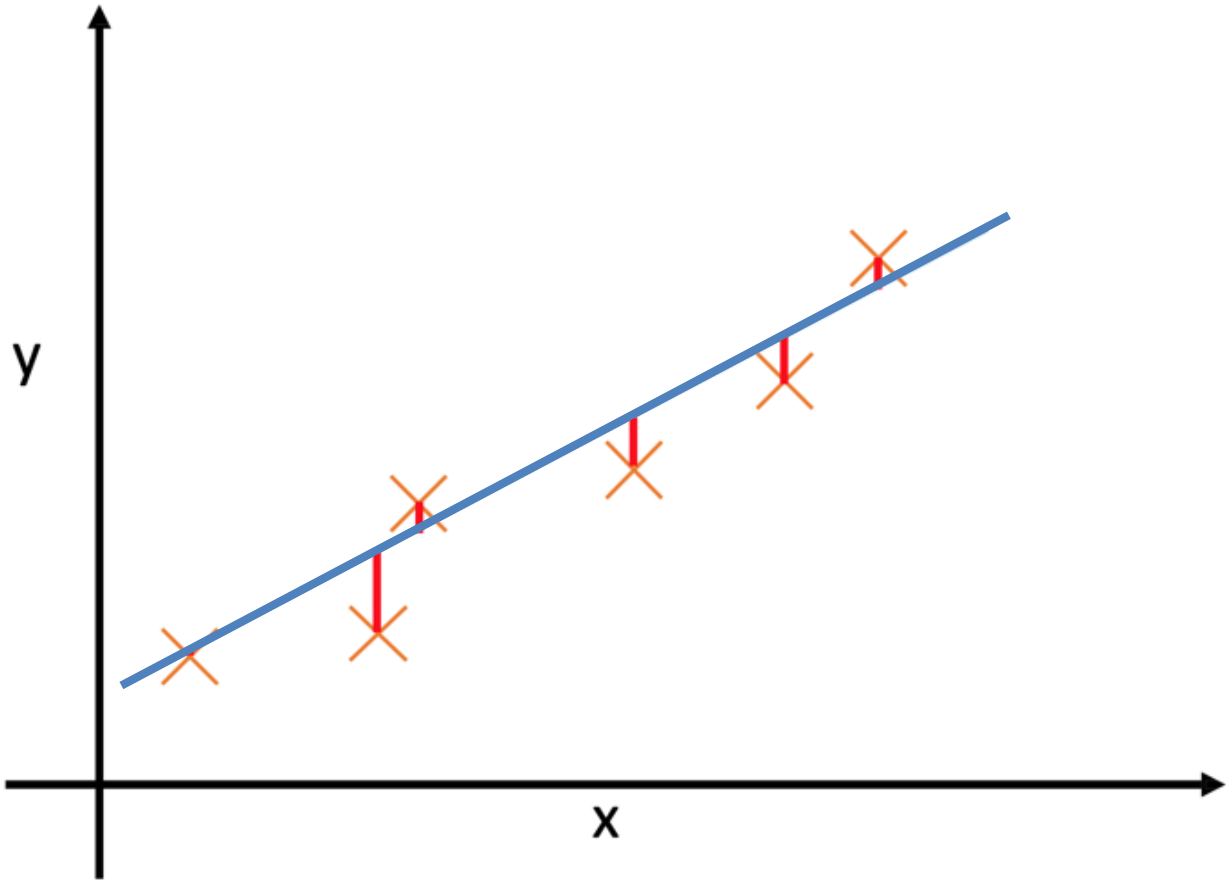


Question: How to choose good values of θ_0 and θ_1?

- **Answer**: Good values of θ_0 and θ_1 make $h_\theta(x)$ **close** to y for training samples (x, y)

Follow-up question: what does **close** mean?

Observation



Mean Squares Error (MSE) is used to estimate of how well the model fits the data.

MSE is the **average** of sum of all squared errors .

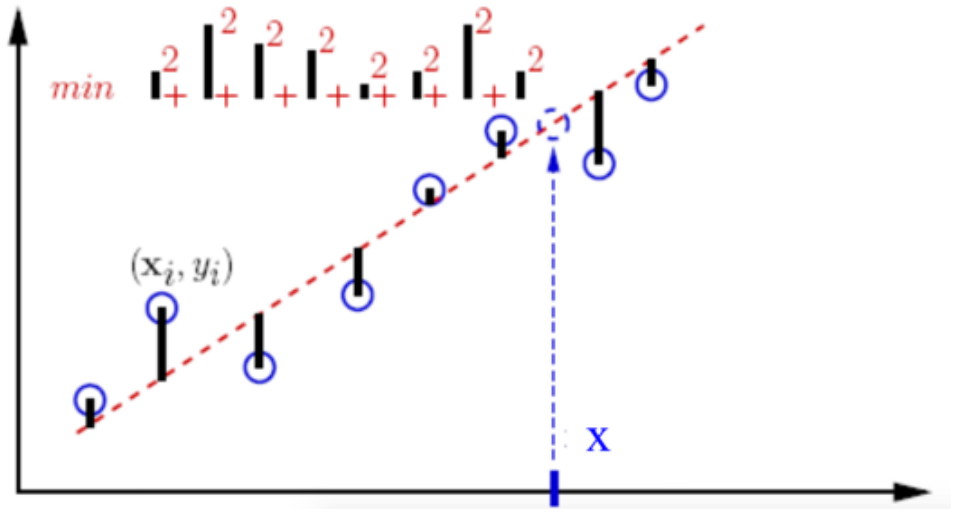
Linear Regression

Generalization of the fitting function:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_d x_d = \sum_{j=0}^{d} \theta_j x_j$$

Data feature is d-dimensional, i.e., $x_1, \ldots, x_d$

We fit the model by minimizing the MSE error.

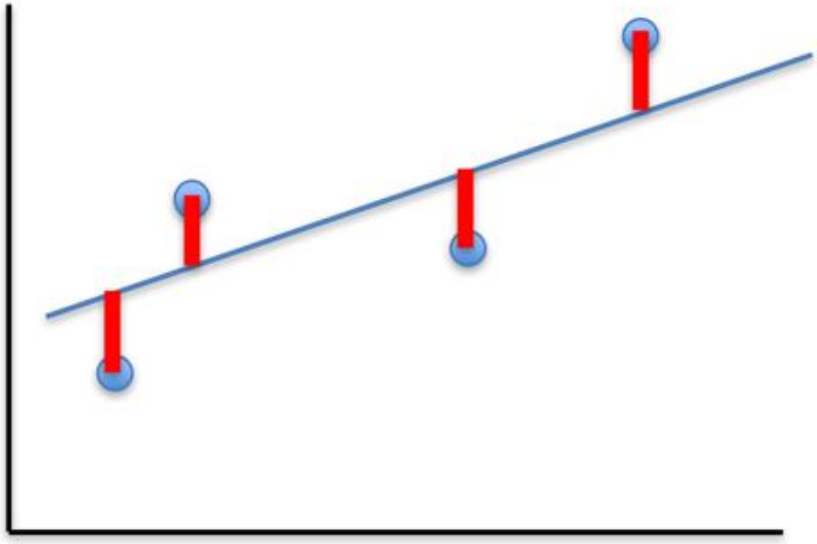


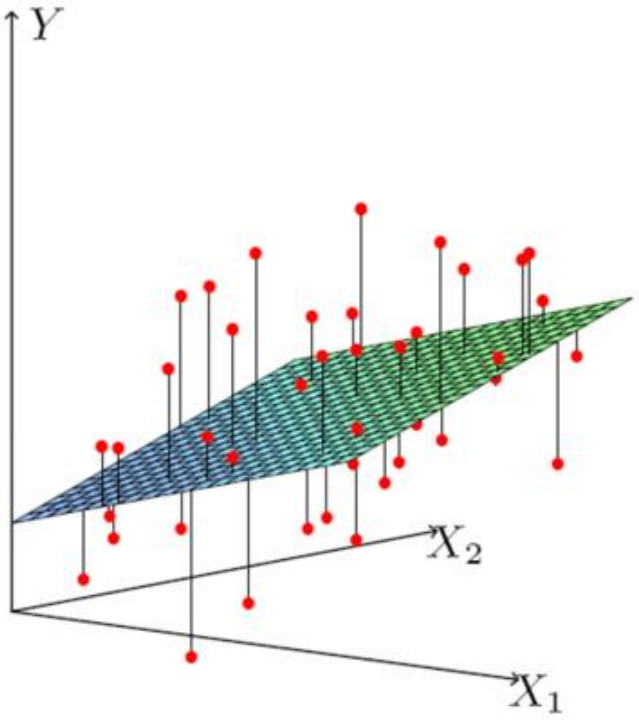
Least Squares Linear Regression

Loss Function

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

Fit by solving $\min_\theta J(\theta)$





Intuition Behind Loss Function

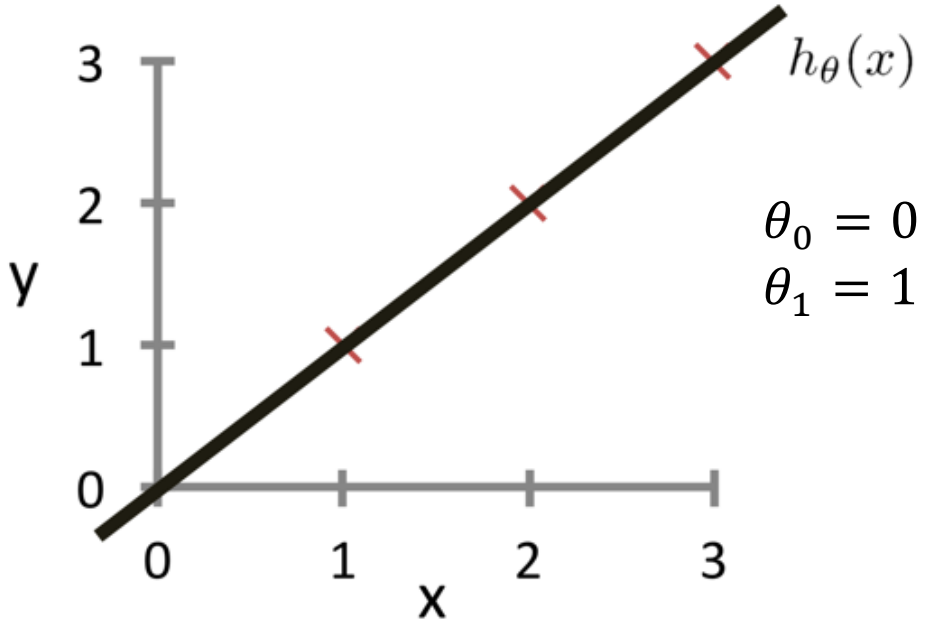
Loss Function

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

For insight on $J(\theta)$, let's assume that $\theta = [\theta_0, \theta_1]$

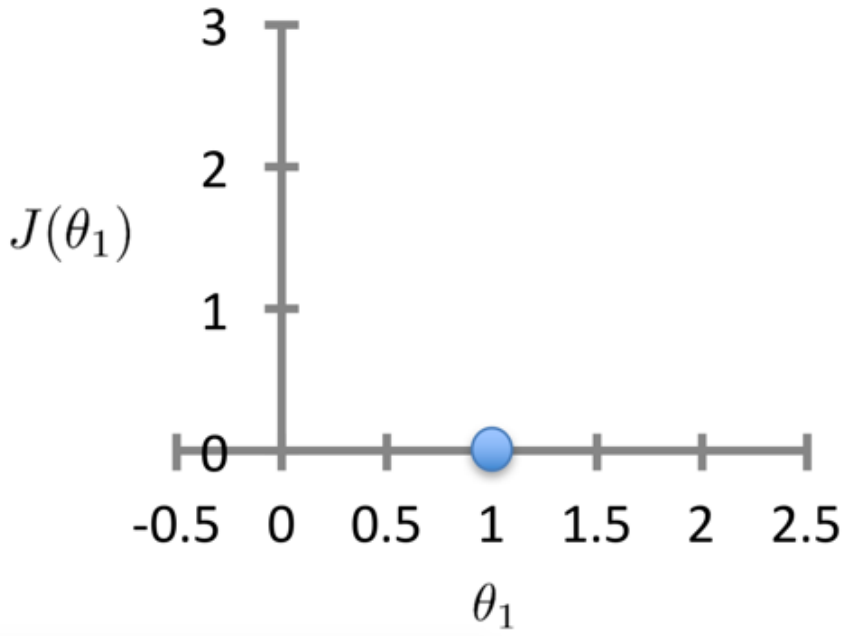
$h_\theta(x)$

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



$J(\theta_1)$

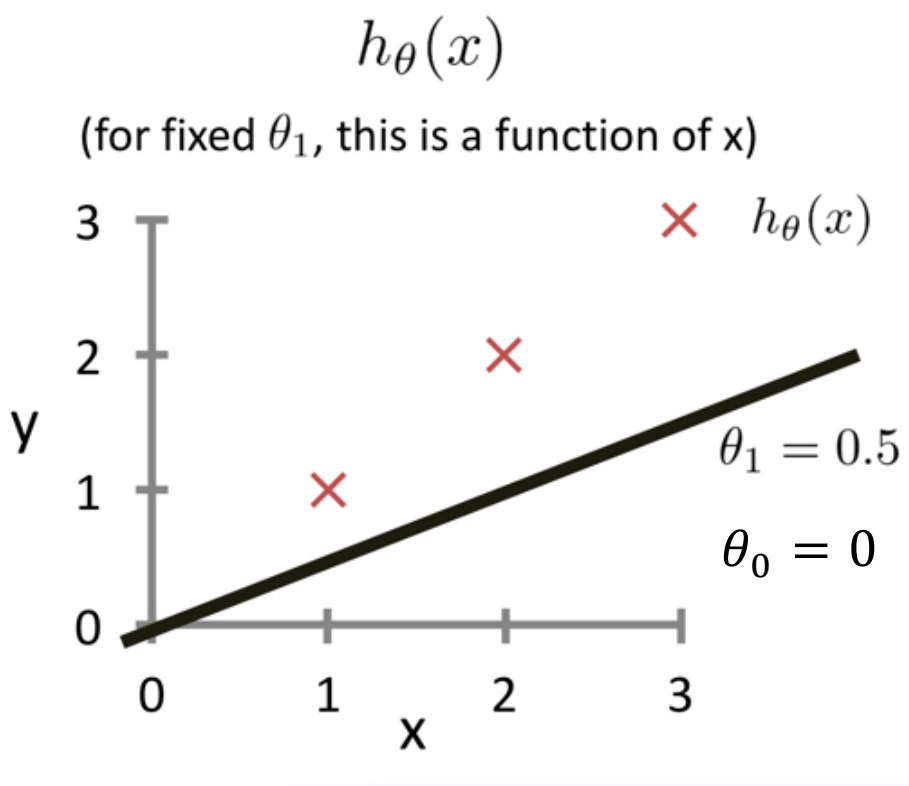
(function of the parameter θ_1)

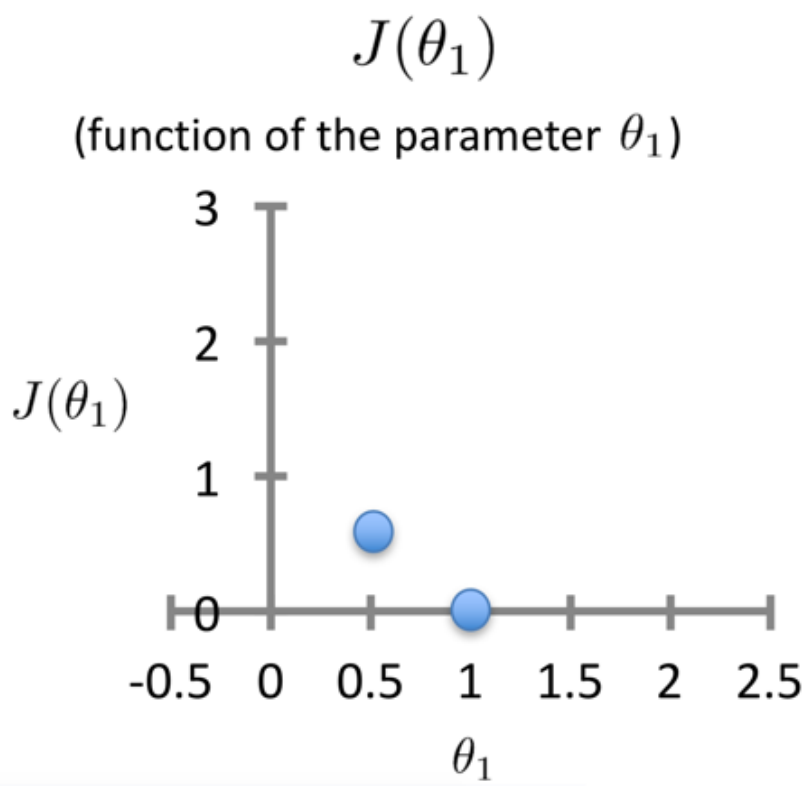


Intuition Behind Loss Function

Loss Function

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



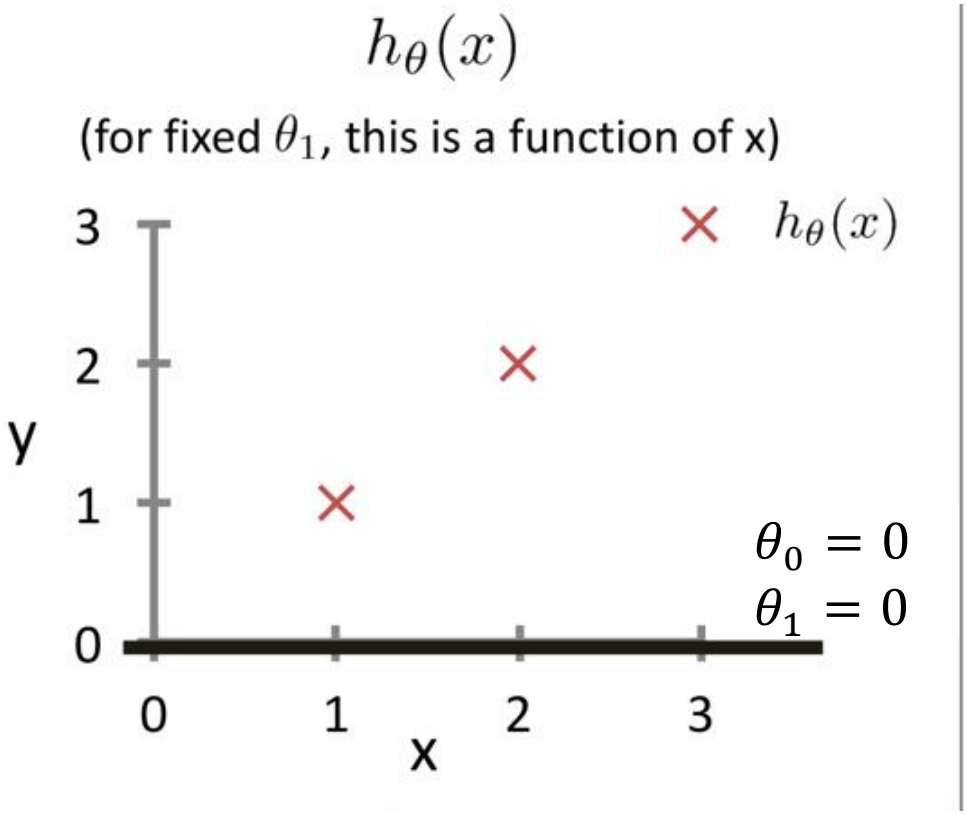


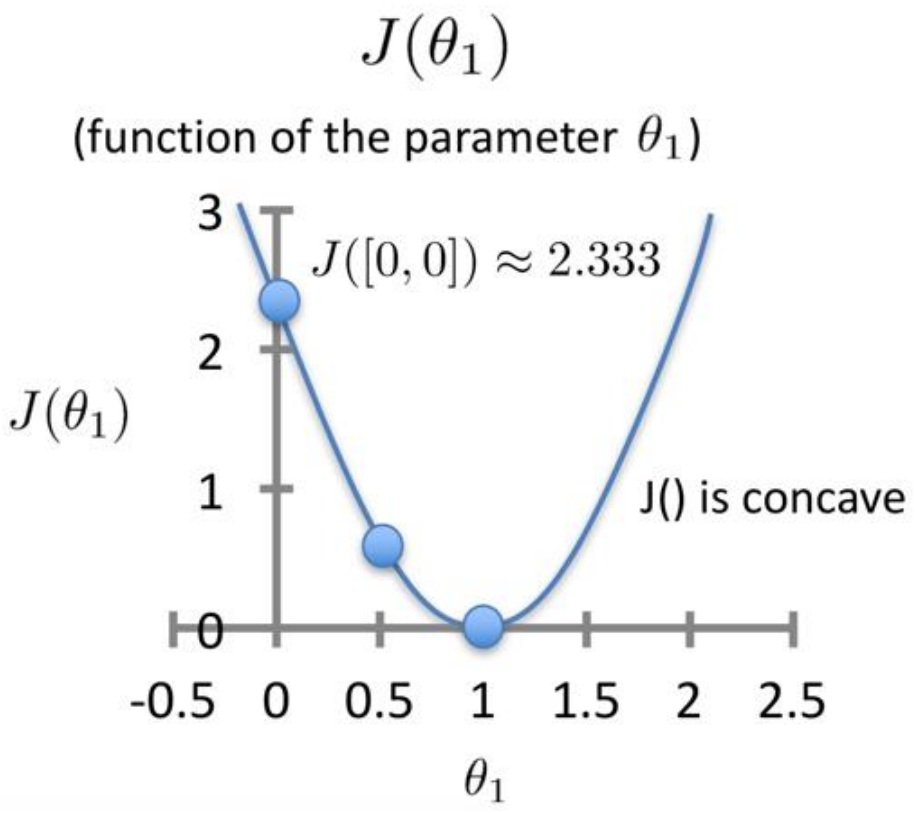
$$J[0,0.5] = \frac{1}{2\times 3}[(0.5-1)^2+(1-2)^2+(1.5-3)^2] \approx 0.58$$

