

# Deep Network Development

Lecture #4

Viktor Varga  
Department of Artificial Intelligence, ELTE IK

# Requirements



The content of the slides marked by this symbol **will not be included in the exams / tests.**

# Last week - Supervised learning

**Given:** The training sample, a set of (input, label) pairs

$$\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$$

$$x \in X \subset \mathbb{R}^n, y \in Y \subset \mathbb{R}^k$$

**Task:** The estimation of the label (the expected output) from the input

I.e., we search for a (hypothesis-)function  $h_{\theta}$ , for which:

$$h_{\theta}(x) = \hat{y} \approx y$$

# Last week - Two main tasks in supervised learning

**Regression:** Continuous labels (The label set is infinite)

$$|Y| = \infty$$

**Example:** Number of cars or the age of a person

**Classification:** Discrete labels (The label set is finite)

$$|Y| < \infty$$

**Example:** Categorization of examples



- What is the profession of the person in the image?

# Last week - Example regression tasks

## Example tasks for regression:

$x_1$ : Population of a city   $y$ : Number of cars in the city

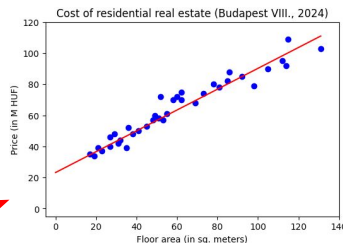
$x_1$ : Floor area of a flat  
 $x_2$ : Distance from city center    $y$ : Price of the flat

$x_1$ : Weight of a patient  
 $x_2$ : Age of a patient  
 $x_3$ : Sex of a patient    $y$ : Cholesterol levels of the patient

# Last week - Linear regression

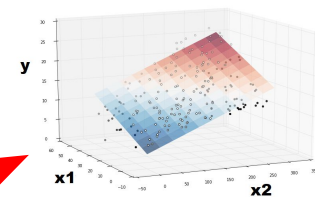
Hypothesis function - **one input variable**:

$$y \approx \hat{y} = h(x) = \theta_1 x + \theta_0$$



Hypothesis function - **two input variables**:

$$y \approx \hat{y} = h(x) = \theta_2 x_2 + \theta_1 x_1 + \theta_0$$



Hypothesis function - **n input variables**:

$$y \approx \hat{y} = h(x) = \theta_n x_n + \theta_{n-1} x_{n-1} + \dots + \theta_1 x_1 + \theta_0$$

(we cannot visualize the sample space for higher dimensionality...)

# Last week - Linear regression

Matrix form

$$\theta = \begin{bmatrix} \theta_0 \\ \dots \\ \theta_n \end{bmatrix} \in \mathbb{R}^n$$

$$\langle x^{(j)}, \theta \rangle = \sum_{i=0}^n x_i^{(j)} \theta_i$$

dot product

$$x = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad \langle x^{(1)}, \theta \rangle = \hat{y}^{(1)} \approx \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

$$h(x) = X\theta = \hat{y} \approx y$$

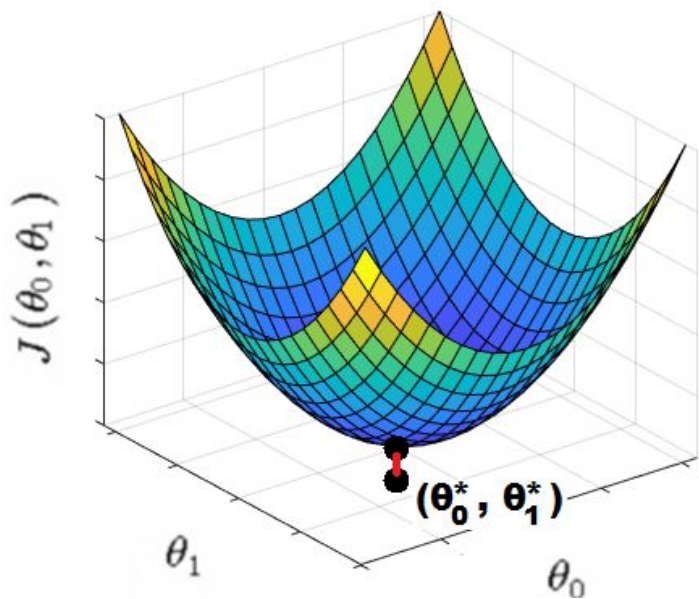
`y_pred = np.dot(X, theta)`

# Last week - Linear regression with least squares

**Goal:** Find parameters where the value of the loss function is minimal!

**Univariate case:**

one input variable, two parameters



$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{j=1}^m (\theta_1 x^{(j)} + \theta_0 - y^{(j)})^2$$

$$\theta_0^*, \theta_1^* = \underset{\theta_0, \theta_1}{\operatorname{argmin}} J(\theta_0, \theta_1)$$

# Last week - Linear regression with least squares

**Goal:** Find parameters where the value of the loss function is minimal!

**Multivariate case** in matrix form:  
**n** input variables, **n+1** parameters, **m** data points (examples)

$$J(\Theta) = \frac{1}{2m} \|X\theta - y\|_2^2 \quad X \in \mathbb{R}^{m \times n+1}, \theta \in \mathbb{R}^{n+1}$$

(we cannot visualize the loss function in higher dimensionality,  
**but it is still quadratic and convex**)

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} J(\Theta)$$

# Last week - Linear regression with least squares

## Gradient descent - multivariate case

repeat until convergence {

for  $i \leftarrow 1 \dots n$  {

$$\mathit{grad}_i = \frac{\partial}{\partial \theta_i} J(\theta)$$


}

for  $i \leftarrow 1 \dots n$  {

$$\theta_i = \theta_i - \alpha \mathit{grad}_i$$

}

}

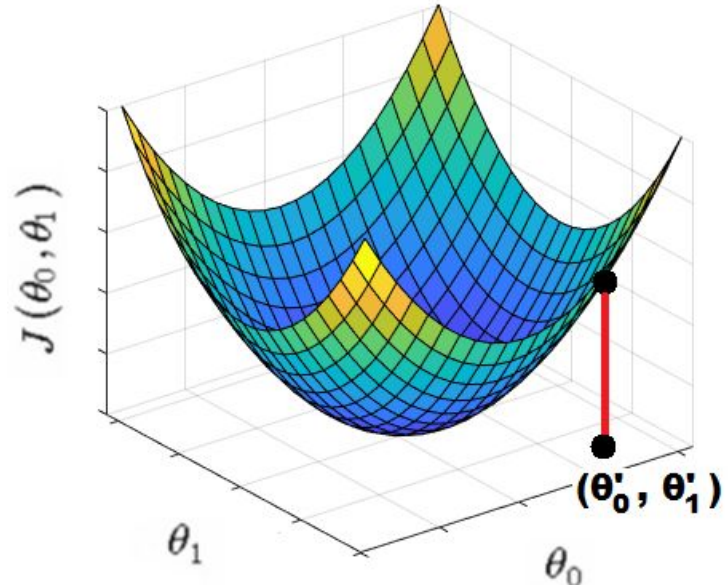
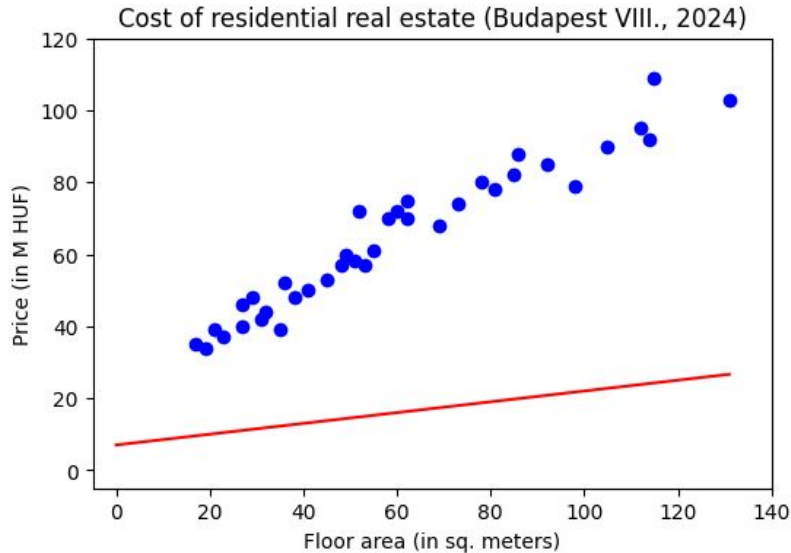

$$\frac{\partial}{\partial \theta_i} J(\theta) = \frac{1}{m} \sum_{j=1}^m (\theta_n x_n^{(j)} + \dots + \theta_0 x_0^{(j)} - y^{(j)}) x_i^{(j)}$$

The slope of the tangent in the  $\theta_i$  direction

(Visualization: univariate case)

# Last week - Linear regression with least squares

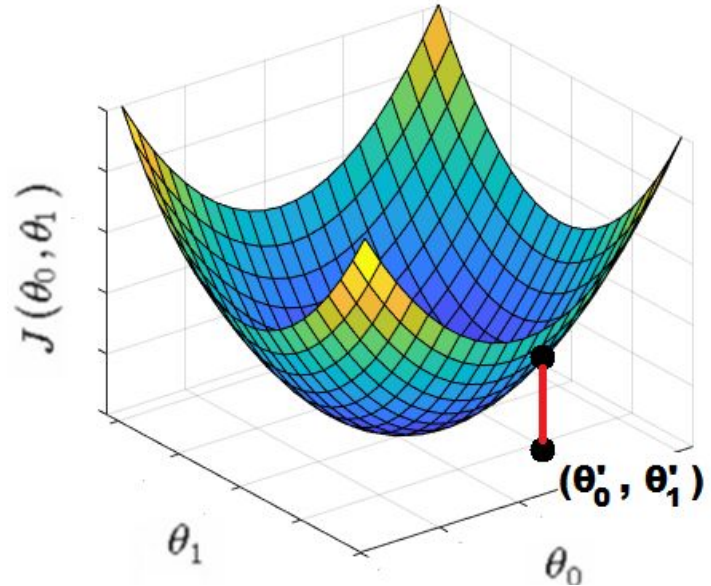
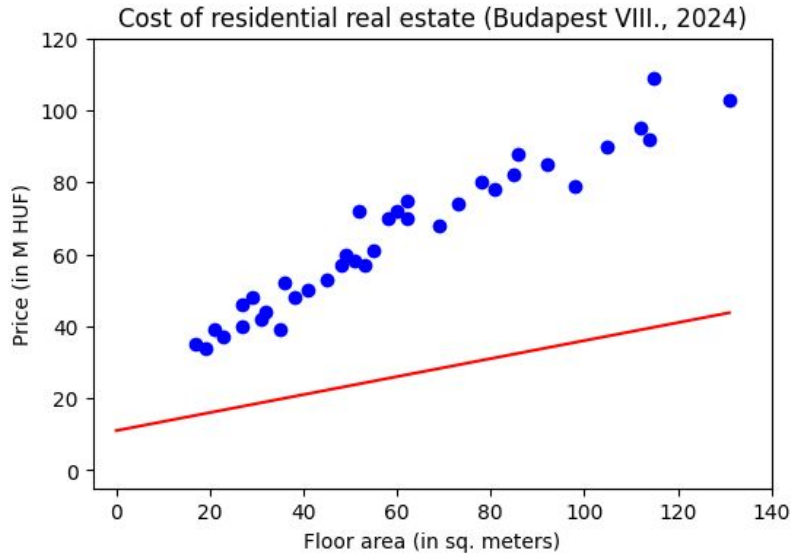
Applying gradient descent,  $T = 0$  (before taking the first step)



We can choose the initial parameters randomly.

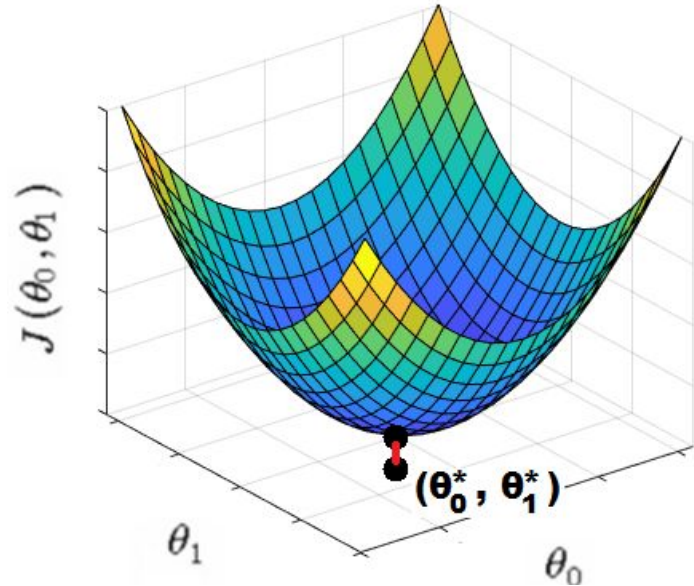
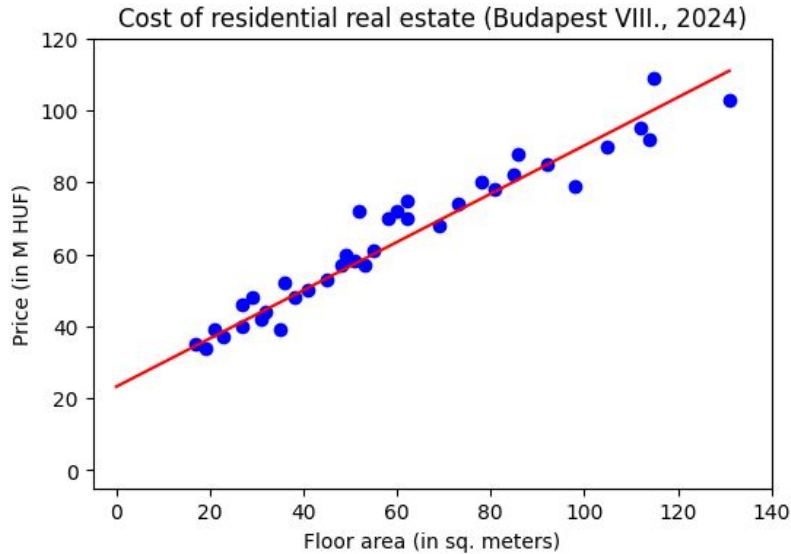
# Last week - Linear regression with least squares