Intuition Behind Loss Function

Loss Function

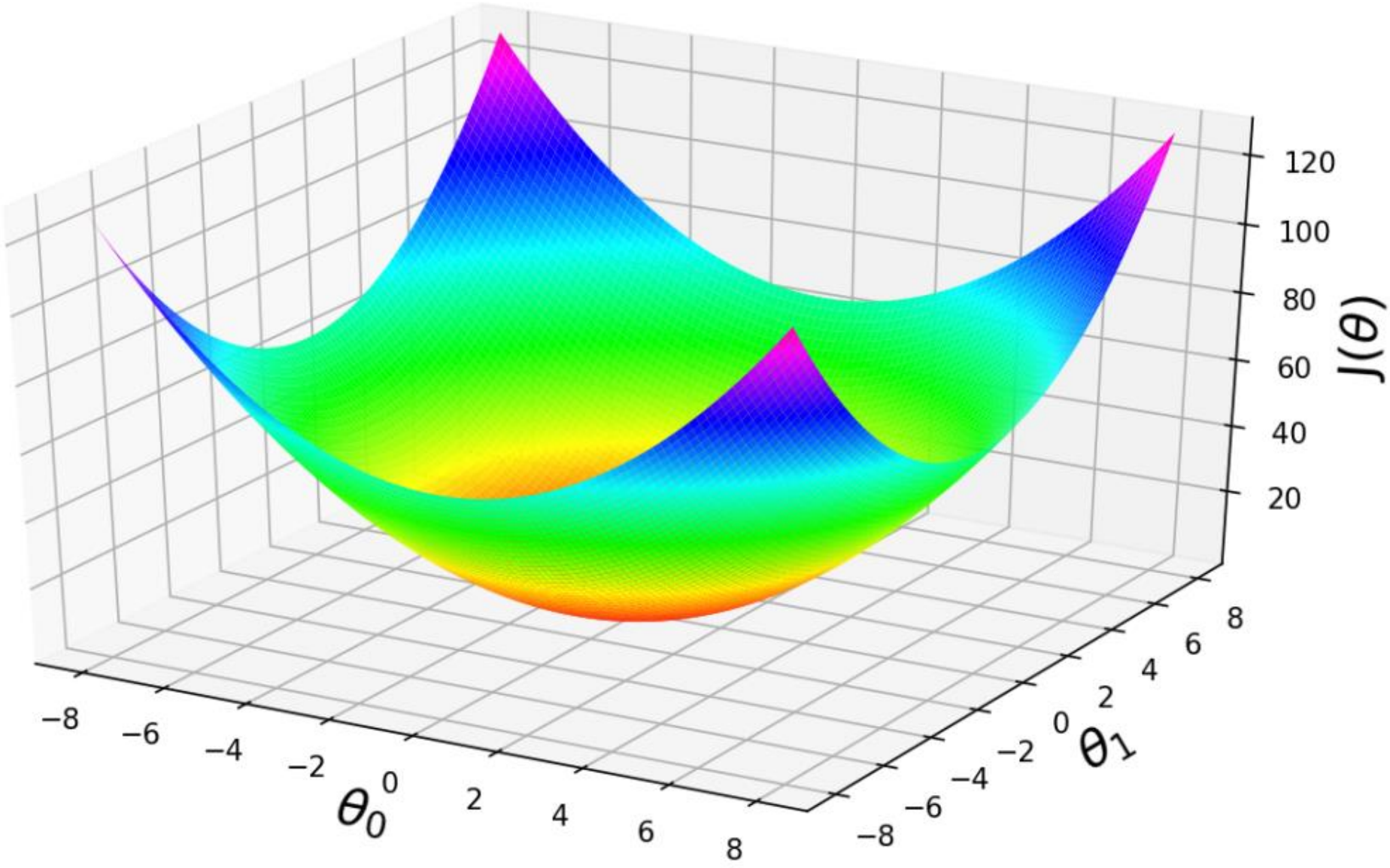
$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$





$$J[0,0] = \frac{1}{2\times 3}[(0-1)^2+(0-2)^2+(0-3)^2] \approx 2.333$$

Intuition Behind Loss Function



Formulation

- Hypothesis:

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

- Parameters:

$$\theta_0, \theta_1$$

- Loss Function:

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

- Goal:

$$\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$$

Formulation

Have some function $J(\theta_0, \theta_1)$ and we want $\underset{\theta_0,\theta_1}{\text{minimize}} J(\theta_0, \theta_1)$

Solution:

- Start with some θ_0, θ_1.
- Keep adjusting θ_0, θ_1 to reduce until we hopefully end up at a minimum $J(\theta_0, \theta_1)$

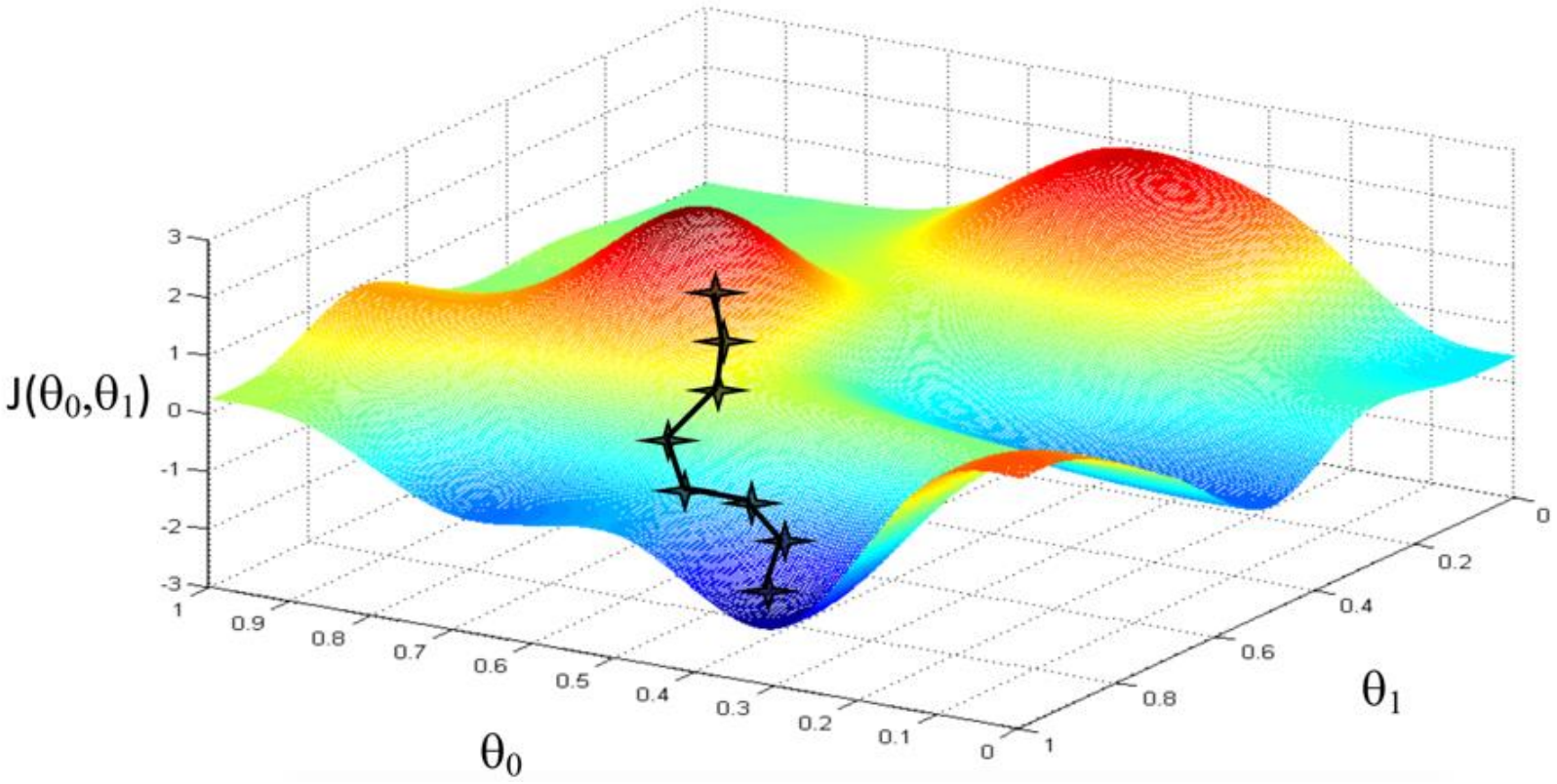
Question: how to properly adjust θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$?

Formulation

Question: how to properly adjust θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$?

Idea: Starting from an initial value, the optimizer aims to gradually move in the direction that leads a to a better (lower) value of $J(\theta_0, \theta_1)$.

→ Gradient descent



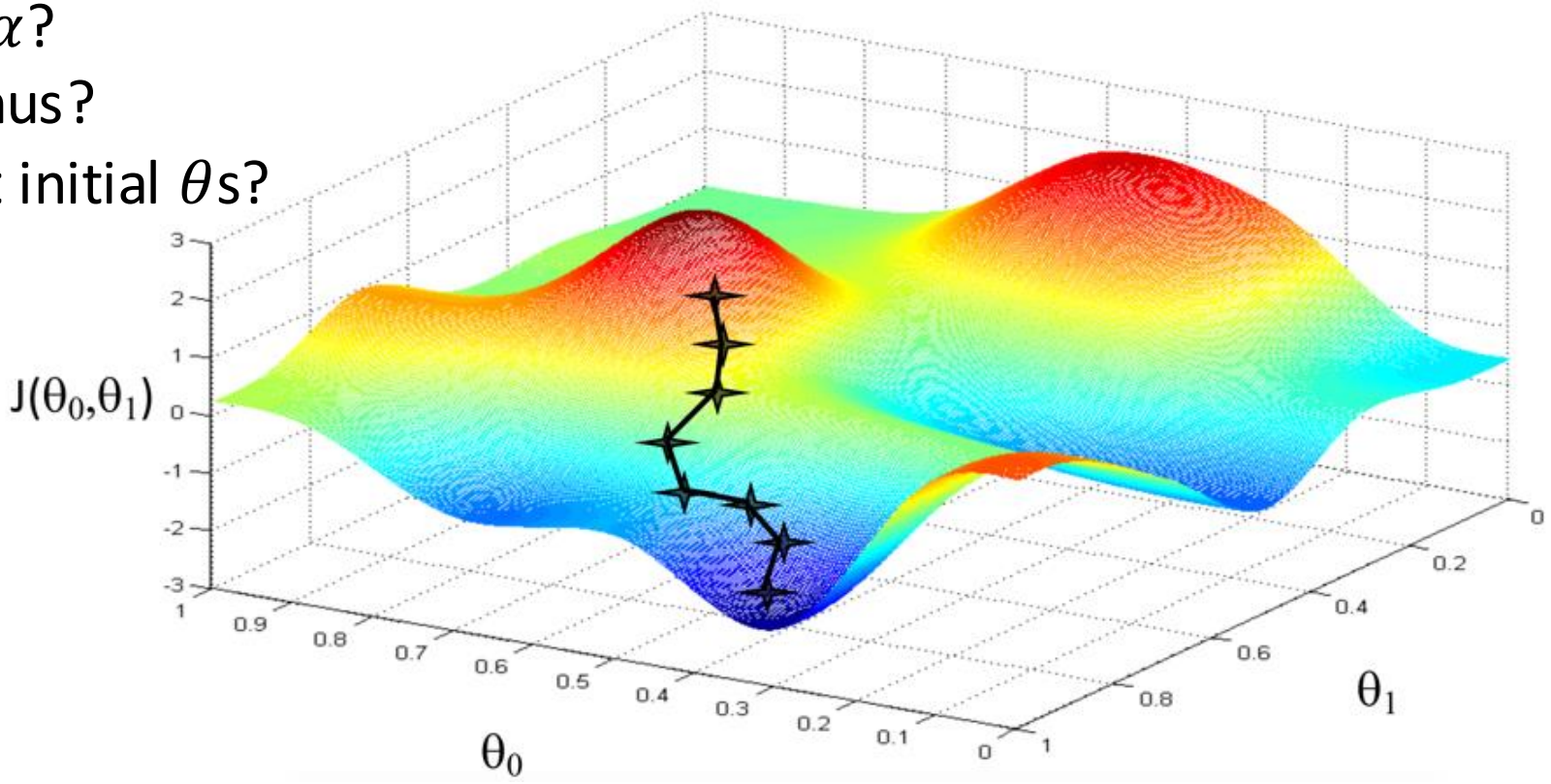
Gradient descent

- Choose initial value for θ.
- Until we reach the minimum, update θ to reduce $J(\theta)$:

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

Questions:

- What is α?
- Why minus?
- Different initial θs?



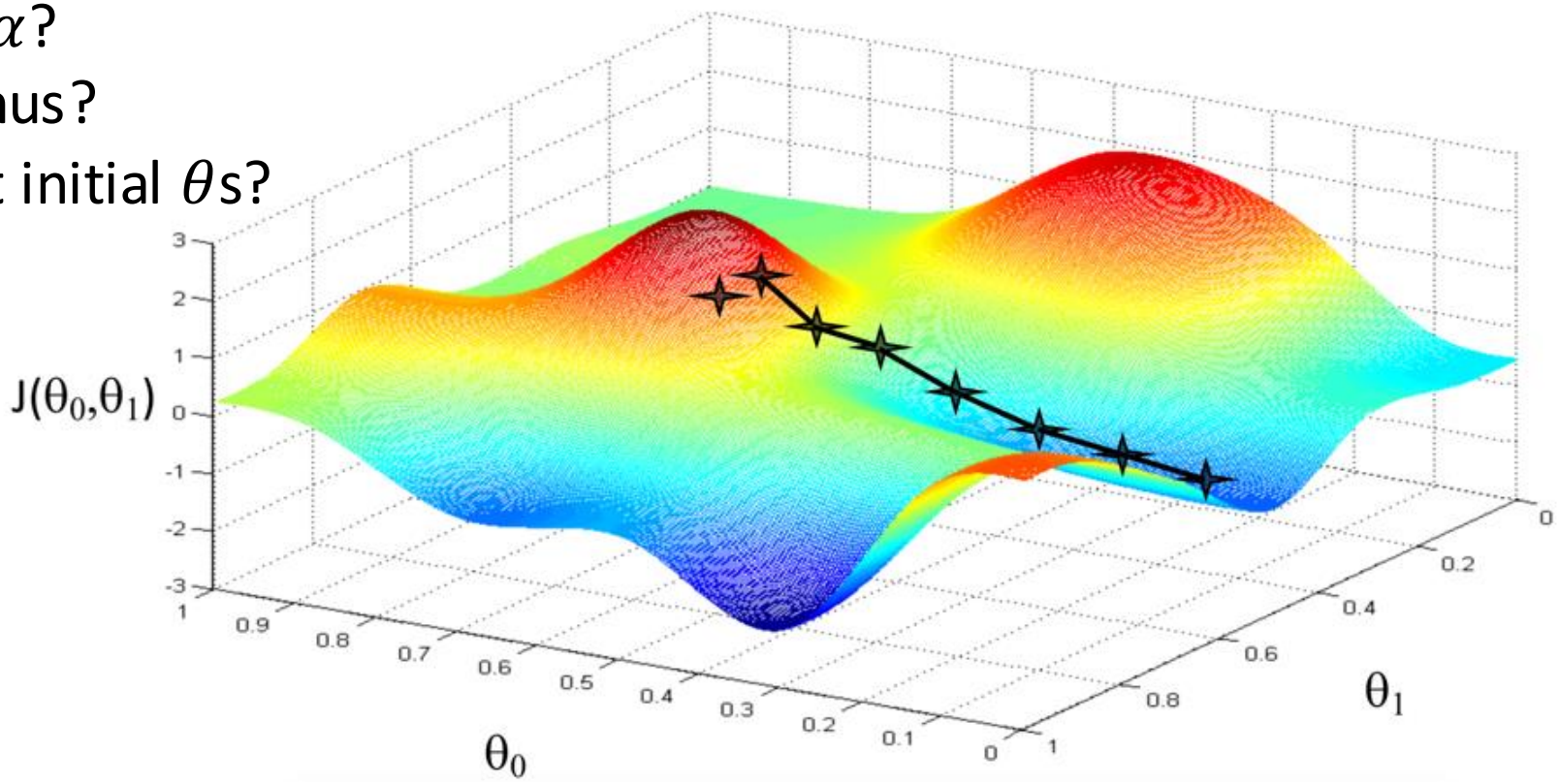
Gradient descent

- Choose initial value for θ.
- Until we reach the minimum, update θ to reduce $J(\theta)$:

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

Questions:

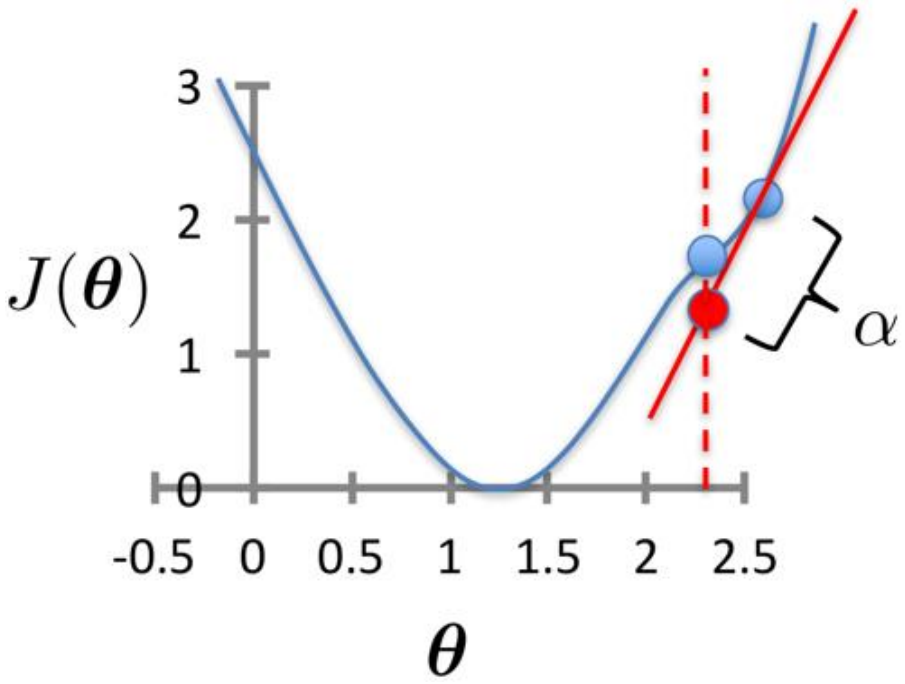
- What is α?
- Why minus?
- Different initial θs?