Applying gradient descent,  $T = 1$



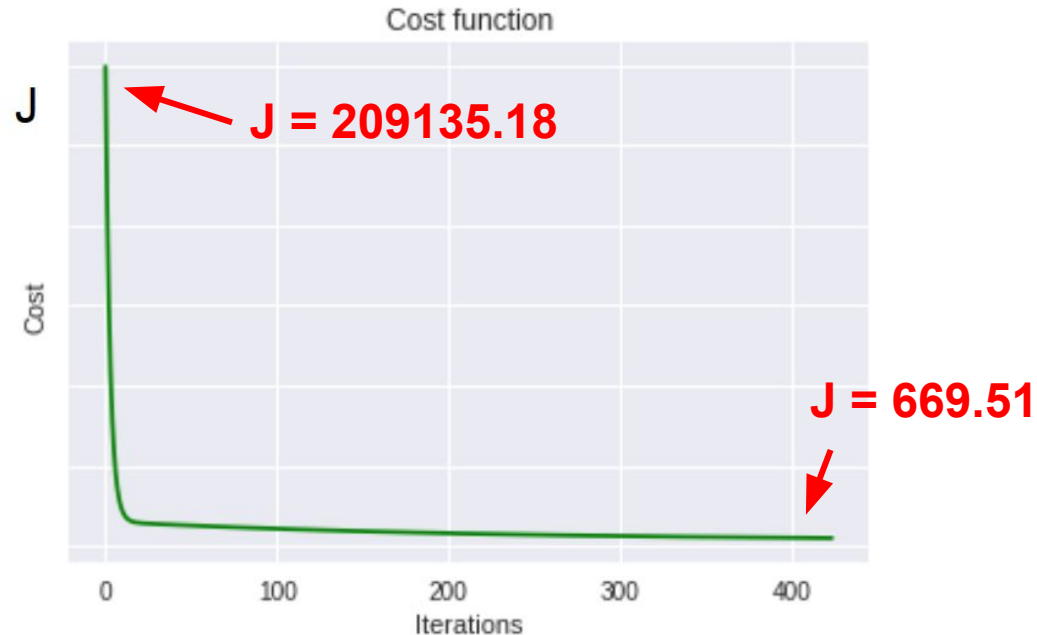
# Last week - Linear regression with least squares

Applying gradient descent,  $T = \langle \text{many} \rangle$



# Another solution to the least squares method

Changes in the loss value during the steps of the gradient descent



# Another solution to the least squares method

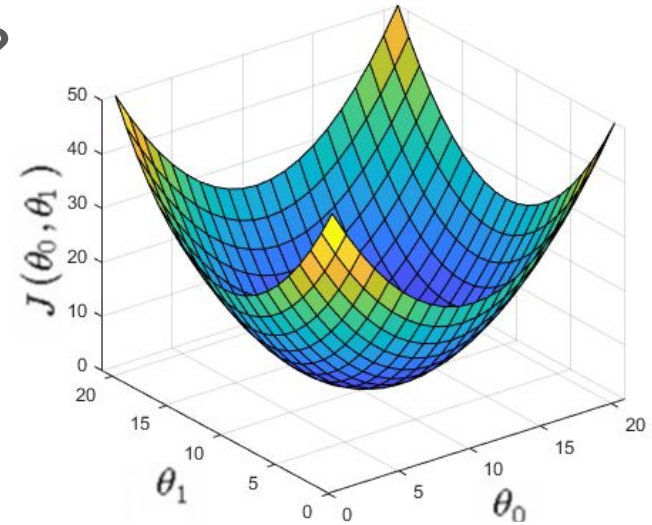
Changes in the loss value during the steps of the gradient descent



We need many iterations and we only approximate the optimal parameters.  
**Is there no better way?**

# Another solution to the least squares method

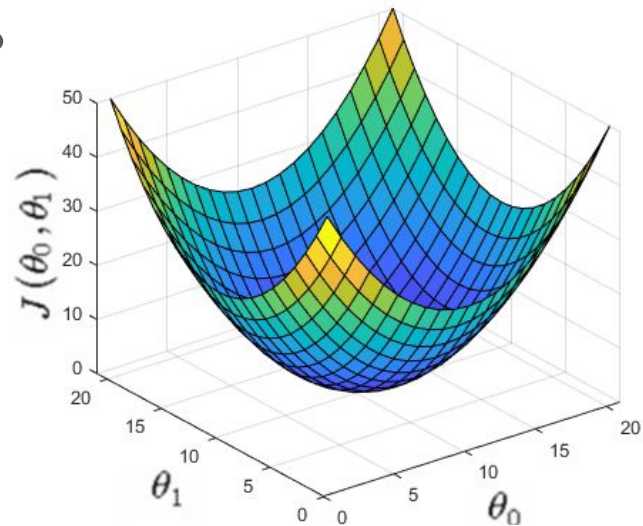
What do we know about the loss function?



# Another solution to the least squares method

What do we know about the loss function?

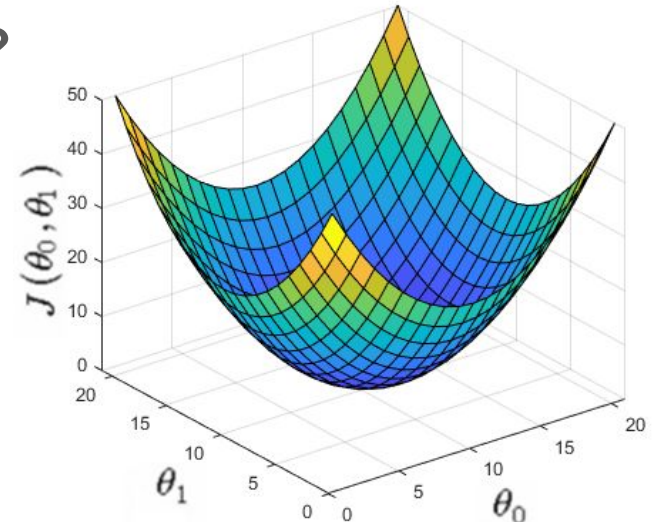
- **Quadratic** (second-degree polynomial, may be multivariate)
  - if the second-degree coefficient is positive, then it is **convex**
  - it has a **single** local (and also global) **minimum**



# Another solution to the least squares method

What do we know about the loss function?

- **Quadratic** (second-degree polynomial, may be multivariate)
  - if the second-degree coefficient is positive, then it is **convex**
  - it has a **single** local (and also global) **minimum**



**What can we say about the minimum point (the optimal parameters)?**

## Another solution to the least squares method

$$\forall i = 0 \dots n : \frac{\partial}{\partial \theta_i} J(\theta) = 0$$

In the minimum of the loss function,  
**all partial derivatives are zero.**

$$\frac{\partial}{\partial \theta_i} J(\theta) = \frac{1}{m} \sum_{j=1}^m (\theta_n x_n^{(j)} + \dots + \theta_0 x_0^{(j)} - y^{(j)}) x_i^{(j)} = 0$$

## Another solution to the least squares method

$$\forall i = 0 \dots n : \frac{\partial}{\partial \theta_i} J(\theta) = 0$$

$$\frac{\partial}{\partial \theta_i} J(\theta) = \frac{1}{m} \sum_{j=1}^m (\theta_n x_n^{(j)} + \dots + \theta_0 x_0^{(j)} - y^{(j)}) x_i^{(j)} = 0$$

→ We can write the above equation for each parameter  $\theta_i$  (**n+1 equations**).

→ We are looking for the value of each parameter  $\theta_i$  (**n+1 unknowns**).