Gradient Descent

- Initialize θ.
- Repeat until convergence:

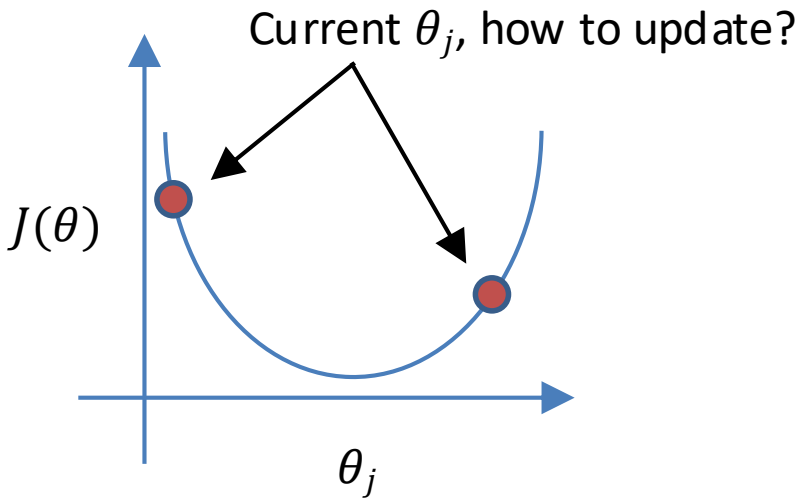
$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$



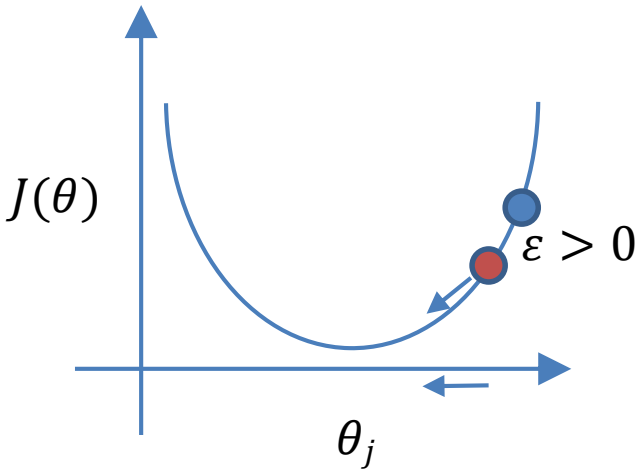
Gradient Descent

- Initialize θ.
- Repeat until convergence:

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

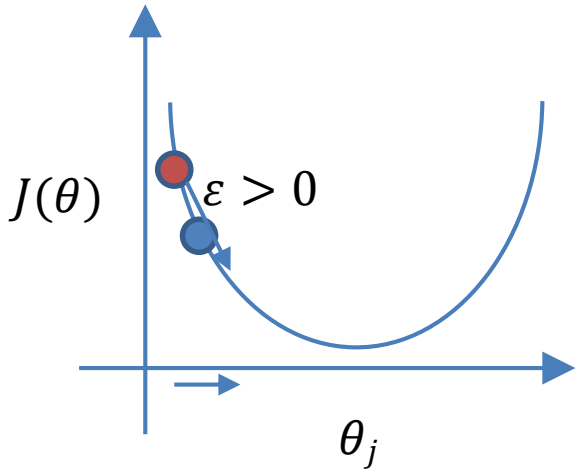


Gradient Descent



$$\frac{\partial J(\theta)}{\partial \theta_j} = \lim_{\varepsilon \to 0} \frac{J(\theta|\theta_j + \varepsilon) - J(\theta|\theta_j)}{\varepsilon} > 0$$

$\rightarrow \theta_j$ moves to the **negative direction** → **counter-direction** with respect to the partial derivative.



$$\frac{\partial J(\theta)}{\partial \theta_j} = \lim_{\varepsilon \to 0} \frac{J(\theta|\theta_j + \varepsilon) - J(\theta|\theta_j)}{\varepsilon} < 0$$

$\rightarrow \theta_j$ moves to the **positive direction** → **counter-direction** with respect to the partial derivative.

Gradient Descent

- Initialize θ.
- Repeat until convergence:

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

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

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Reminder
$(u^2)' = 2uu'$

Gradient Descent

- Initialize θ.
- Repeat until convergence:

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

Question: when to stop the algorithm, i.e., when the convergence is considered to happen?

Gradient Descent

- Initialize θ.
- Repeat until convergence:

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

Question: when to stop the algorithm, i.e., when the convergence is considered to happen?

Answer: convergence is reached when either

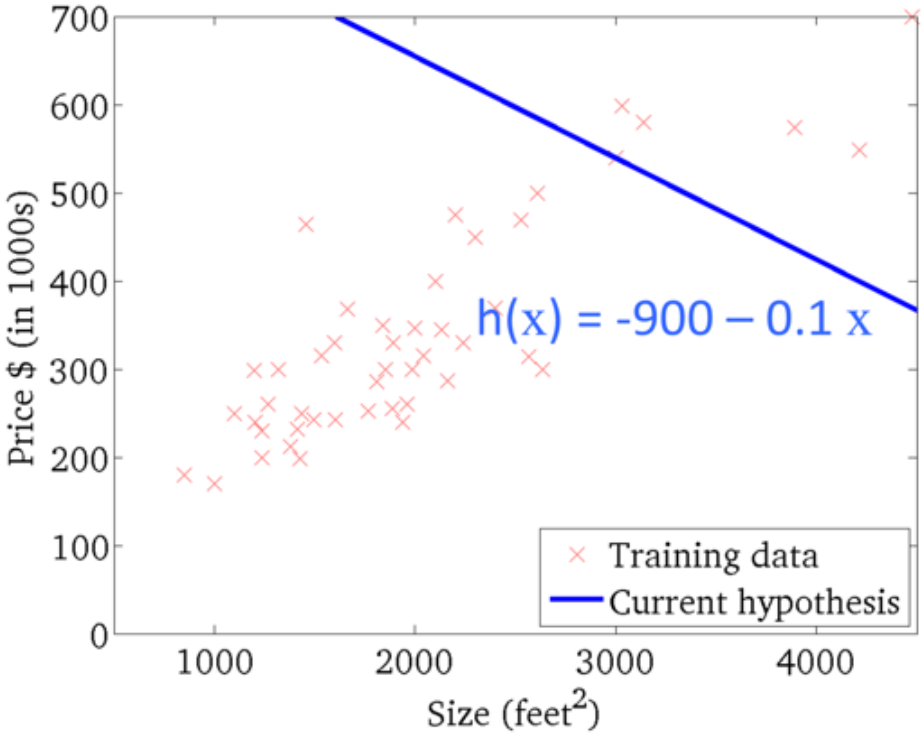
- $J(\theta) < \varepsilon$
- $\|\theta_{new} - \theta_{old}\|_2 < \varepsilon$ where L2-norm $\|v\|_2 = \sqrt{\sum_i v_i^2} = \sqrt{v_1^2 + v_2^2 + \cdots + v_{|v|}^2}$
- Model accuracy is good enough on an independent test set.

Question: how much is a model accuracy considered to be good enough?

Gradient Descent

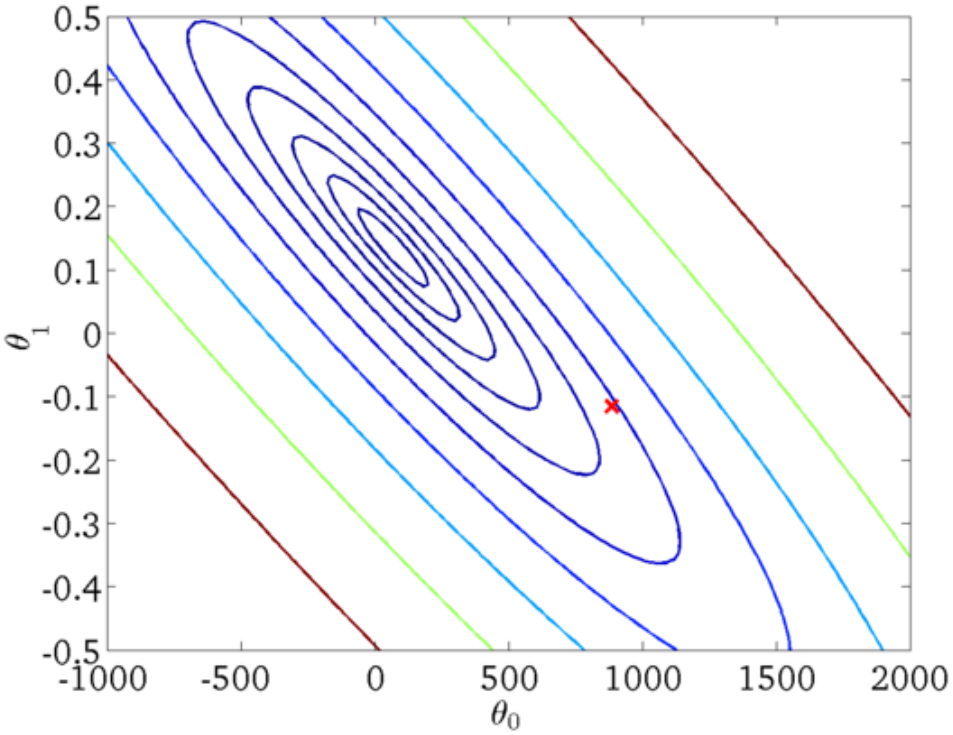
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

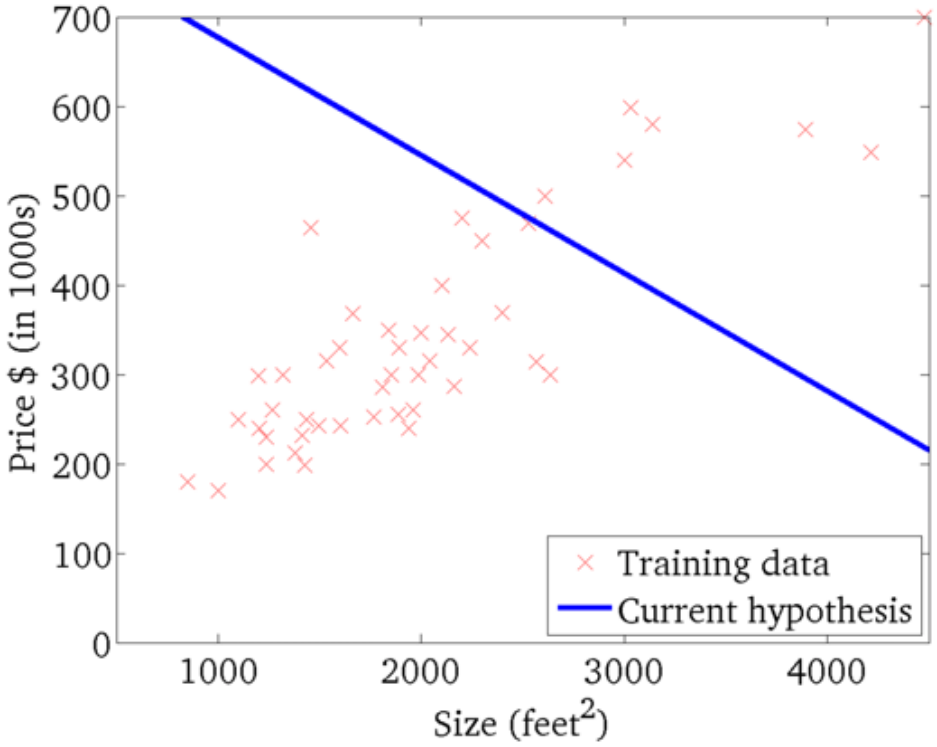
(function of the parameters θ_0, θ_1)



Gradient Descent

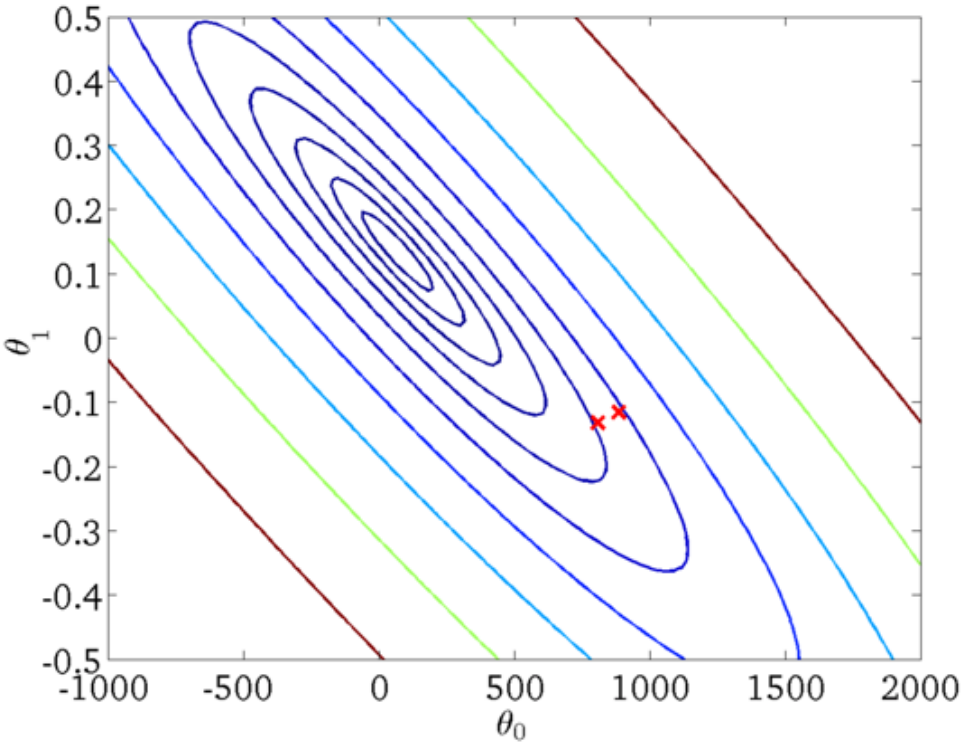
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

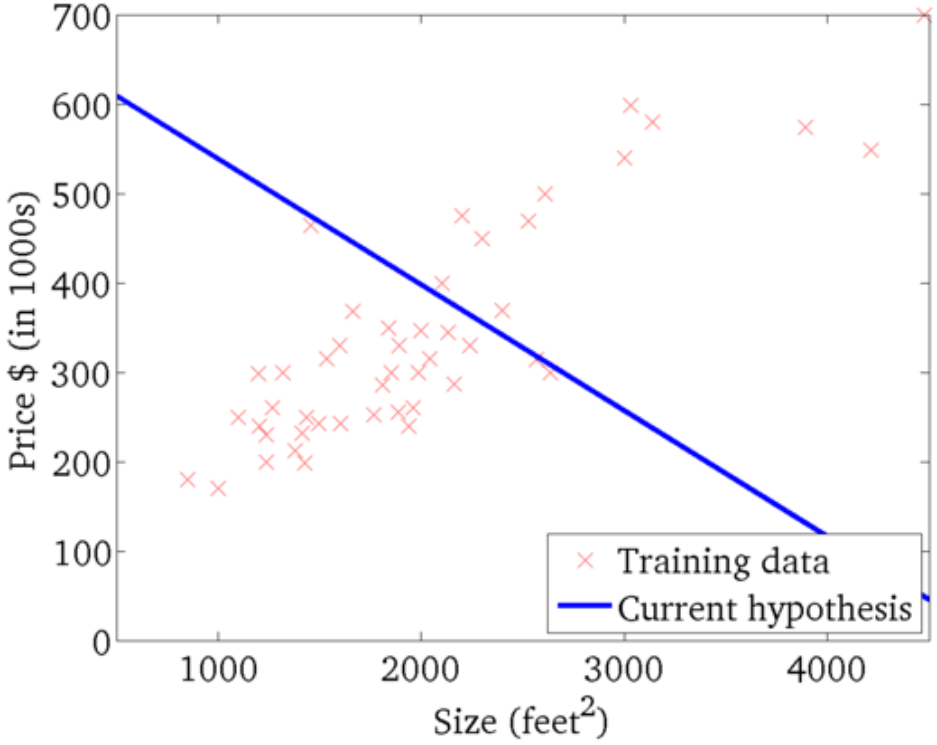
(function of the parameters θ_0, θ_1)



Gradient Descent

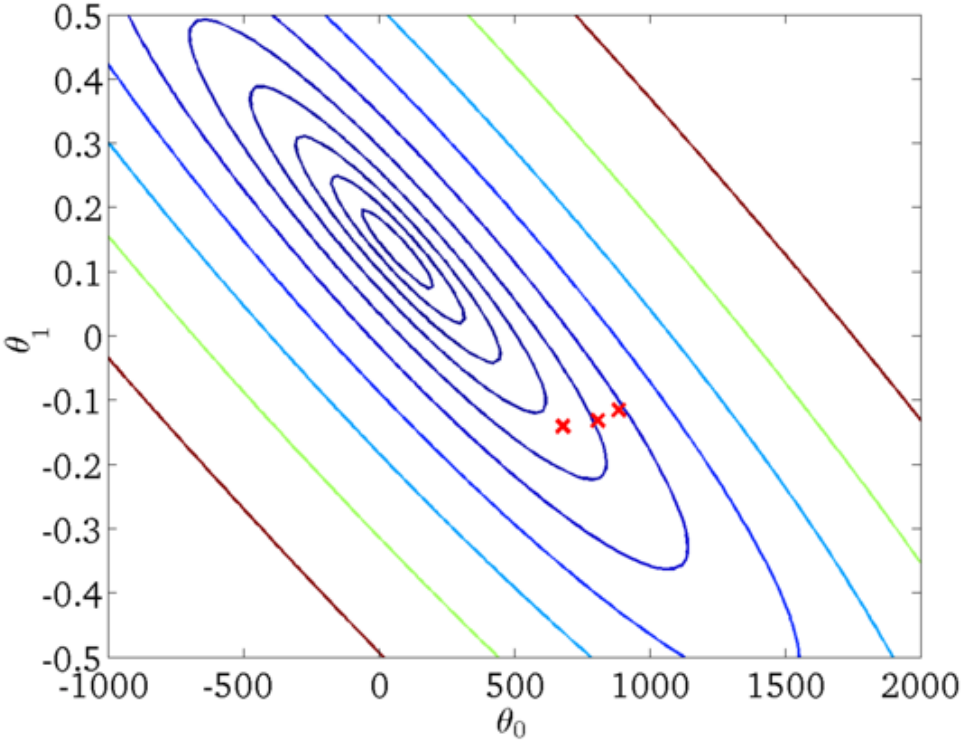
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

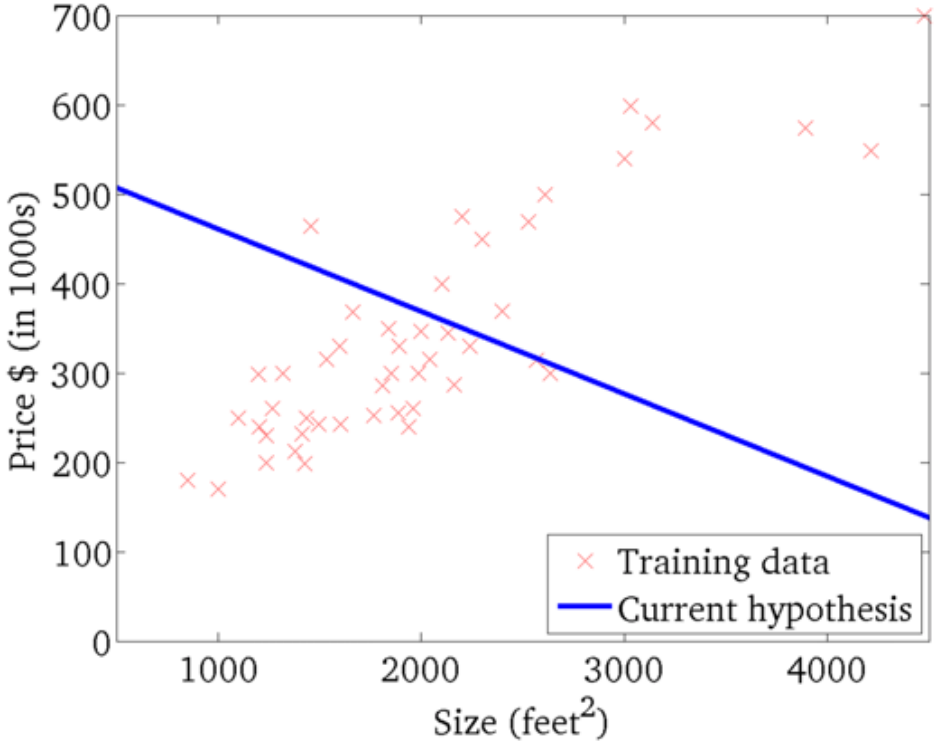
(function of the parameters θ_0, θ_1)



Gradient Descent

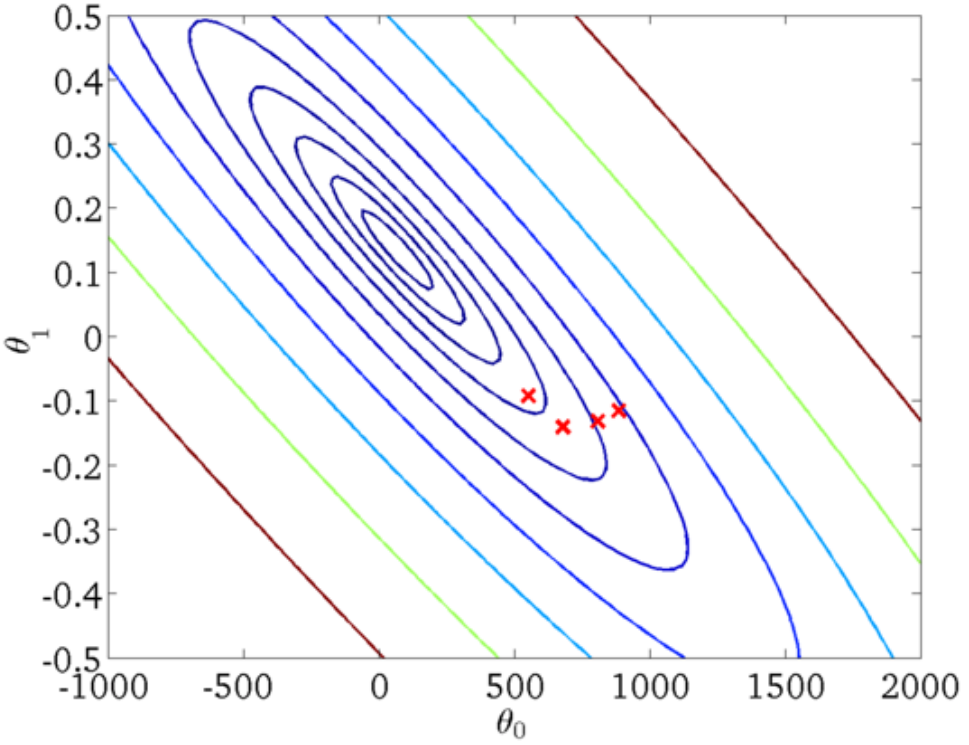
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

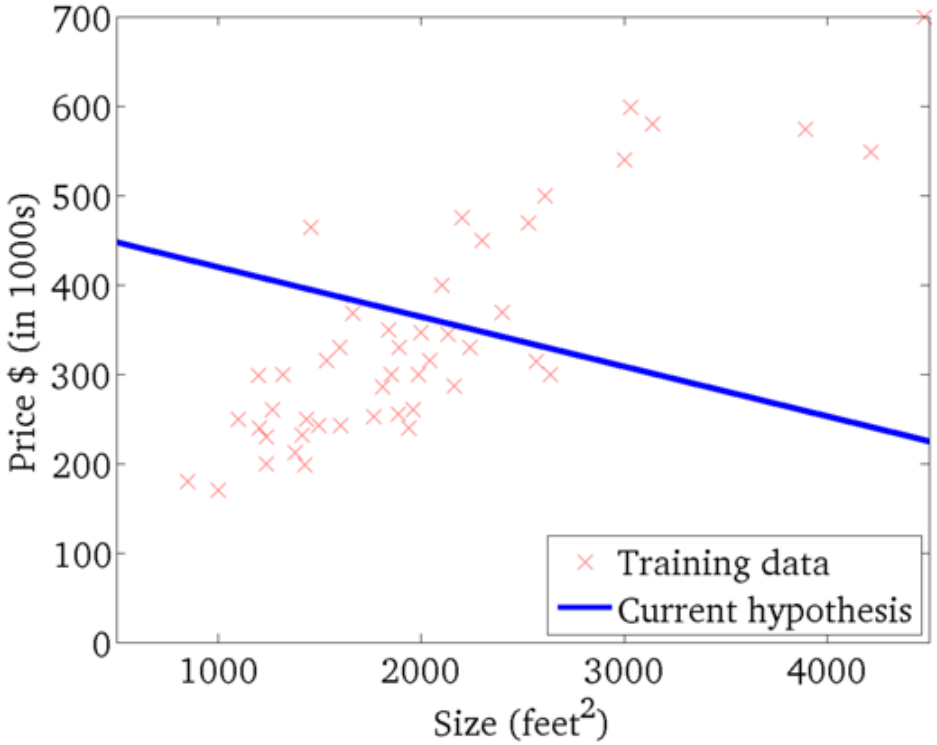
(function of the parameters θ_0, θ_1)



Gradient Descent

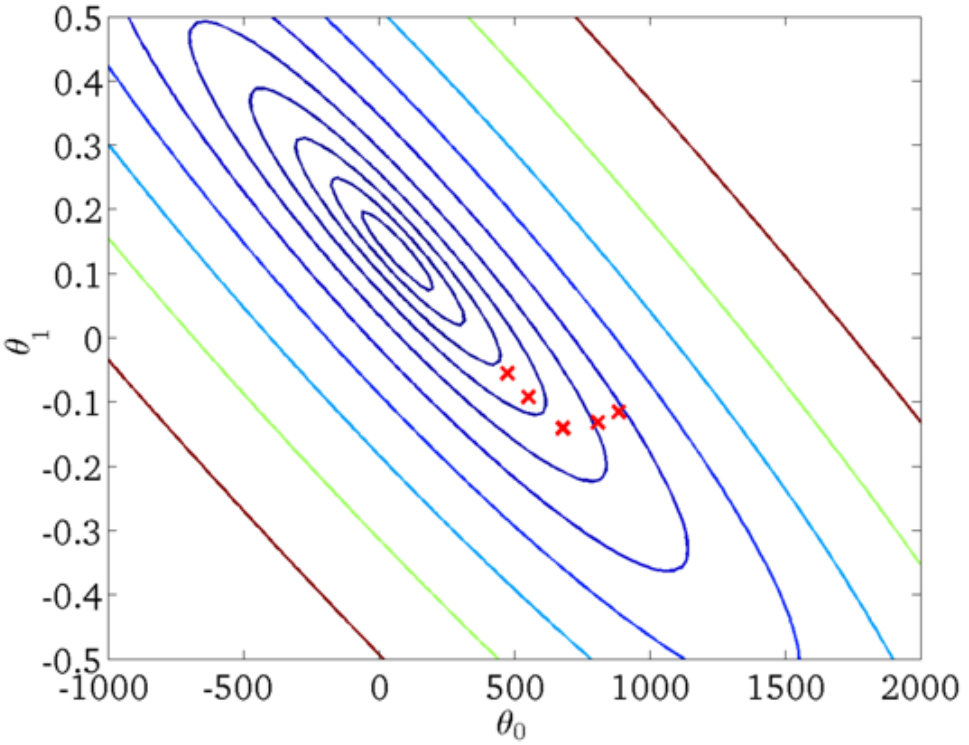
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

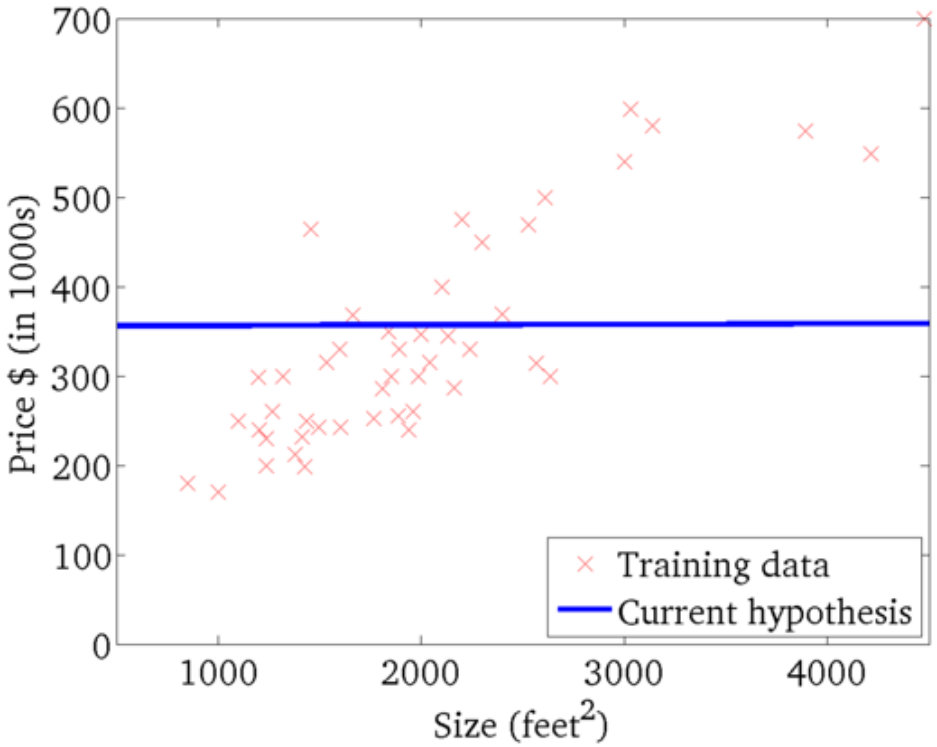
(function of the parameters θ_0, θ_1)



Gradient Descent

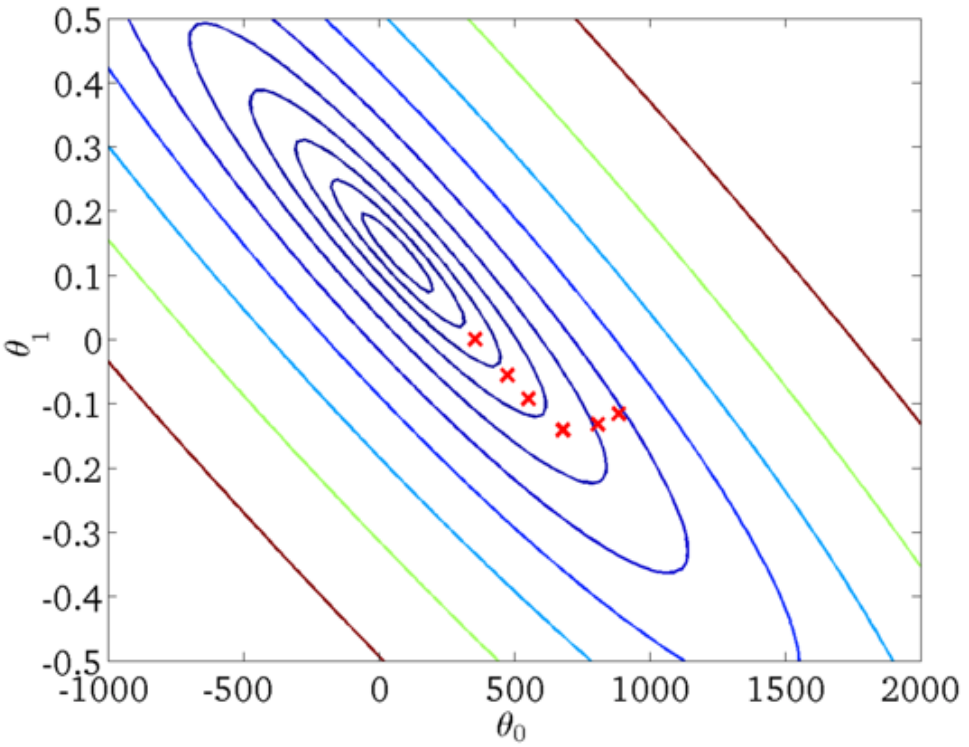
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

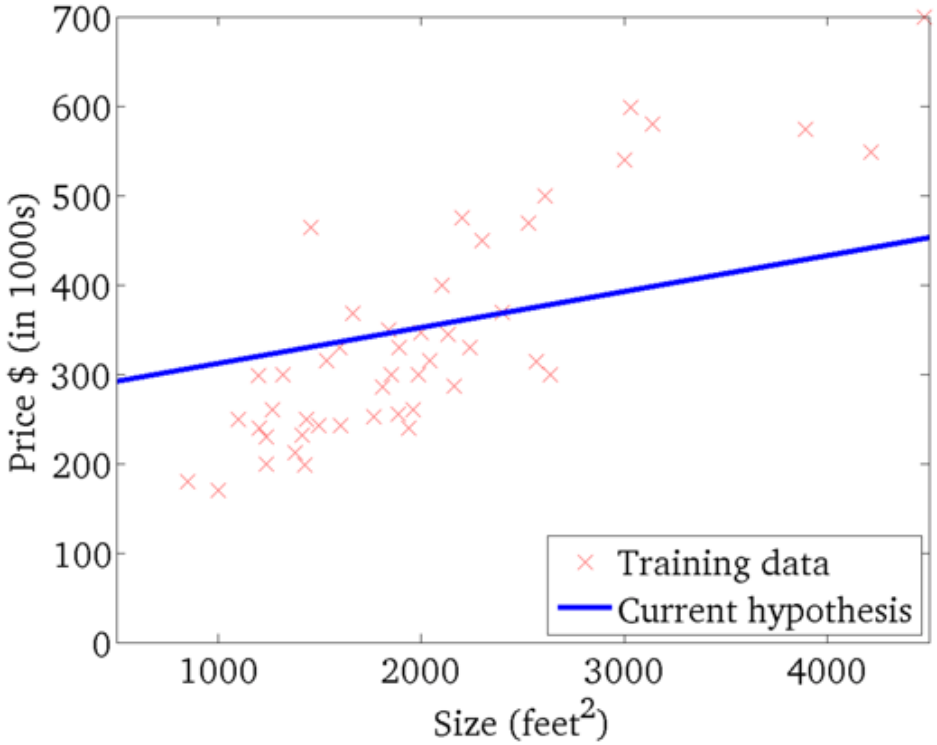
(function of the parameters θ_0, θ_1)



Gradient Descent

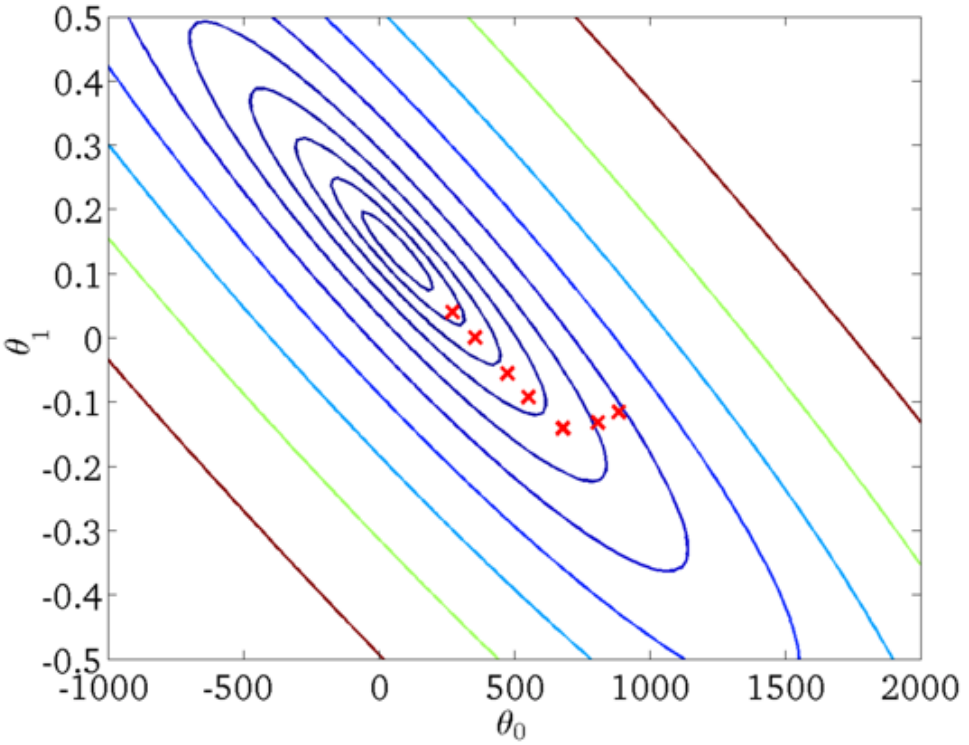
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

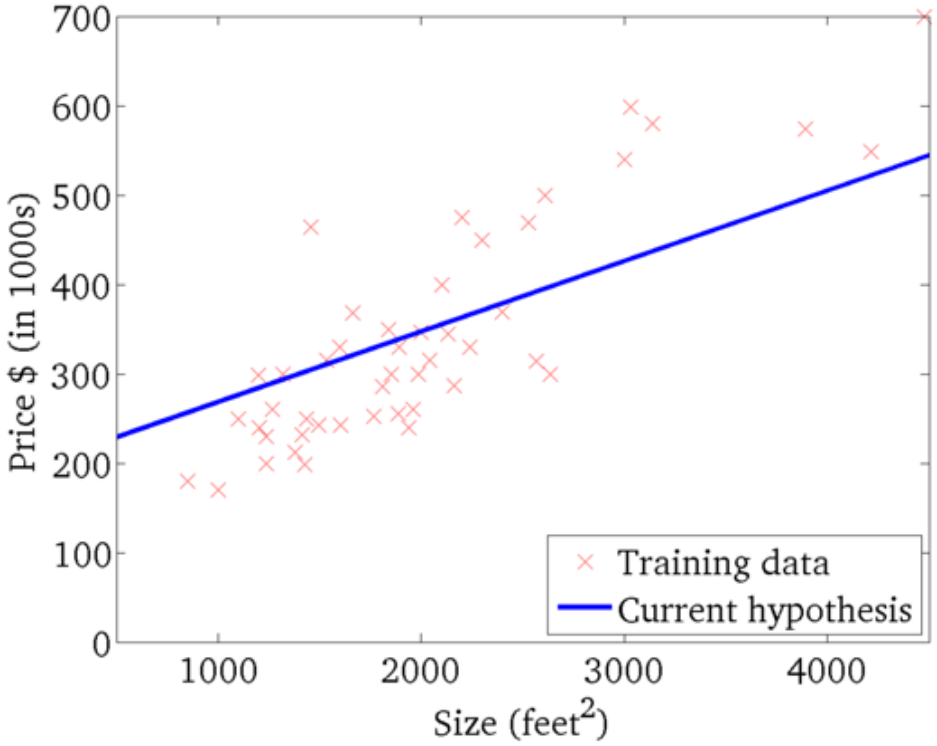
(function of the parameters θ_0, θ_1)