Derivation of the normal equation is omitted on this course...

## Another solution to the least squares method

The normal equation - an **exact solution** to the least squares problem

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

$$\theta = X^+ y$$

Moore-Penrose  
pseudoinverse



```
theta = np.matmul(np.linalg.pinv(X), y)
```

# Another solution to the least squares method

The normal equation - an **exact solution** to the least squares problem

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

**We can find the optimal parameters to a linear regression problem  
(with least squares) in a single step, by using the formula above!  
(X, y are known from the data)**

# Another solution to the least squares method

The normal equation - an exact solution to the least squares problem

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

The regularized (L2) least squares solution:

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

**In practice, we use the regularized version of the normal equation:**  
By choosing an appropriate lambda value, we ensure the stability of the solution (and thus a valid input to the inverse operation).

# Last week - Two main tasks in supervised learning

**Regression:** Continuous labels (The label set is infinite)

$$|Y| = \infty$$

**Example:** Number of cars or the age of a person

**Classification:** Discrete labels (The label set is finite)

$$|Y| < \infty$$



**Example:** Categorization of examples



- What is the profession of the person in the image?

# Last week - Example (binary) classification tasks

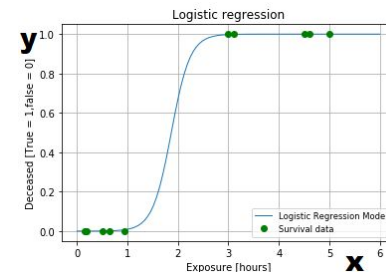
## Example tasks for (binary) classification:

**x**<sub>1</sub>: Duration of exposure to radiation       **y**: Did the person die?

**x**<sub>1</sub>: Population of a settlement  
**x**<sub>2</sub>: Annual number of tourists        **y**: Is the settlement a town?

**x**<sub>1</sub>: Weight of an animal  
**x**<sub>2</sub>: Number of hairballs coughed up  
**x**<sub>3</sub>: Degree of drooling        **y**: Is it a dog or a cat?

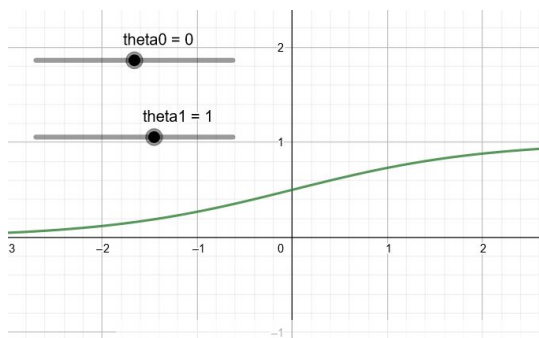
# Last week - Logistic regression



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}} = \hat{y}$$

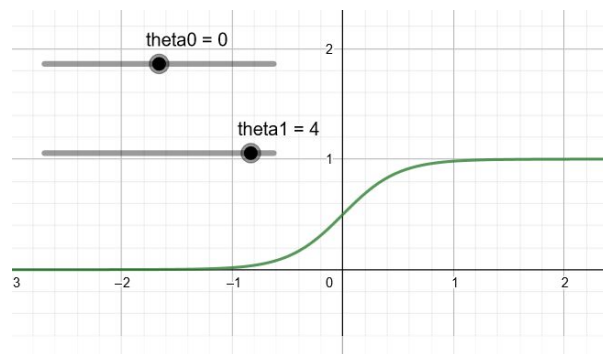
$$\theta_0 = 0$$

$$\theta_1 = 1$$



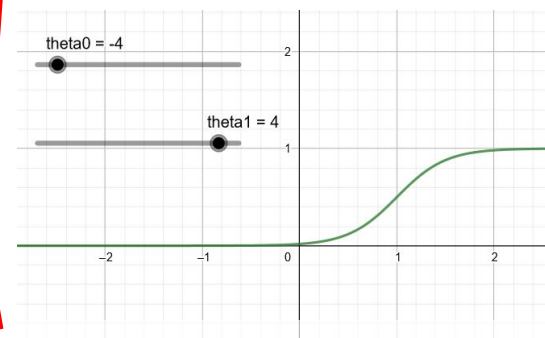
$$\theta_0 = 0$$

$$\theta_1 = 4$$



$$\theta_0 = -4$$

$$\theta_1 = 4$$

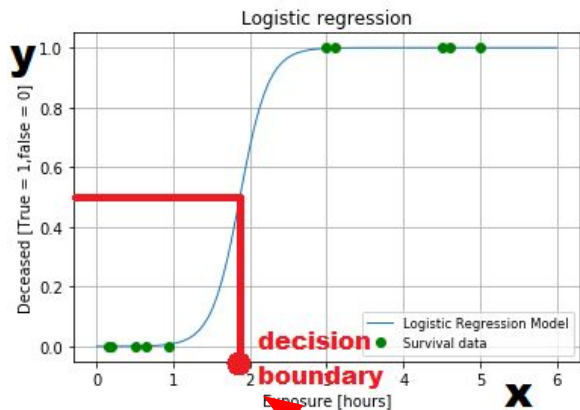


**Logistic regression hypothesis: Sigmoid  $\circ$  Linear regression**

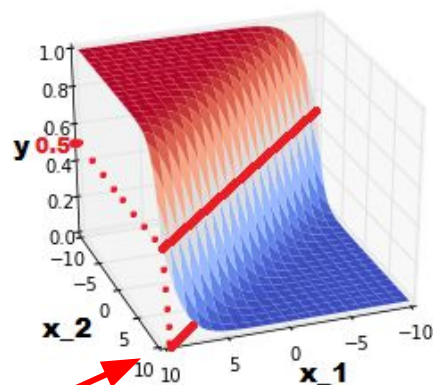
# Last week - Logistic regression

Logistic regression - **Decision boundary**  $\{x \mid \hat{y} = h(x) = 0.5\}$

## Univariate sample



## Bivariate sample



**decision boundaries**

# Last week - Logistic regression

**Loss function for logistic regression:** Logistic loss / Binary cross-entropy

$$J(\theta) = \frac{1}{m} \sum_{j=1}^m [-y^{(j)} \log(h_{\theta}(x^{(j)})) - (1 - y^{(j)}) \log(1 - h_{\theta}(x^{(j)}))]$$

$$h_{\theta}(x) = \frac{1}{1 + e^{-(\theta_1 x + \theta_0)}}$$

**Logistic loss** (using the logistic regression hypothesis) **is convex**

→ **a single local** (and global) **minimum**

→ **gradient descent will find the globally optimal parameters**



# Logistic regression in NumPy

```
def __sigmoid(self, z):  
    return 1 / (1 + np.exp(-z) + self.eps)  
  
h = self.__sigmoid(np.dot(X, self.theta))  
  
loss = np.mean(-y * np.log(h + self.eps) - \  
                (1 - y) * np.log(1 - h + self.eps))  
  
gradient = np.dot(X.T, (h - y)) / y.size  
self.theta -= self.lr * gradient
```