Gradient Descent

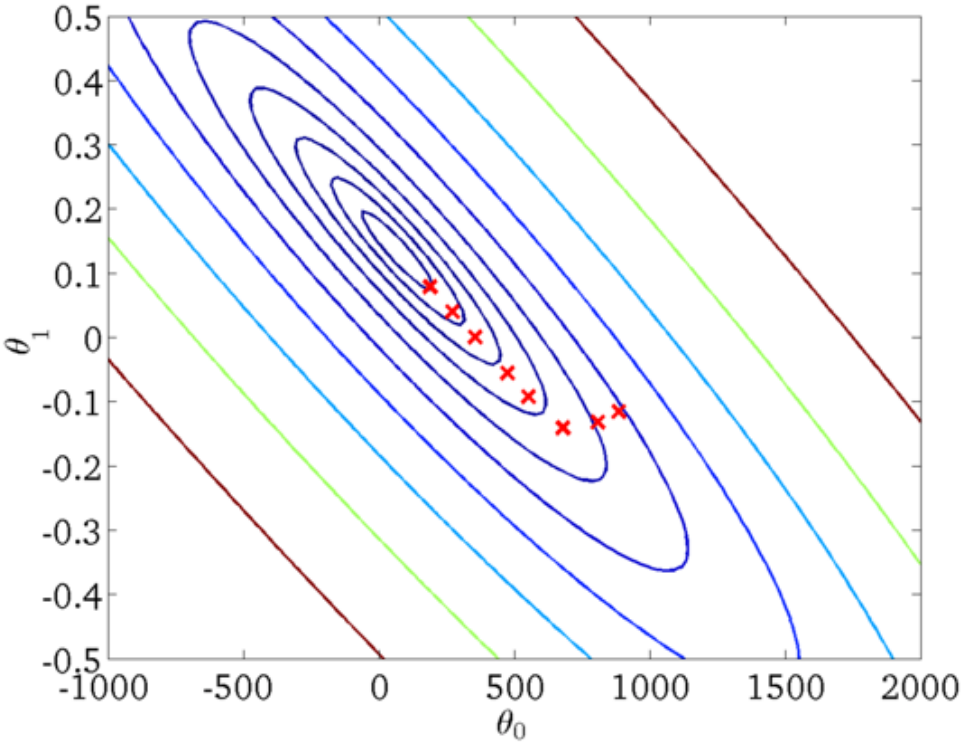
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

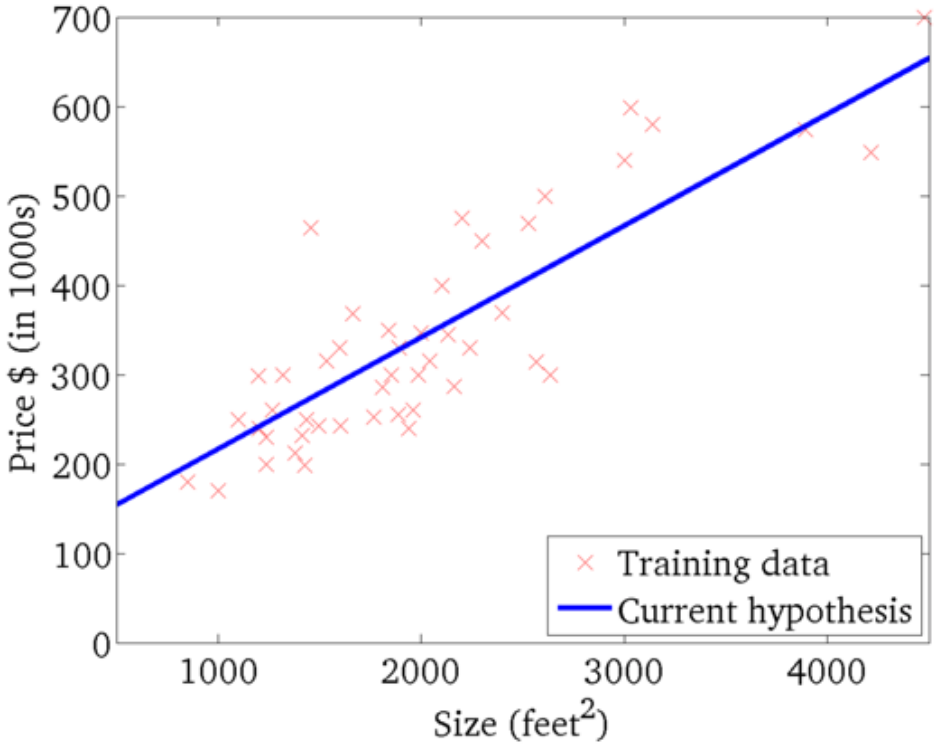
(function of the parameters θ_0, θ_1)



Gradient Descent

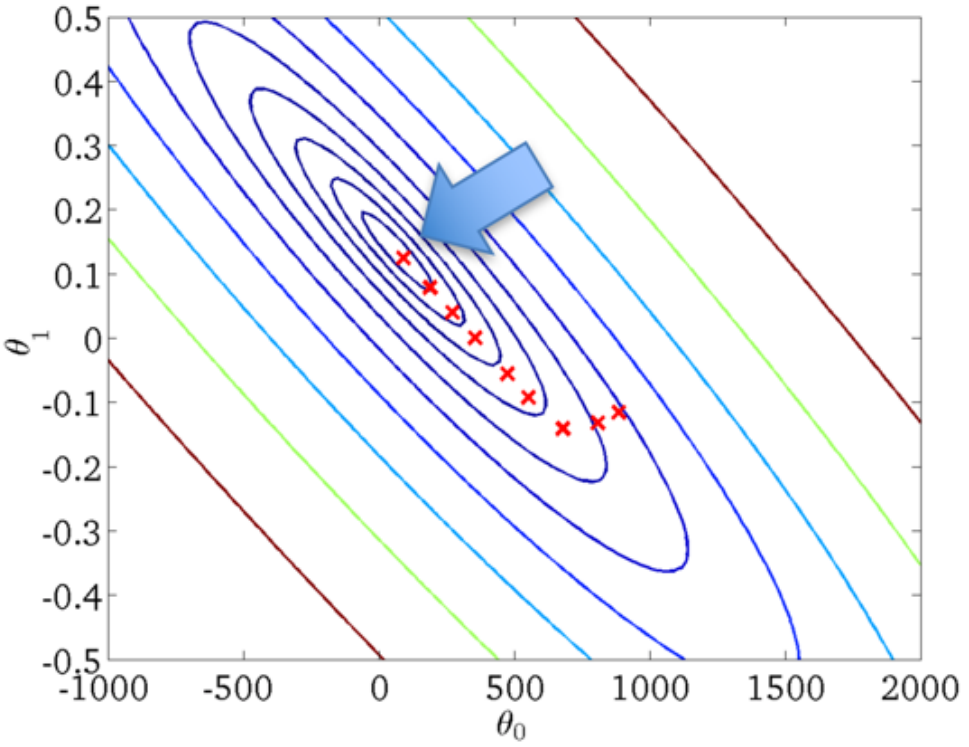
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Gradient Descent variants

There are three variants of gradient descent.

- **Batch** gradient descent
- **Stochastic** gradient descent
- **Mini-batch** gradient descent

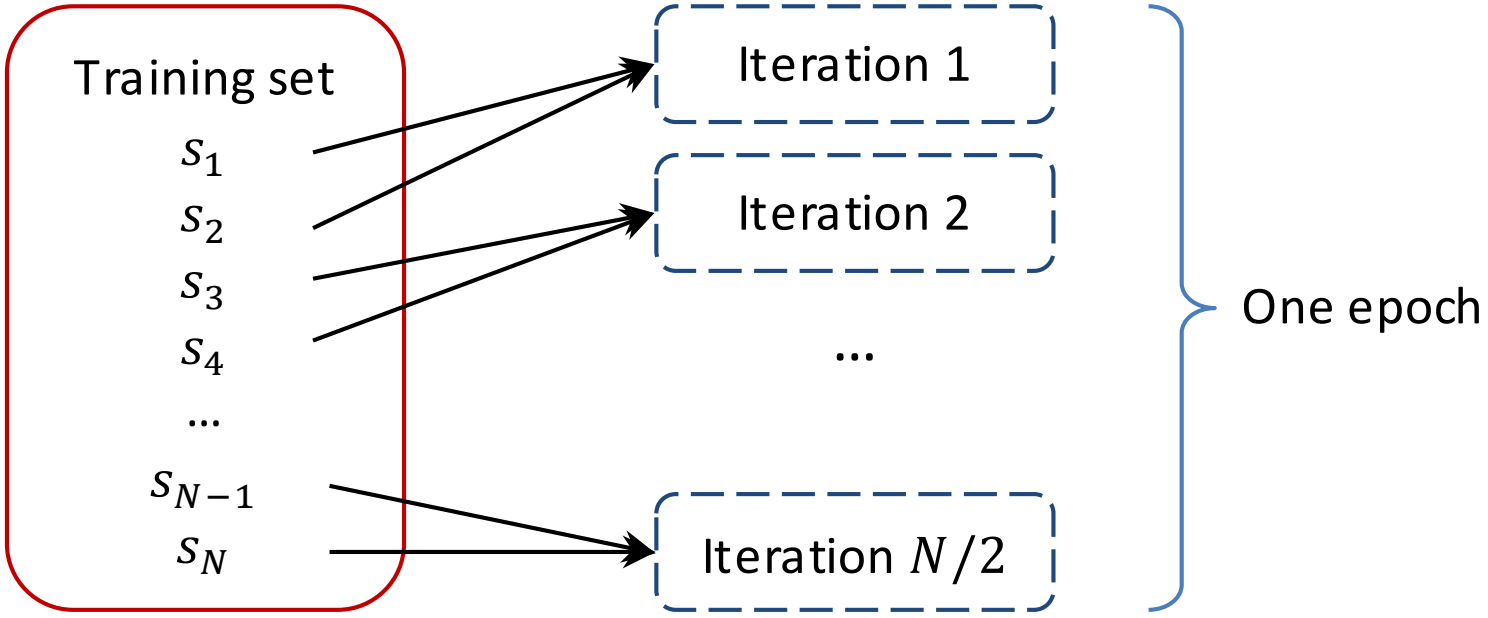
The difference of these algorithms is the *amount of data*.

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

This term is different by each method

Iteration vs. Epoch

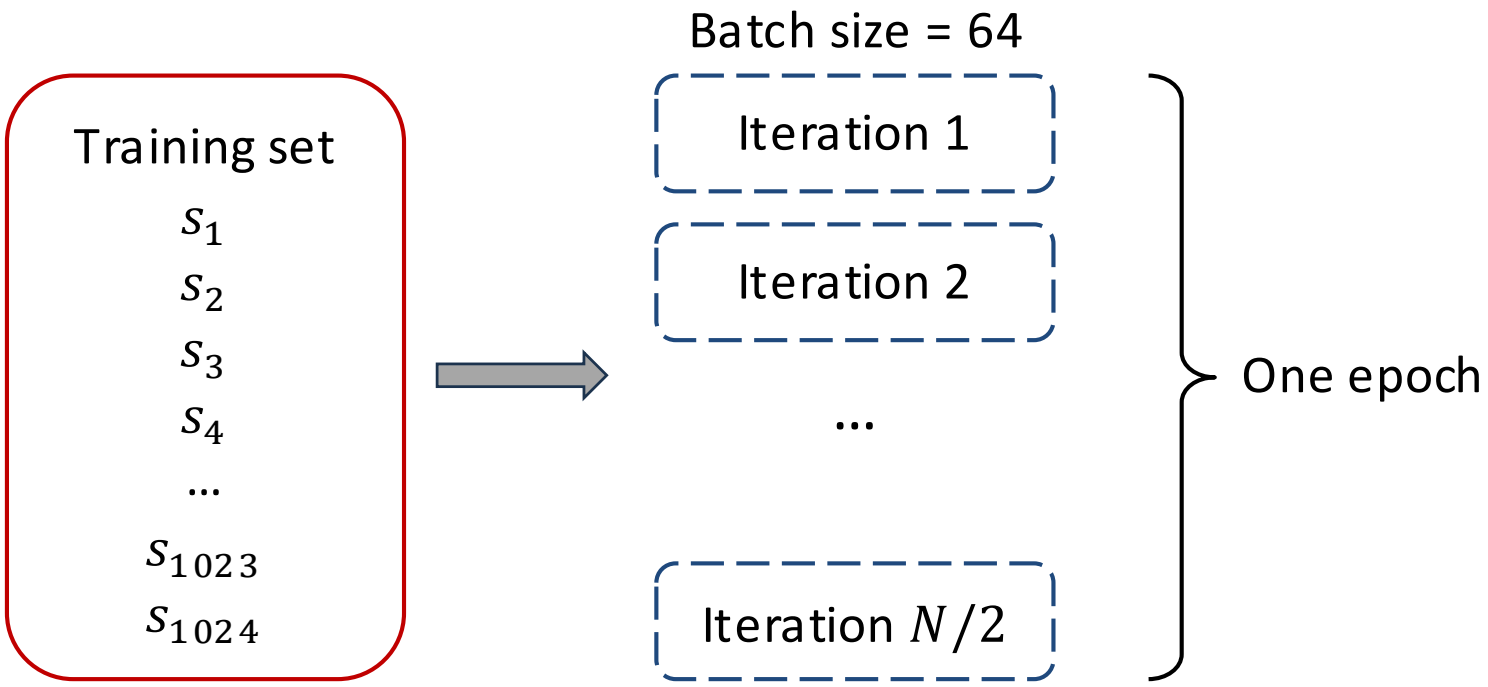
One **iteration** is equivalent to when a **batch of data** samples is fed to the model for training.
One **epoch** is equivalent to when the **entire dataset** is fed to the model for training.



An example about the difference between one iteration and one epoch, with a dataset of N data samples and batch size $= 2$.

Iteration vs. Epoch

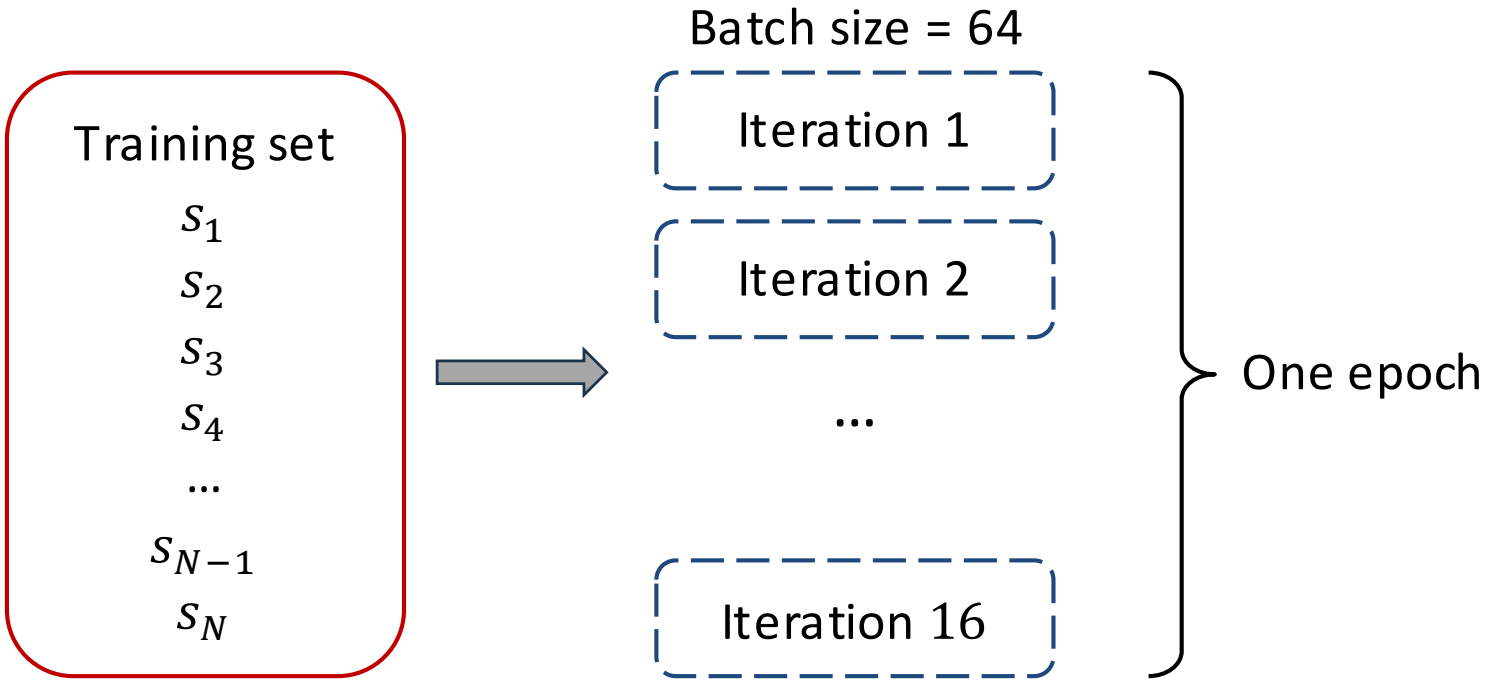
Question: how many iterations are there in one epoch if the dataset has 1024 data samples and the batch size is 64?



Iteration vs. Epoch

Question: how many iterations are there in one epoch if the dataset has 1024 data samples and the batch size is 64?

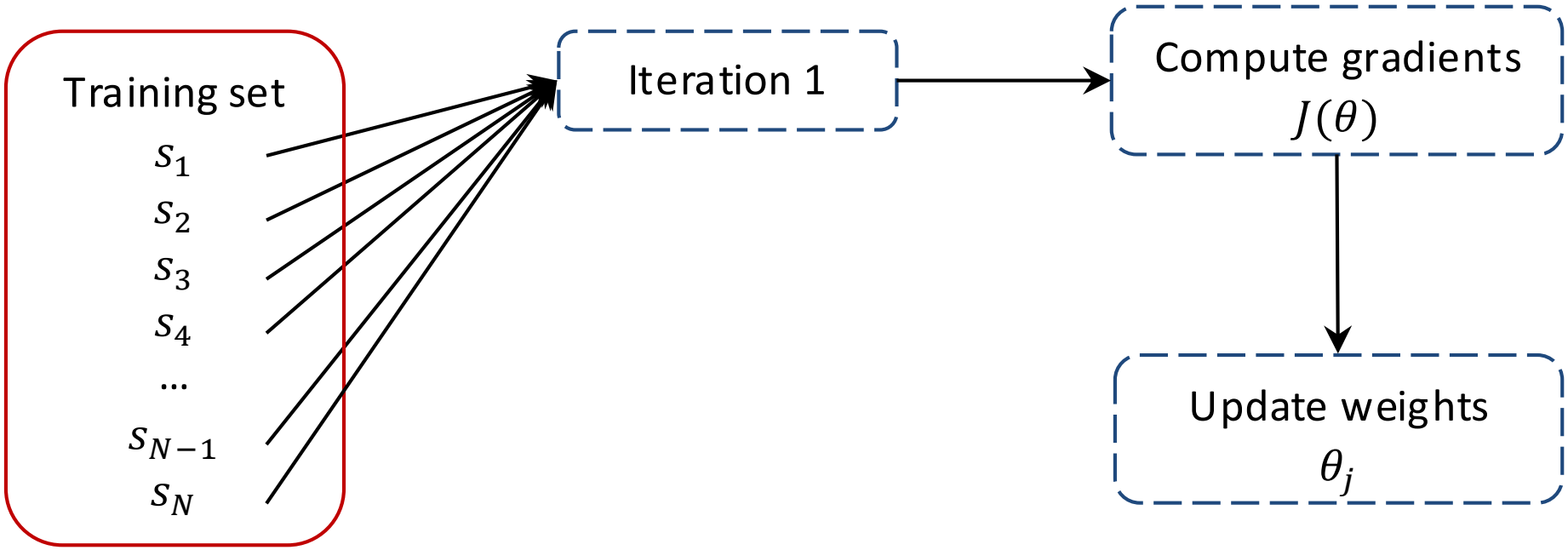
Answer: one epoch is equivalent to 1024/64 = 16 iterations.



Batch gradient descent

Batch gradient descent: Considers the **entire training dataset** at once to calculate the average gradient.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta, x^{(1:N)}, y^{(1:N)})$$



Batch gradient descent

Advantages:

- *Accurate Updates*: Uses the entire dataset for gradient calculation, leading to a more accurate estimation of the true gradient.
- *Smooth Convergence*: Updates tend to be smoother, leading to a more stable descent towards the minimum.

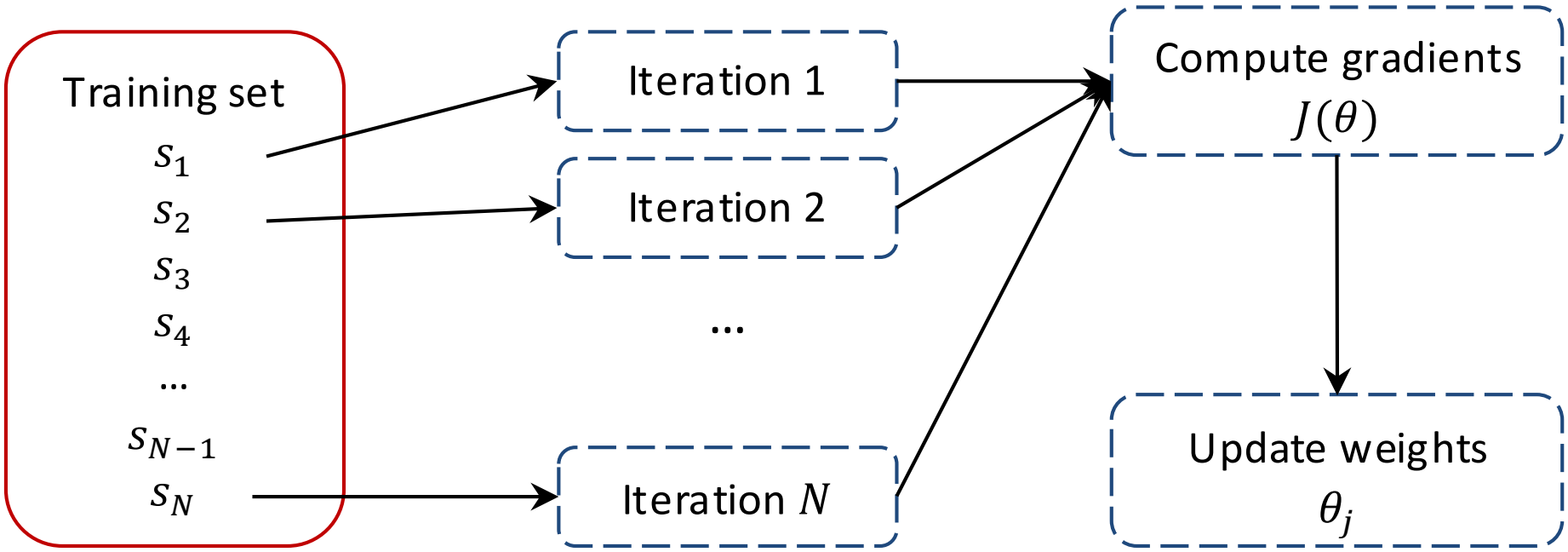
Disadvantages:

- *Slow for Large Datasets*: This method requires processing the entire dataset during each training iteration, leading to slow training times for big data.
- *Memory Intensive*: Storing the entire dataset in memory is necessary, which can be a challenge for large datasets.

Stochastic gradient descent

- Stochastic gradient descent: Uses a **single data point** at a time to calculate the gradient.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta, x^{(i)}, y^{(i)})$$



Stochastic gradient descent

Advantages:

- *Fast:* Processes data one point at a time, making it very fast for large datasets.
- *Less Memory Intensive:* Only needs to store a single data point in memory at a time.
- *Can Escape Local Minima:* Noise from individual examples can sometimes help escape local minima.

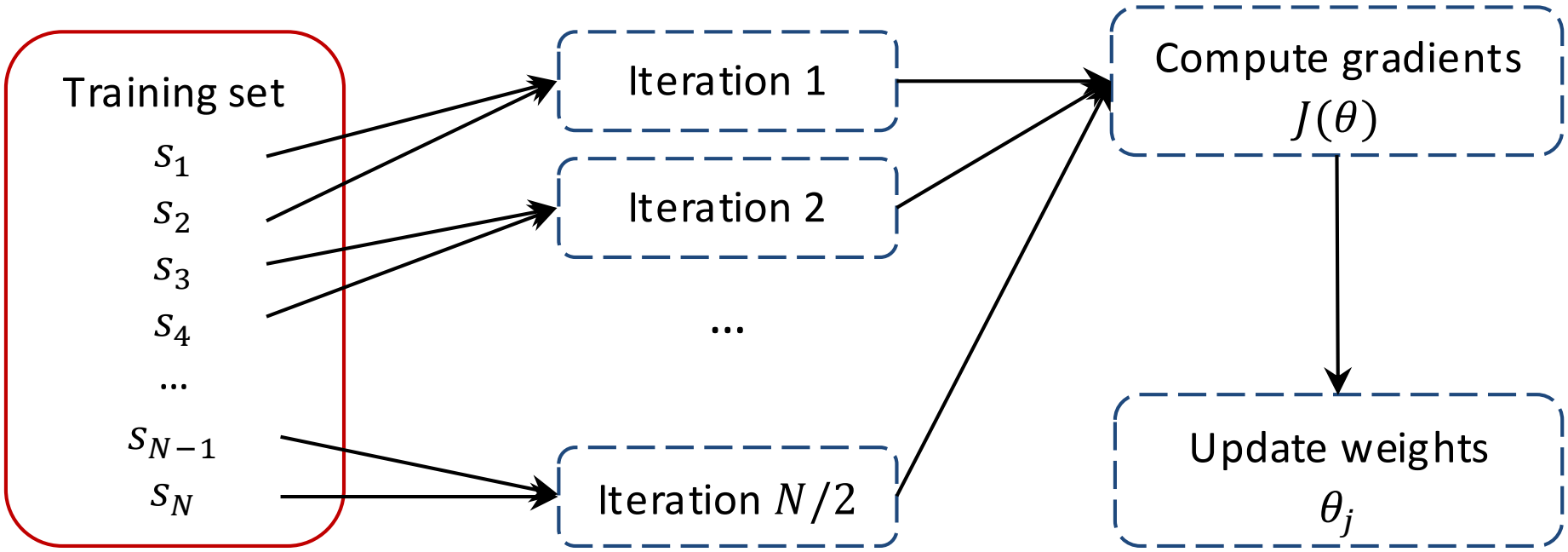
Disadvantages:

- *Noisy Updates:* Updates based on single data points are noisy, leading to erratic convergence and oscillations.
- *Less Stable Convergence:* Can be less stable than Batch GD, potentially getting stuck in bad local minima.

Mini-batch gradient descent

Mini-batch gradient descent: Uses a **subset of the data** (mini-batch) to calculate the gradient. The mini-batch size $\boldsymbol{k}$ is a hyperparameter you can tune.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta, x^{(i:i+k)}, y^{(i:i+k)})$$



Mini-batch gradient descent

Advantages:

- *Faster than Batch GD:* Processes data in batches, making it faster for large datasets.
- *More Stable than SGD:* Uses more data than SGD, leading to smoother updates and less noisy convergence.
- *Less Memory Intensive:* Requires storing only a small batch of data in memory.

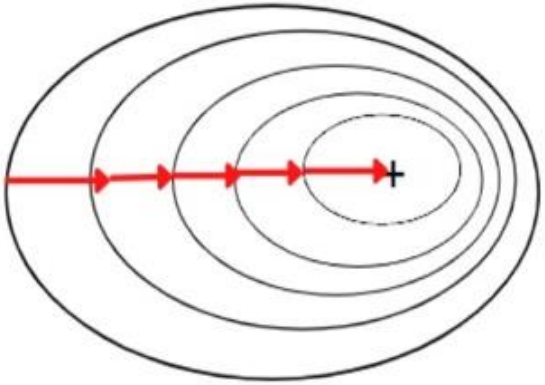
Disadvantages:

- *Not as Accurate as Batch GD:* Updates are less accurate than Batch GD because they only use a subset of the data.
- *Tuning Batch Size:* Requires choosing the right batch size for optimal performance.

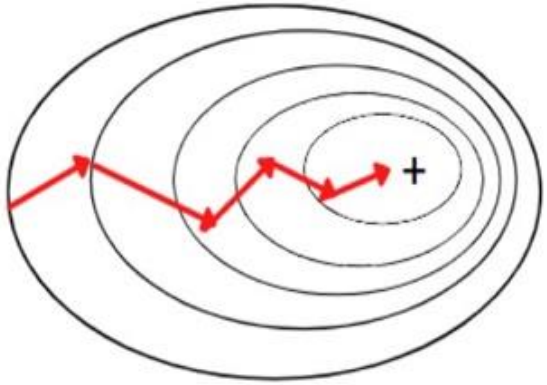
Question: How to choose the optimal batch size?

Batch vs. Mini-batch vs. Stochastic gradient descent

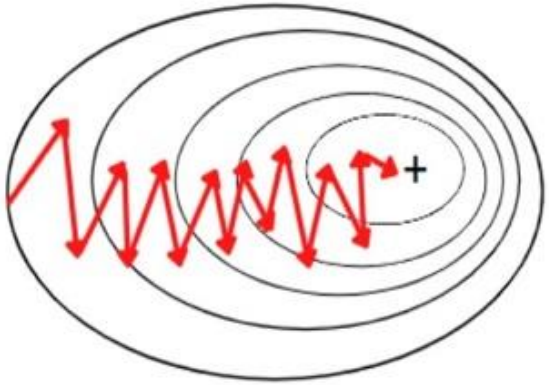
Batch gradient descent



Mini-batch gradient descent



Stochastic gradient descent



Convergence comparison between Batch, Mini-batch and Stochastic gradient descent.

Summary of gradient descent variants

Comparison between Batch, Mini-batch and Stochastic gradient descent in terms of convergence quality, running time and memory usage.

Method	Convergence	Running time	Memory Usage
Batch gradient descent	Good	Slow	High
Stochastic gradient descent	Not very good	Fast	Low
Mini-batch gradient descent	Medium	Medium	Medium

Note: With the use of GPUs, batch and mini-batch gradient descent can handle multiple data samples simultaneously, leading to faster running time.

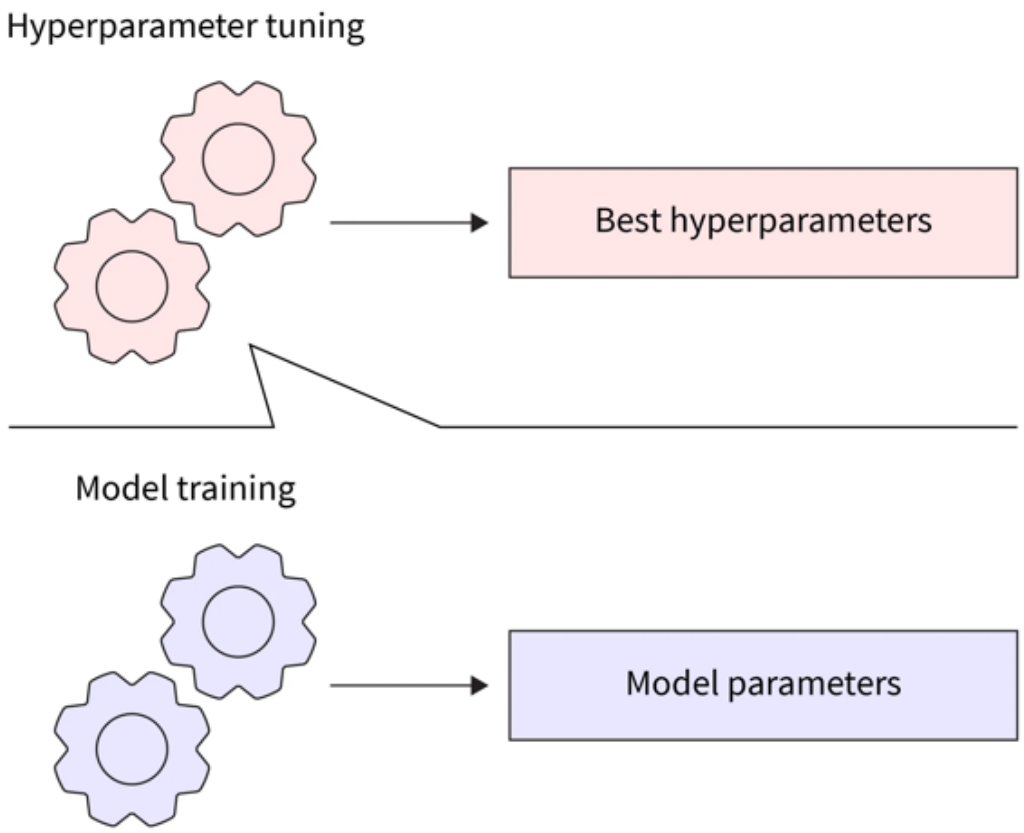
Question: Which variant of gradient descent do you believe is most commonly used in practical applications?

Model hyper-parameters

In machine learning, a hyperparameter is essentially a control knob you **adjust before training** a model. It determines the learning process itself, rather than being learned from the data.

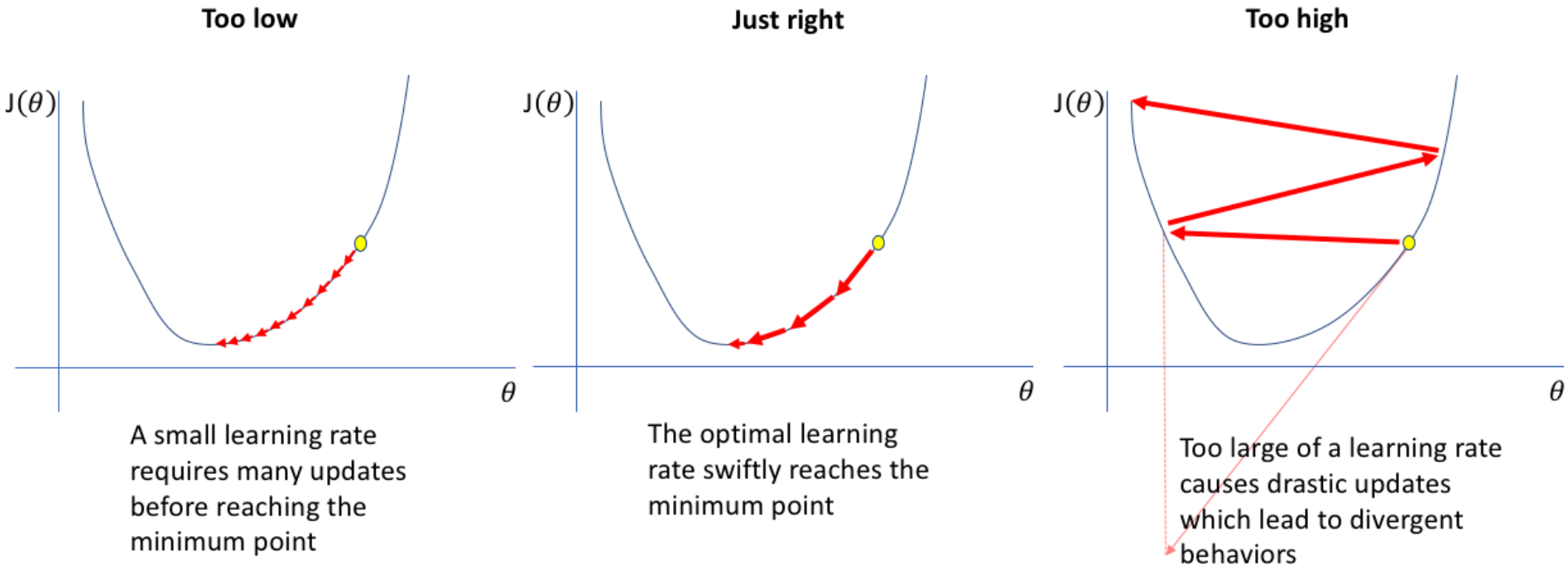
Example of hyperparameters:

- Batch size.
- Learning rate.
- Model function.
- Regularization weight.
- …



Model hyperparameters - Learning Rate

Learning rate is a hyperparameter. It determines the **step size** used to update the model's learnable parameters during training.



Impact of learning rate on model convergence in three scenarios: too low, just right and too high learning rate.

Question: How can we know when to increase / decrease / keep the same the learning rate when training a model?

Linear Regression for more complex models

Linear regression can have complex forms:

- Transformation of quantitative inputs:

  Example: $h_\theta(x) = \theta_0 + \theta_1 \log(x_1) + \theta_2 x_2^2 + \theta_3 \sqrt{x_3}$

- Polynomial transformation

  Example: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$

- Interaction between variables:

  Example: $h_\theta(x) = \theta_0 + \theta_1 x_1 x_2 + \theta_2 x_1 x_3 + \theta_d x_2 x_3$

The general form of the transformations is:

$$h_\theta(x) = \sum_{j=0}^{d} \theta_j \underbrace{\emptyset_j(x)}_{\text{Basis function}}$$

where typically $\emptyset_j(x) = 1$ so that θ_0 acts as a bias.

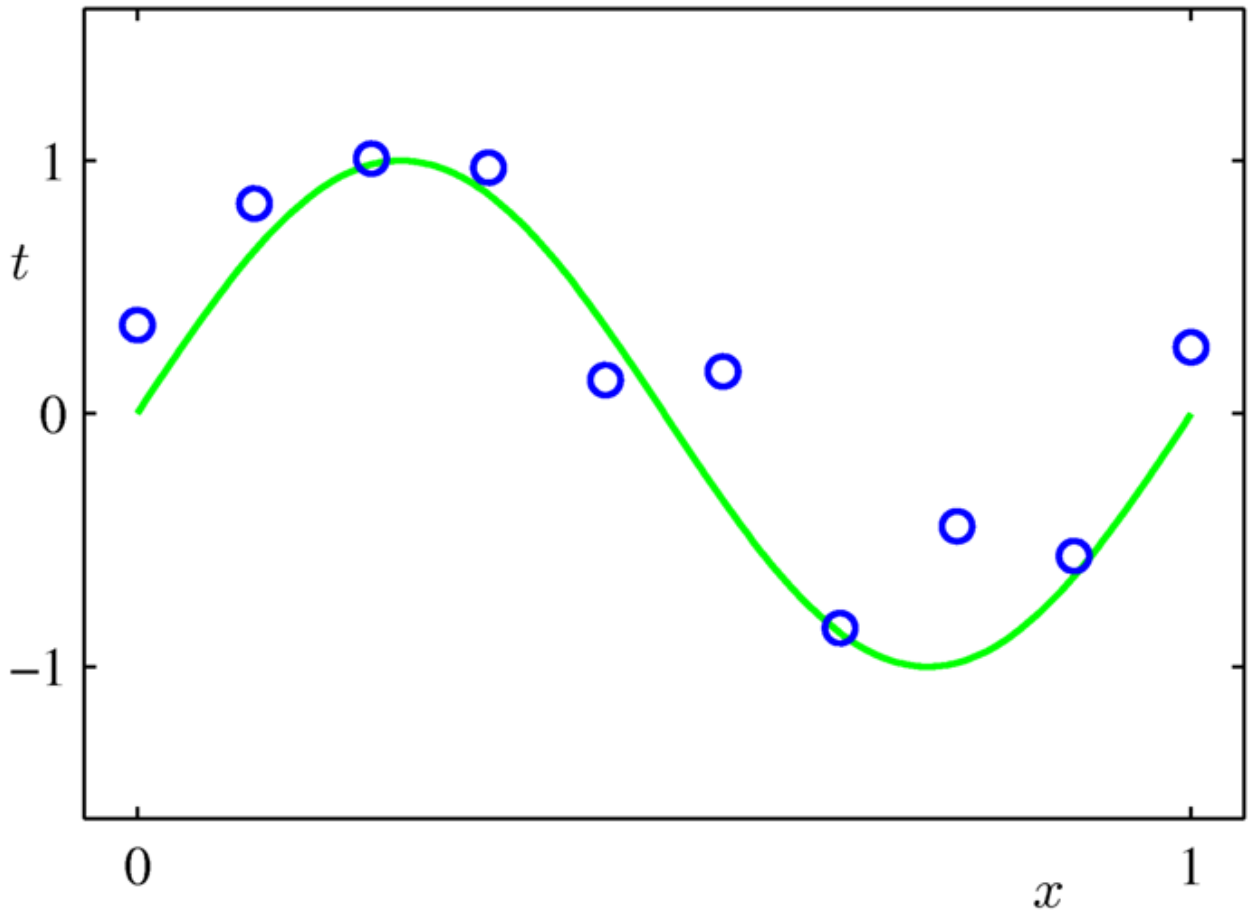
In the simplest case, we use linear basis function:

$$\emptyset_j(x) = x_j$$

These transformation allows the use of linear regression to fit non-linear dataset.