**sigmoid function (vectorized)**

**the label prediction (y hat)**

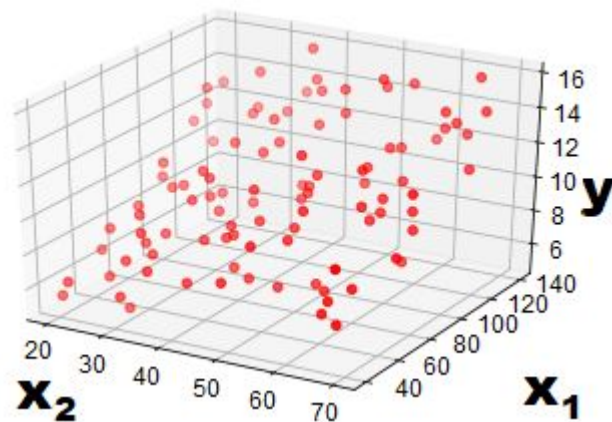
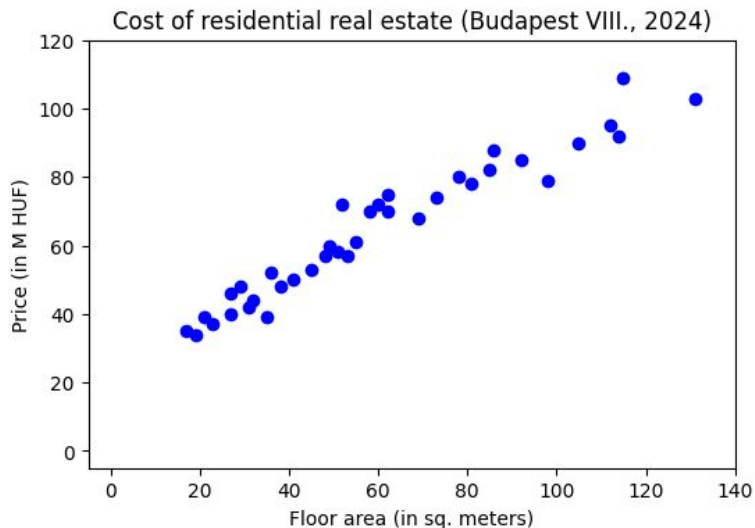
**loss value**

**a single step of gradient descent**

**(eps: to avoid division by zero)**

# Back to regression...

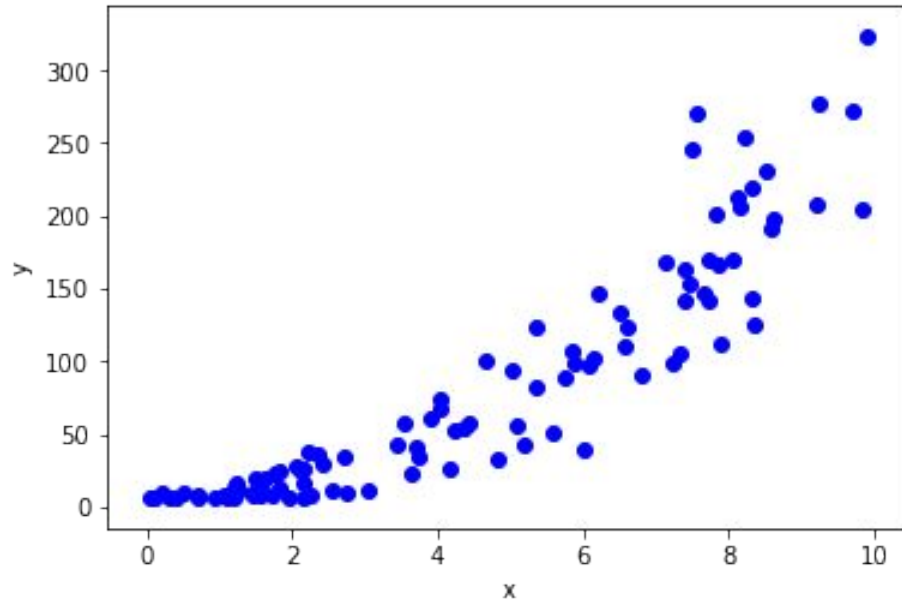
Linear regression worked well on a sample where the **linear combination of variables** closely approximated the label.





# Back to regression...

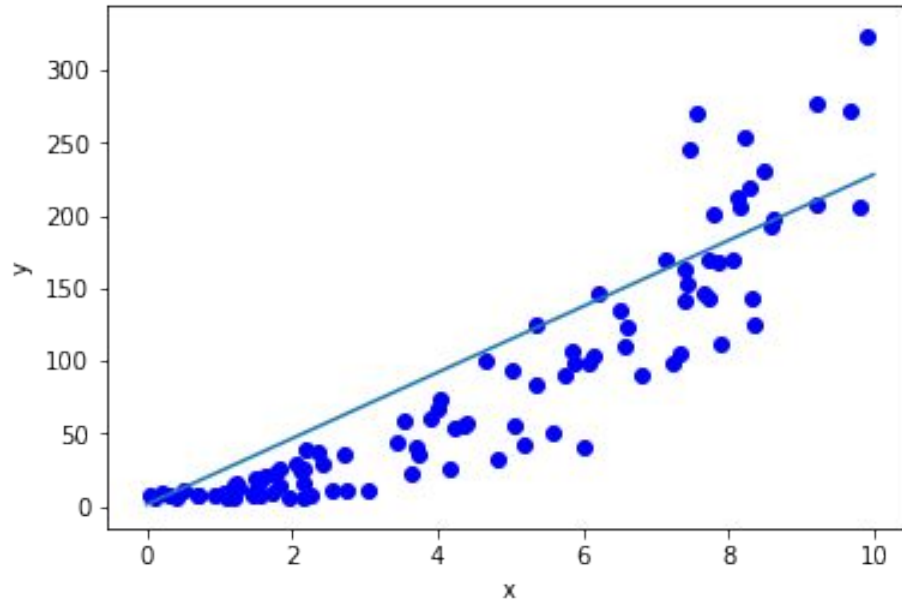
What can we do when a linear approximation is not suitable?



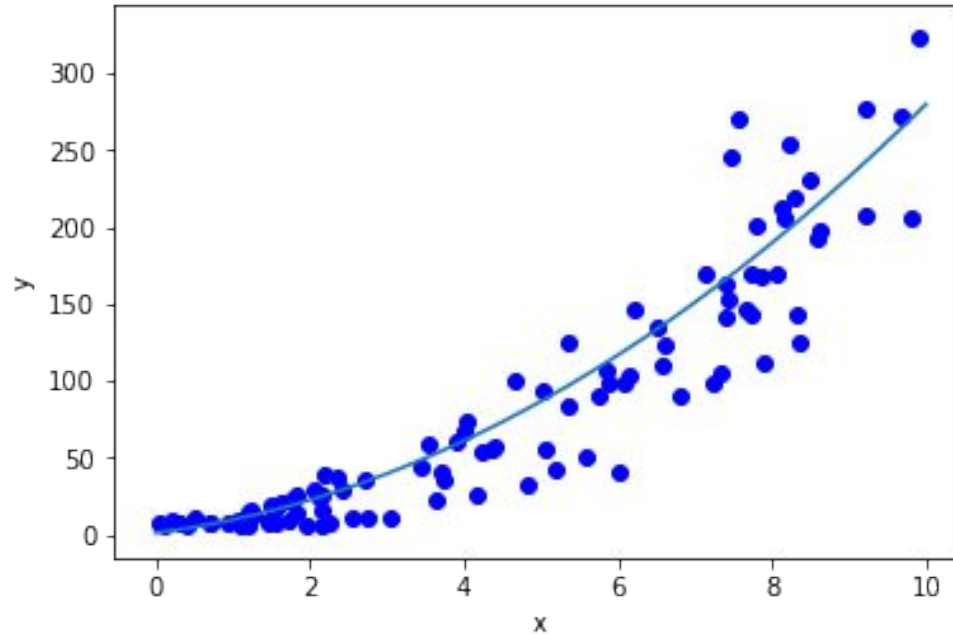


# Back to regression...

E.g., let's assume  $y \approx ax^2 + bx + c$ . In this case, linear regression does not provide very good results. **What can we do?**



# Polynomial regression





# Polynomial regression

## Hypothesis function

Univariate:  $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots$

Multivariate, e.g.,:  $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \dots$

**Loss function (MSE):**

$$J(\theta) = \frac{1}{2m} \sum_{j=1}^m \overbrace{(h_{\theta}(x^{(j)}))}^{\hat{y}^{(j)}} - y^{(j)} )^2$$



# Polynomial regression

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

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

## Solution using gradient descent:

The loss function is differentiable.

→ Gradient descent can be used to find good  $\theta$  coefficients.



# Polynomial regression

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

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

Now, we have exponents inside  
the hypothesis function (and thus the loss).  
**Is this a problem?**

## Solution using gradient descent:

The loss function is differentiable.

→ Gradient descent can be used to find good  $\theta$  coefficients.



# Polynomial regression

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

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

## Solution using gradient descent:

The loss function is differentiable.

→ Gradient descent can be used to find good  $\theta$  coefficients.

We do not raise  $\theta$  parameters to the power.

→ **J remains quadratic with respect to the  $\theta$  parameters!**



# Polynomial regression

## Solution:

Treat polynomial regression as if it was linear regression!

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

... is transformed into:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_{1'} + \theta_2 x_{2'} + \theta_3 x_{3'} + \theta_4 x_{4'} + \theta_5 x_{5'} + \dots$$



# Polynomial regression

## Solution in practice:

- Due to exponents, **gradients** associated with higher-order coefficients **can be on vastly different scale**.
- As we saw in multivariate linear regression, the gradient method is sensitive to this.

## Solution?



# Polynomial regression

## Solution in practice:

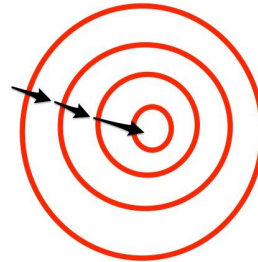
- Due to exponents, **gradients** associated with higher-order coefficients **can be on vastly different scale**.
- As we saw in multivariate linear regression, the gradient method is sensitive to this.

## Solution?

Without feature scaling



With feature scaling





# Polynomial regression

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

... is transformed into:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_{1'} + \theta_2 x_{2'} + \theta_3 x_{3'} + \theta_4 x_{4'} + \theta_5 x_{5'} + \dots$$

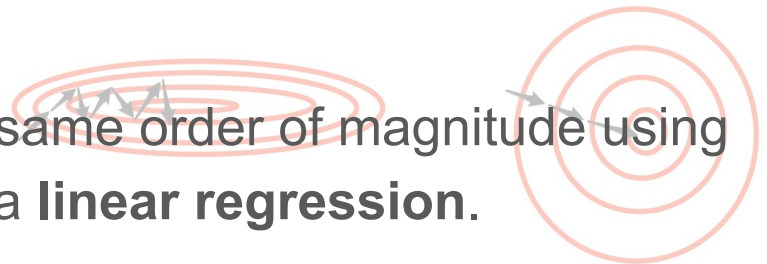
→ we **substitute** and calculate the powers of the variables in advance.

For example:  $x_{4'} := x_2^2$

Without feature scaling

With feature scaling

Finally, we bring the new variables to the same order of magnitude using feature scaling and solve the problem as a **linear regression**.





# Polynomial regression

**Univariate case:** Vandermonde matrix

$$X = \begin{bmatrix} 1 & x^{(1)} & (x^{(1)})^2 & \dots & (x^{(1)})^n \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x^{(m)} & (x^{(m)})^2 & \dots & (x^{(m)})^n \end{bmatrix} \in \mathbb{R}^{m \times n}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \dots \\ \theta_n \end{bmatrix} \in \mathbb{R}^n \quad y = \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

**Feature scaling:**

We treat  $x$ ,  $x^2$ , ...,  $x^n$  as separate variables and scale them independently of each other.

$$h(x) = X\theta = \hat{y} \approx y$$



# Polynomial regression

**Multivariate case, for example:**

$$x = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & (x_1^{(1)})^2 & (x_1^{(1)})^3 (x_2^{(1)})^2 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & x_2^{(m)} & (x_1^{(m)})^2 & (x_1^{(m)})^3 (x_2^{(m)})^2 & \dots \end{bmatrix} \in \mathbb{R}^{m \times n}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \dots \\ \theta_n \end{bmatrix} \in \mathbb{R}^n$$

$$y = \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

**Feature scaling:**

We treat  $x_1, x_2, x_1^2, \dots$  as separate variables and scale them independently of each other.

$$h(x) = X\theta = \hat{y} \approx y$$

# Splitting the sample

The life cycle of the model:

1. We train the model on the **training set**
2. We evaluate the model on the **test set**

The two sets must be **disjoint!**

# Splitting the sample

The life cycle of the model:

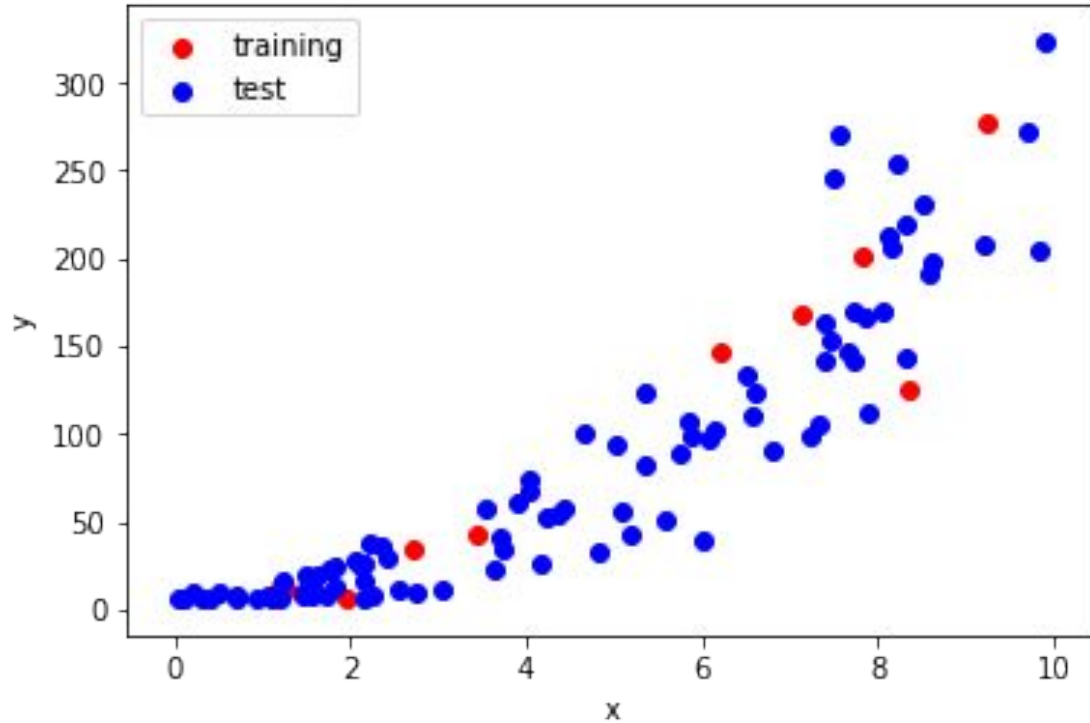
1. We train the model on the **training set**
2. We evaluate the model on the **test set**

The two sets must be **disjoint!**

**Do not use the test set for learning!**

We use the test set to estimate how our trained model will perform on new data points, not seen during training.