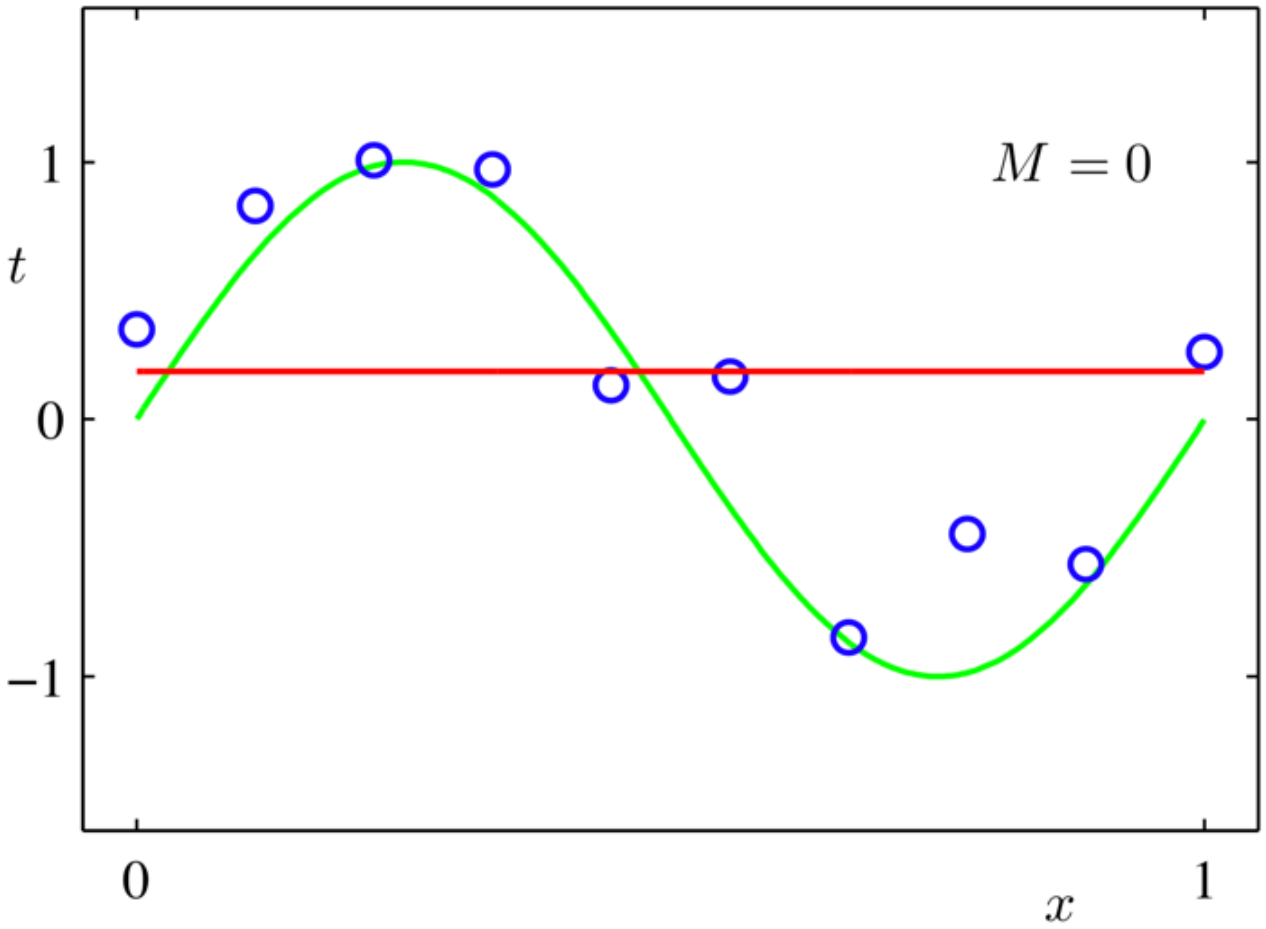
Question: The transformations are not linear. Does linear regression still work/apply?

Example of fitting a Polynomial Curve with Linear Regression



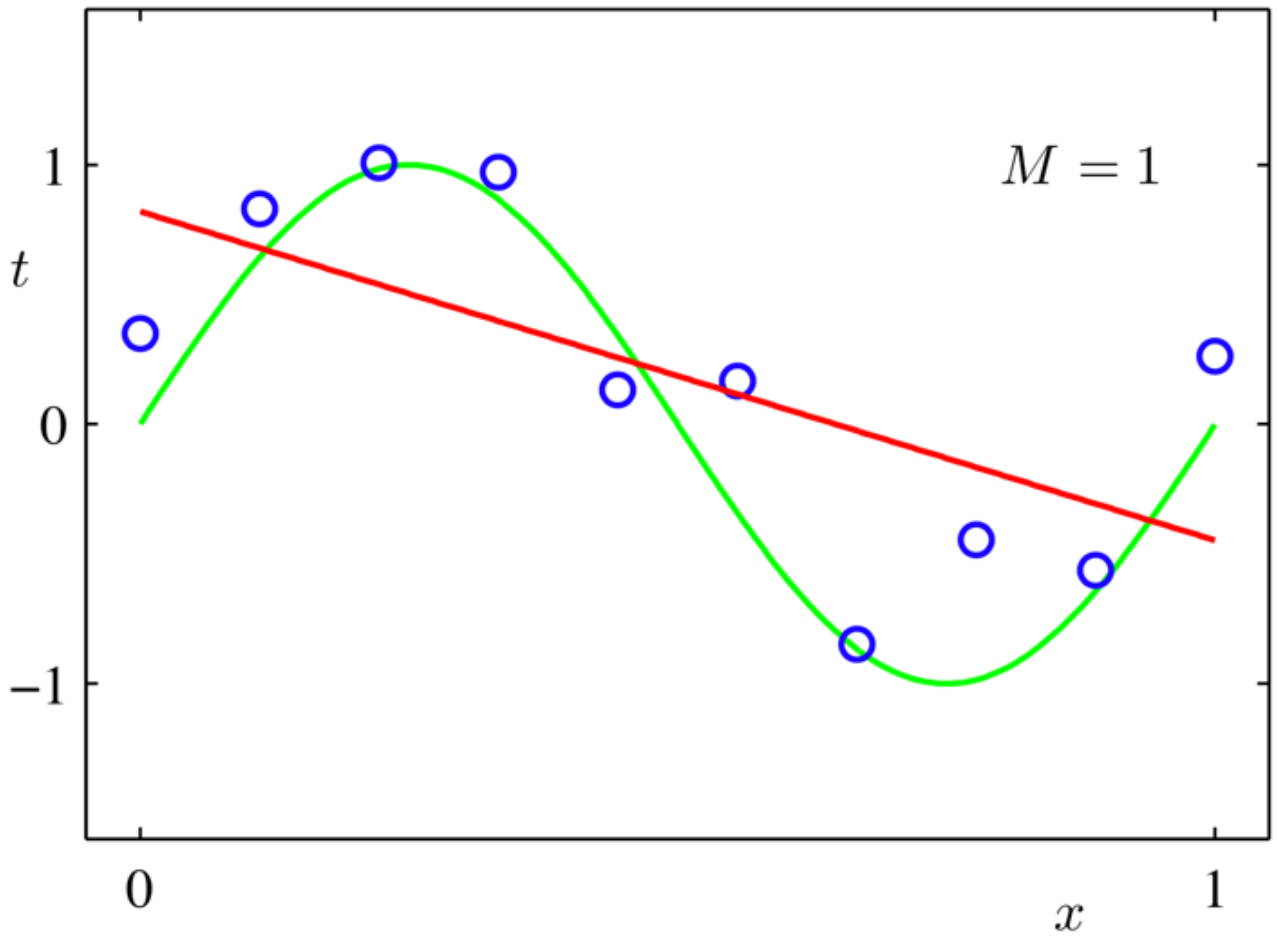
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_d x^d = \sum_{j=0}^{d} \theta_j \, x^j$$

Example of fitting a Polynomial Curve with Linear Regression



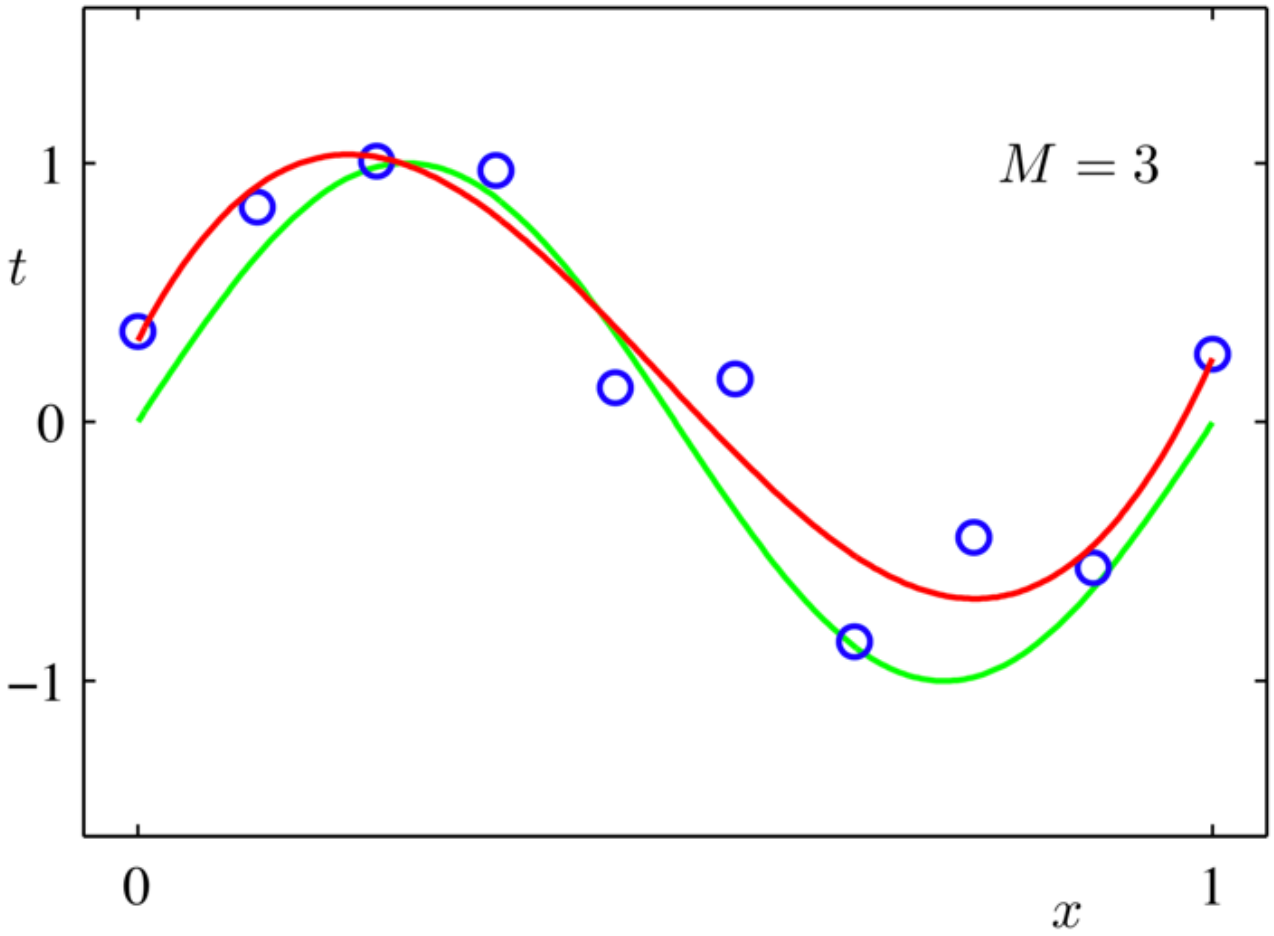
$$h_\theta(x) = \theta_0$$

Example of fitting a Polynomial Curve with Linear Regression



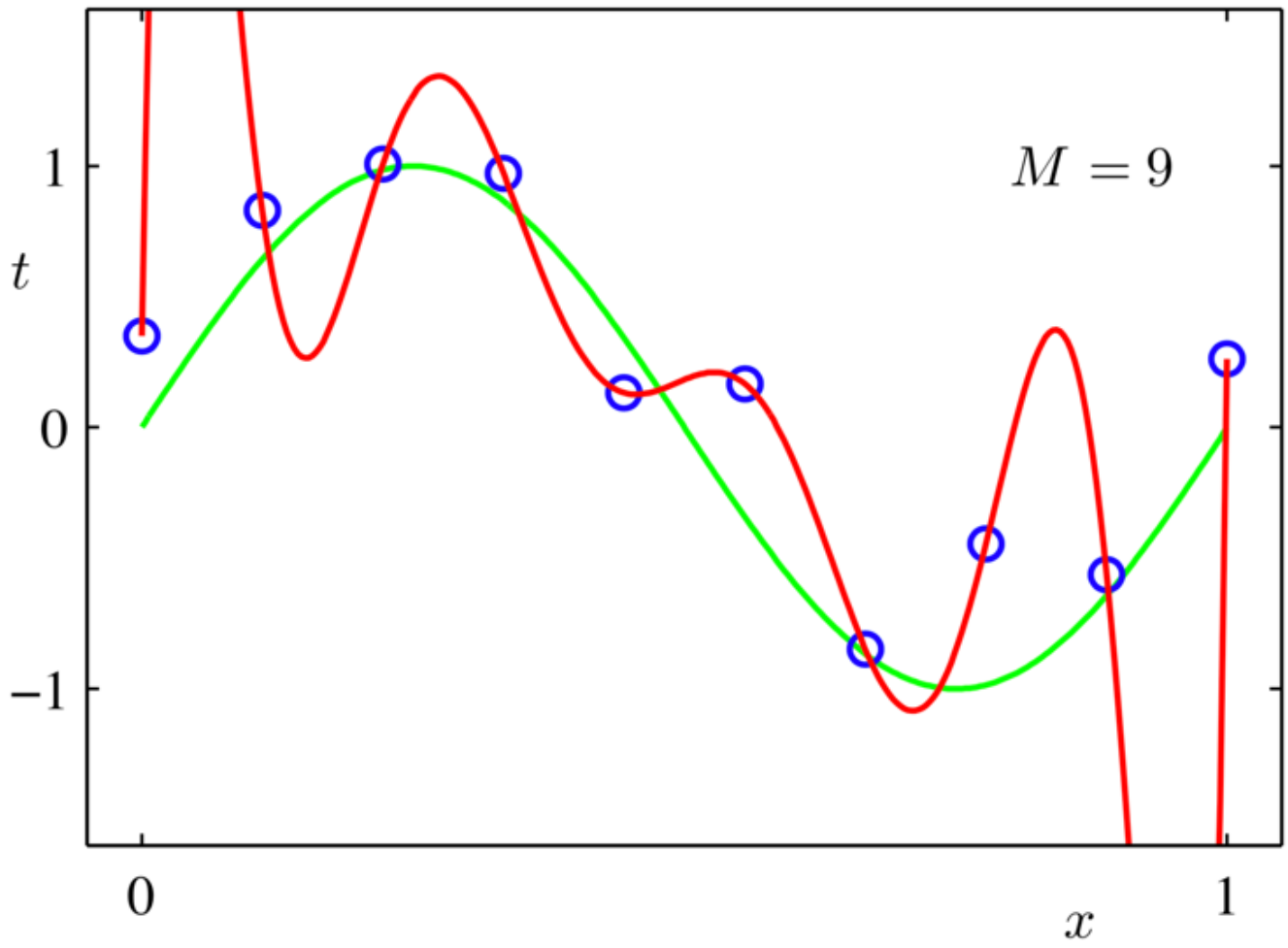
$$h_\theta(x) = \theta_0 + \theta_1 x$$

Example of fitting a Polynomial Curve with Linear Regression



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

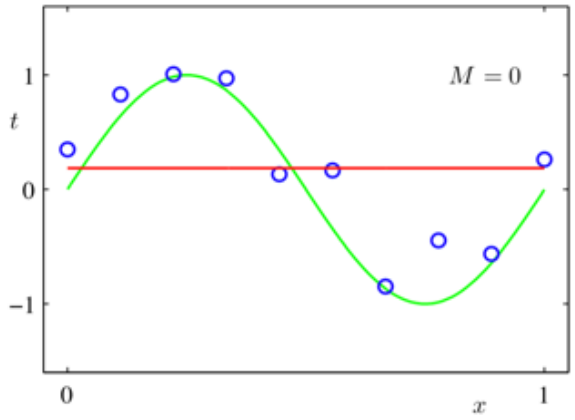
Example of fitting a Polynomial Curve with Linear Regression



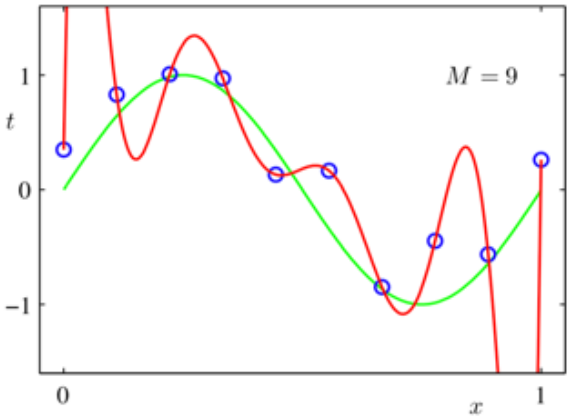
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \cdots + \theta_9 x^9$$