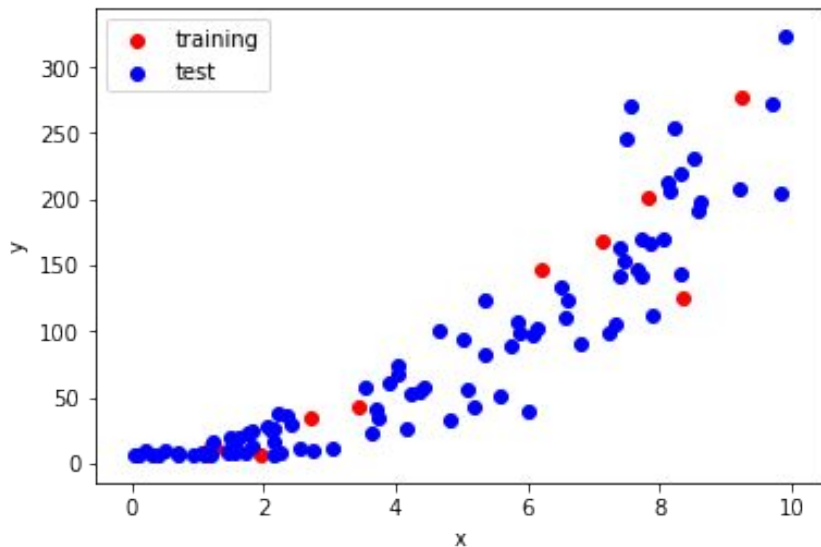
# Splitting the sample



# Observing the loss curve during training

Perform polynomial regression on the following sample!

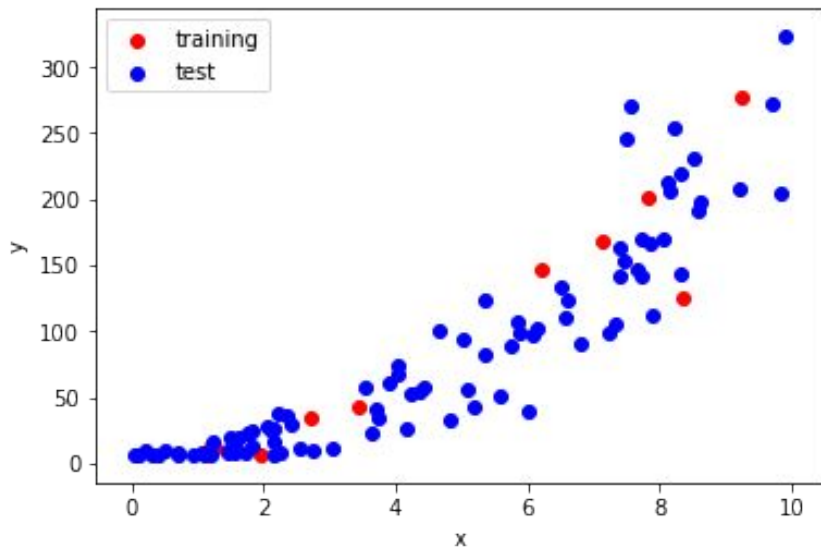
**What degree polynomial should we fit?**



# Observing the loss curve during training

Perform polynomial regression on the following sample!

What degree polynomial should we fit?



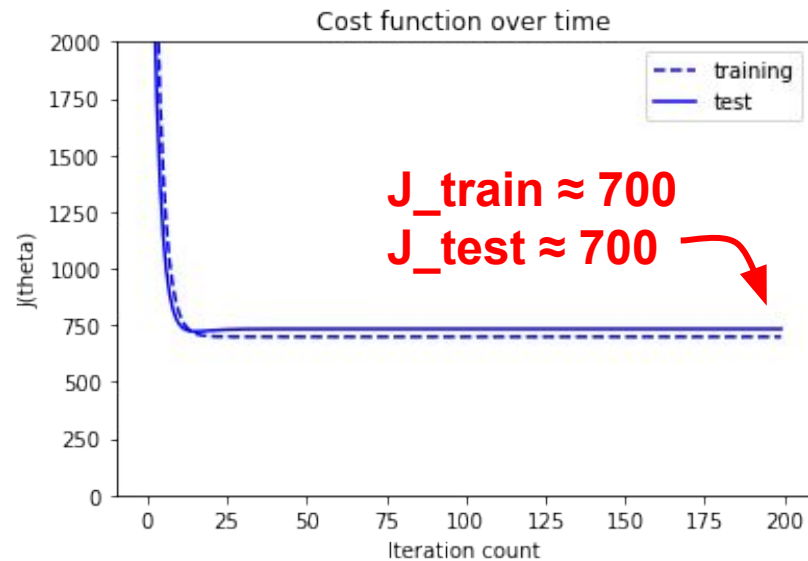
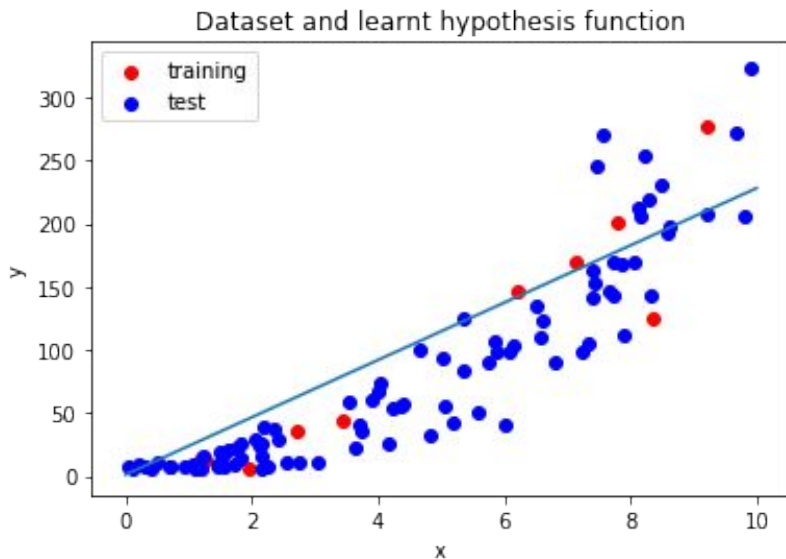
With a single variable, it is clear from looking at the graph that a second-degree polynomial fits the sample well.

Unfortunately, **we cannot visualize the data like this with more than two variables**. Therefore, without additional experimentation, we cannot tell what degree polynomial would be ideal.