Underfitting and Overfitting

Underfitting: when the model is too simple

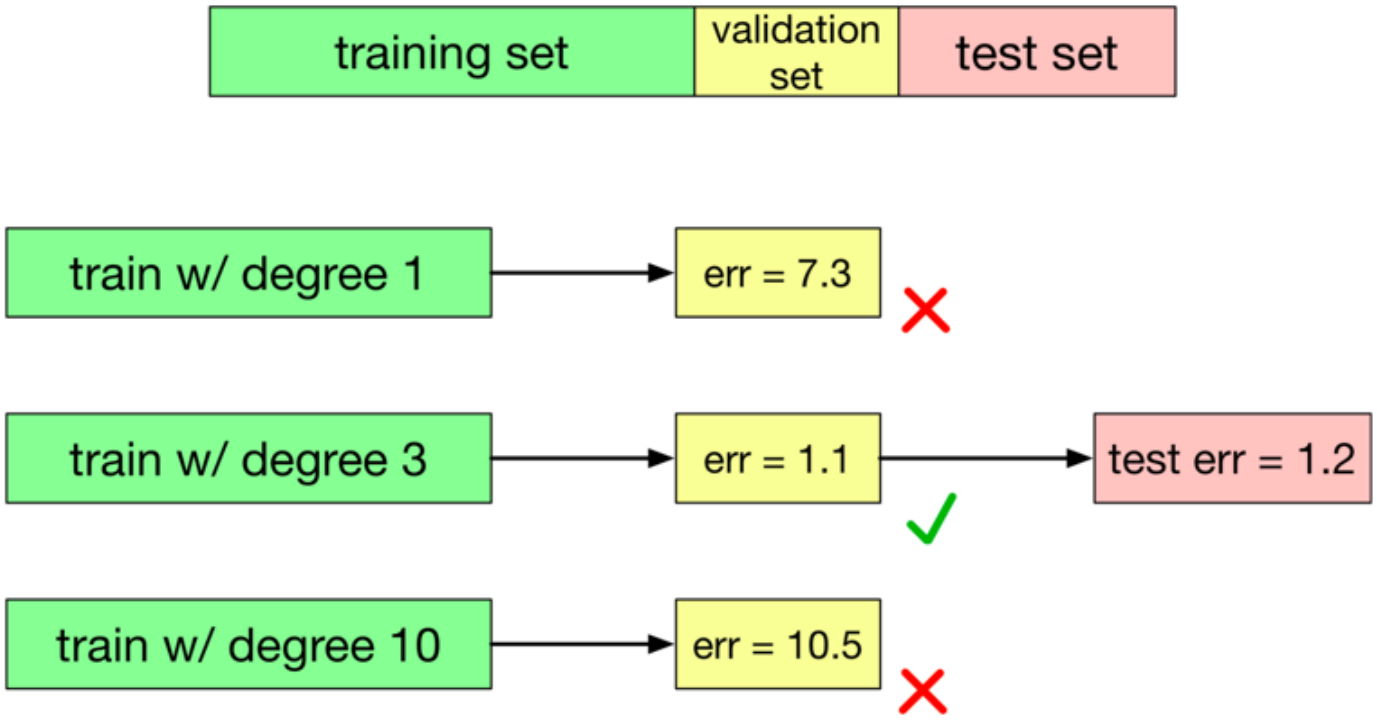


Overfitting: when the model is too complex



Model generalization – Training, validation and test split

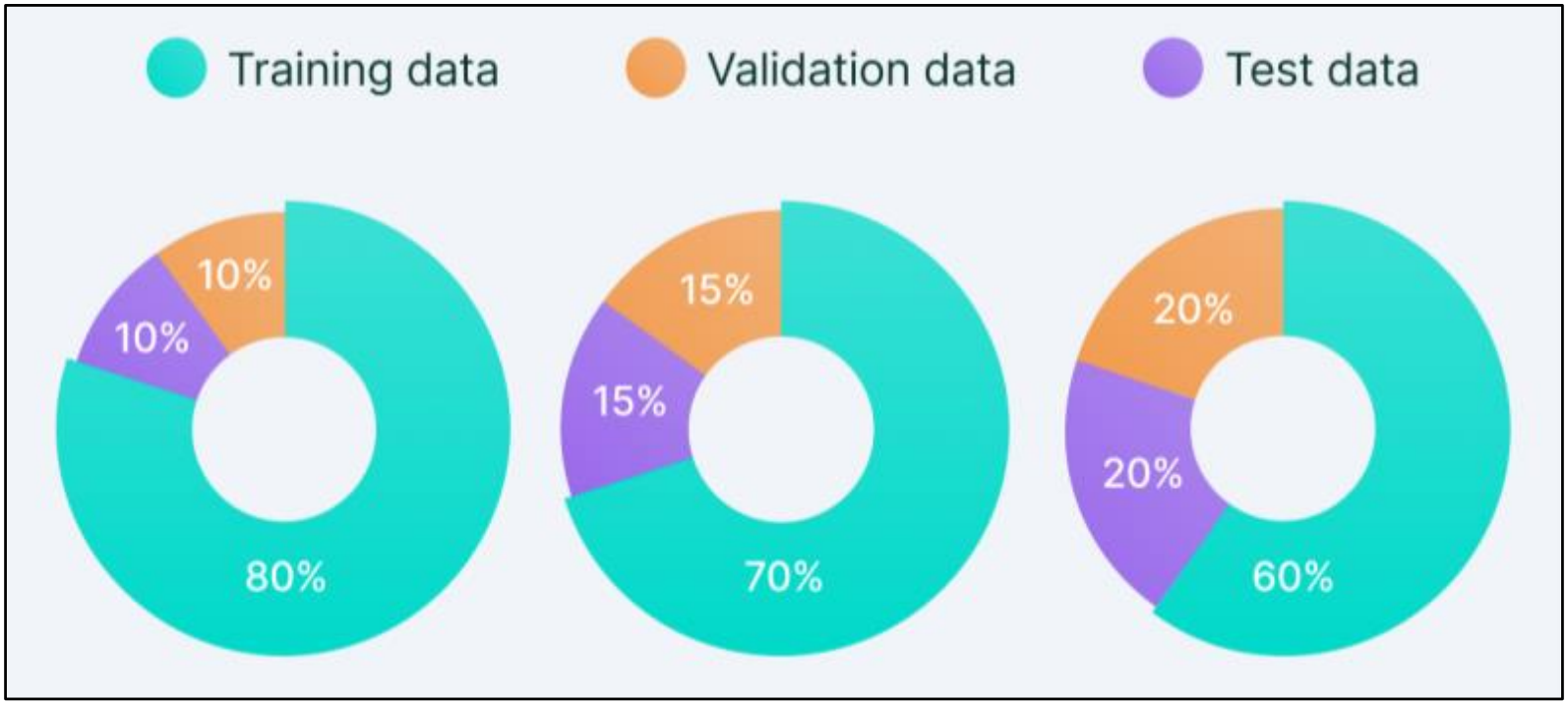
Models should be generalized to data they have not seen before.



Example of how to train, validate and test the models to ensure generalization.

Question: What should be a good ratio between training, validation and test set?

Model generalization – Training, validation and test split

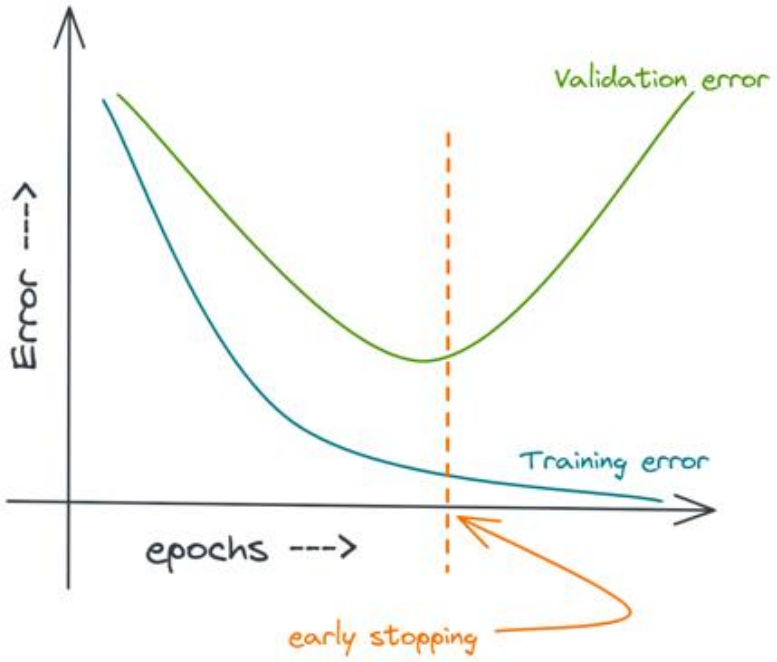


Question: Observing the model's accuracy solely on the training set might result in overfitting. With the use of **validation set**, how can we determine the **right moment to halt** the training process?

Model generalization – Training, validation and test split

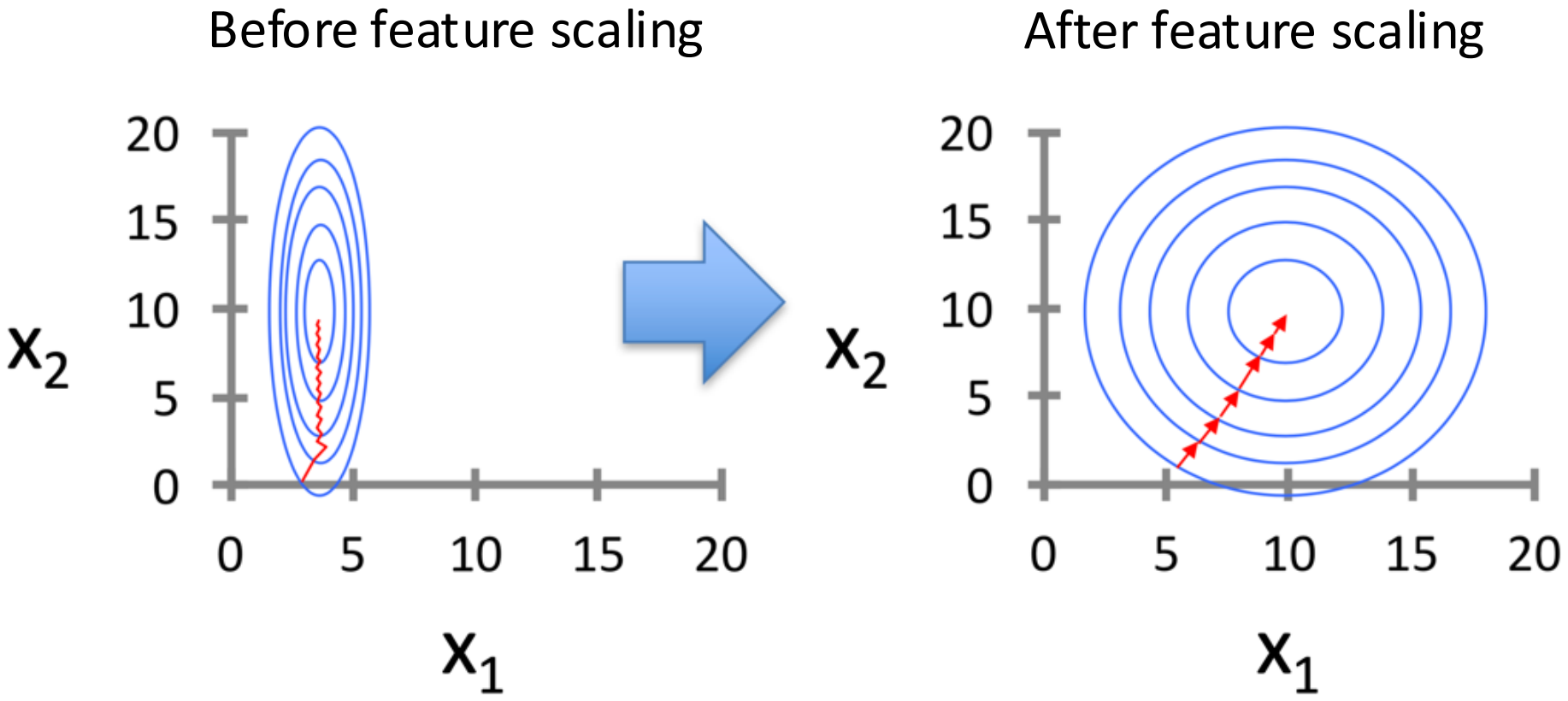
Question: Observing the model's accuracy solely on the training set might result in overfitting. How can we determine the right moment to halt the training process?

Answer: By looking at the model performance on validation set.



Feature Scaling

Feature scaling is a preprocessing technique used in machine learning to standardize the range of independent variables or features in the dataset.



Feature Scaling – Standardization

Standardization rescales features to have zero mean and unit variance.

- Let μ_j be the mean of feature j:

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)}$$

- Let s_j be the standard deviation of feature j:

$$s_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_j^{(i)} - \mu_j\right)^2}$$

- Replace each feature j with:

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$$

Question: How can we perform the transformation on training, validation and test set?

Feature Scaling – Mean normalization

Mean normalization rescales features to value range between $[-1,1]$.

- Let μ_j be the mean of feature j: $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)}$.
- Let $\max_i x_j$ be the maximum value of feature j of all data samples.
- Let $\min_i x_j$ be the minimum value of feature j of all samples.
- Replace each feature j with:

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\max_i x_j - \min_i x_j}$$

- This distribution will have the value range between $[-1,1]$.
- Example $x^{(1)} = [1,2]$ and $x^{(2)} = [3,4]$; We have $\mu_1 = 2$, $\mu_2 = 3$, $\max_i x_1 = 3$, $\min_i x_1 = 1$, $\max_i x_2 = 4$, $\min_i x_2 = 2$

$x_1^{(1)} = \frac{1-2}{3-1} = -\frac{1}{2}$; $x_2^{(1)} = \frac{2-3}{4-2} = -\frac{1}{2}$; $x_1^{(2)} = \frac{3-2}{3-1} = \frac{1}{2}$; $x_2^{(2)} = \frac{4-3}{4-2} = \frac{1}{2}$

Therefore after mean normaliaztion, $x^{(1)} = \left[-\frac{1}{2}, -\frac{1}{2}\right]$ and $x^{(2)} = \left[\frac{1}{2}, \frac{1}{2}\right]$

Feature Scaling – Min-max scaling

Min-max scaling rescales features to the value range between $[0,1]$.

- Let $\max_i x_j$ be the maximum value of feature j of all data samples.
- Let $\min_i x_j$ be the minimum value of feature j of all data samples.
- Replace each feature j with:

$$x_j^{(i)} = \frac{x_j^{(i)} - \min_i x_j}{\max_i x_j - \min_i x_j}$$

- Example $x^{(1)} = [1,2]$ and $x^{(2)} = [3,4]$;

We have $\max_i x_1 = 3, \min_i x_1 = 1, \max_i x_2 = 4, \min_i x_2 = 2$

$x_1^{(1)} = \frac{1-1}{3-1} = 0; x_2^{(1)} = \frac{2-2}{4-2} = 0; x_1^{(2)} = \frac{3-1}{3-1} = 1; x_2^{(2)} = \frac{4-2}{4-2} = 1$

Therefore after Min-max scaling, $x^{(1)} = [0,0]$ and $x^{(2)} = [1,1]$

Feature Scaling – Unit vector scaling

Unit vector scaling rescales each feature so that the entire feature vector has a length of 1 (unit vector).

- Let $\left\|x^{(i)}\right\| = \sqrt{\sum_{j=1}^{n}\left(x_j^{(i)}\right)^2}$ be the length of the entire feature vector $x^{(i)}$.
- Replace each feature j with:

$$x_j^{(i)} = \frac{x_j^{(i)}}{\left\|x^{(i)}\right\|}$$

- Example $x^{(1)} = [1,2]$ and $x^{(2)} = [3,4]$;

We have $\left\|x^{(1)}\right\| = \sqrt{5}, \left\|x^{(2)}\right\| = 5$

$x_1^{(1)} = \frac{1}{\sqrt{5}}$; $x_2^{(1)} = \frac{2}{\sqrt{5}}$; $x_1^{(2)} = \frac{3}{5}$; $x_2^{(2)} = \frac{4}{5}$. Therefore after Unit vector scaling, $x^{(1)} = \left[\frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}}\right]$, $x^{(2)} = \left[\frac{3}{5}, \frac{4}{5}\right]$, such that $\left\|x^{(1)}\right\| = \left\|x^{(2)}\right\| = 1$

Regularization

Question: With the same dataset, you trained and obtained two models as follows:

- Model 1: $h_\theta(x) = 1078 + 19254x - 3115x^2 - 982x^3$
- Model 2: $h_\theta(x) = 4 - 12x - 7x^2 + 13x^3$

Which model do you believe is more likely to exhibit better generalization to new data samples?

Regularization

Regularization is a technique used in machine learning and statistics to prevent overfitting and improve the generalization performance of a predictive model.

- **L1 Regularization (Lasso regression):** L1 regularization adds a penalty term proportional to the *absolute value* of the model's coefficients (θs) to the objective function.
- **L2 Regularization (Ridge regression):** L2 regularization adds a penalty term proportional to the *squared value* of the model's coefficients (θs) to the objective function.

In Ridge regression, L2 regularization imposes a stronger penalty (squared) on the magnitude of the model's coefficients.

Regularization – Objective function

Objective function of linear regression with L2 regularization:

$$J(\theta) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}_{\text{Model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{L2 regularization}}$$

- λ is the regularization parameter ($\lambda \geq 0$)
- No regularization on bias θ_0

Questions:

- What happens if λ is large?
- What happens if λ is small?

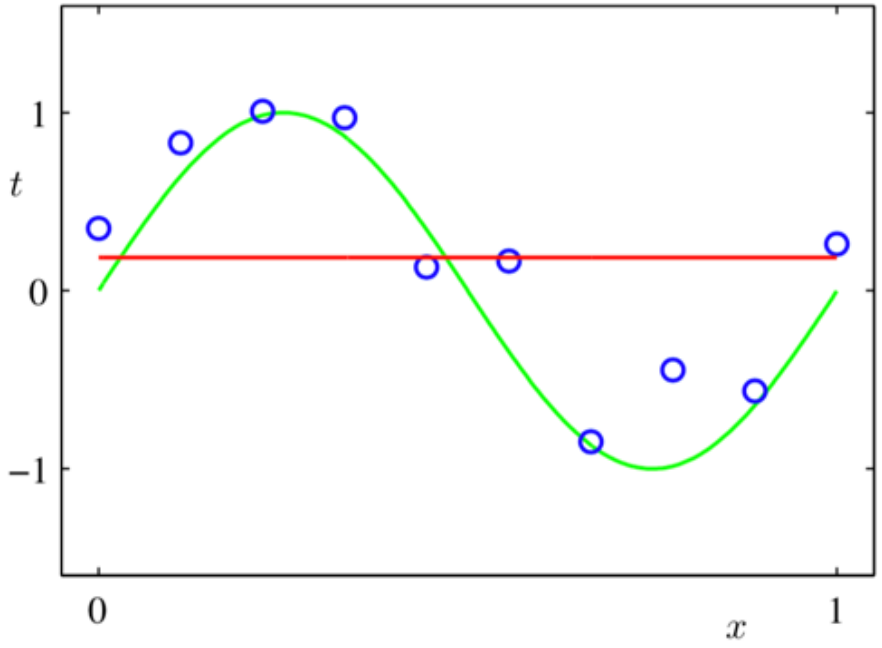
Regularization – Objective function

Objective function of linear regression with L2 regularization:

$$J(\theta) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}_{\text{Model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{L2 regularization}}$$

Questions:

- **What happens if λ is large?**
  - θ_j tends to be small
  - $h_\theta(x) \approx \theta_0$
  - **The model is …**



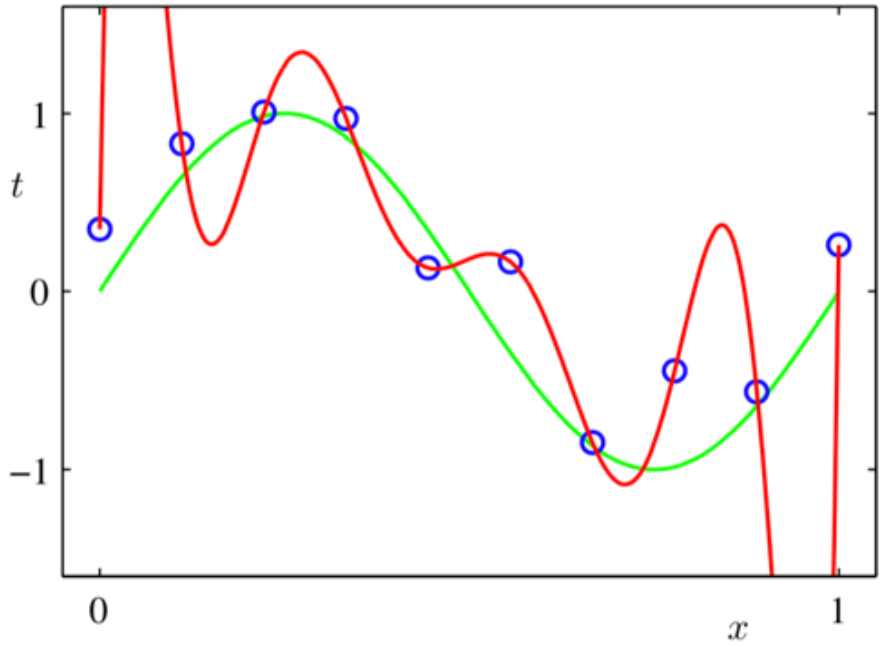
Regularization – Objective function

Objective function of linear regression with L2 regularization:

$$J(\theta) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}_{\text{Model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{L2 regularization}}$$

Questions:

- **What happens if λ is small?**
  - θ_j tends to be large
  - **The model is …**



Regularization – Parameter update

Objective function of linear regression with L2 regularization:

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

Fitting by solving:

$$\min_\theta J(\theta)$$

Parameter update:

$$\theta_j = \theta_j - \alpha\underbrace{\left(\frac{1}{n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} + \lambda\theta_j\right)}_{\text{Gradient } \frac{\partial}{\partial\theta_j}J(\theta)}$$

Summary

- Linear regression
- Loss Function
- Gradient descent
    - Batch Gradient Descent
    - Stochastic Gradient Descent
    - Mini-batch Gradient Descent
- Learning Rate
- Underfitting & Overfitting
- Generalization using Training, Validation and Test set
- Feature scaling
    - Standardization
    - Mean Normalization
    - Min-Max scaling
    - Unit Vector
- Regularization

Q&A

Thank you