# Observing the loss curve during training: **underfitting**

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

**The training and test losses are both (similarly) large.**

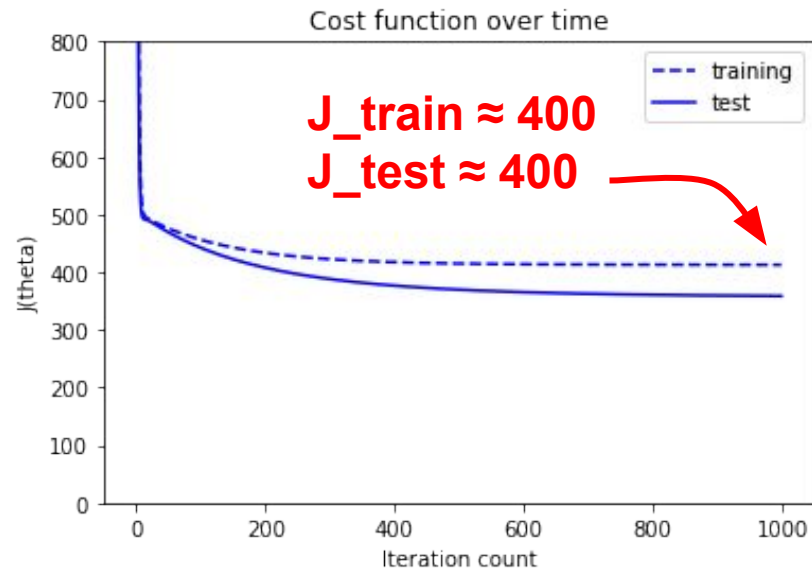
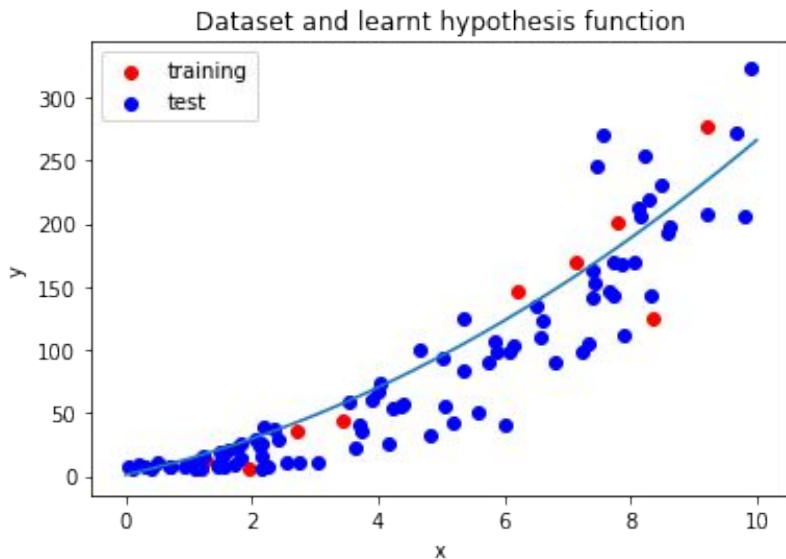


**“high bias, low variance”**

# Observing the loss curve during training: “just right”

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

**Both the training and the test losses are small and both decrease over time.**

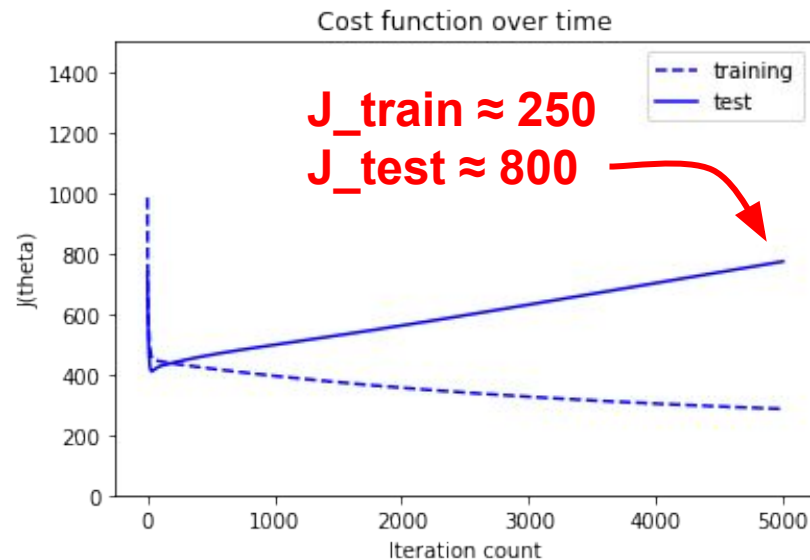
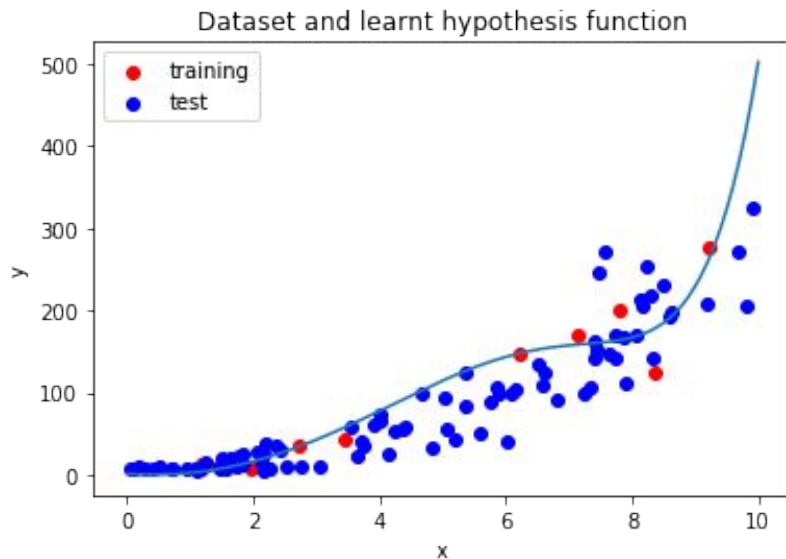


**“balancing bias & variance”**

# Observing the loss curve during training: **overfitting**

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_9 x^9$$

**The training loss decreases,  
but the test loss increases.**

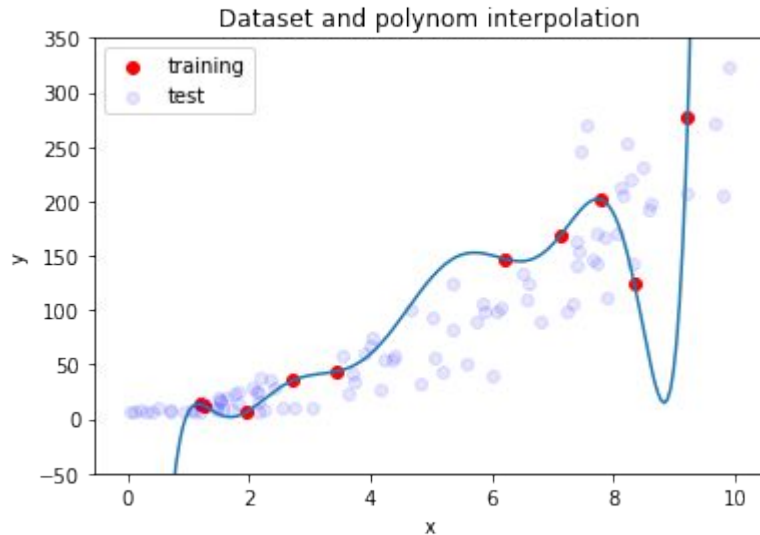


**“low bias, high variance”**



# Detour: Polynomial interpolation

With polynomial interpolation, we can even fit a polynomial exactly to the training set.



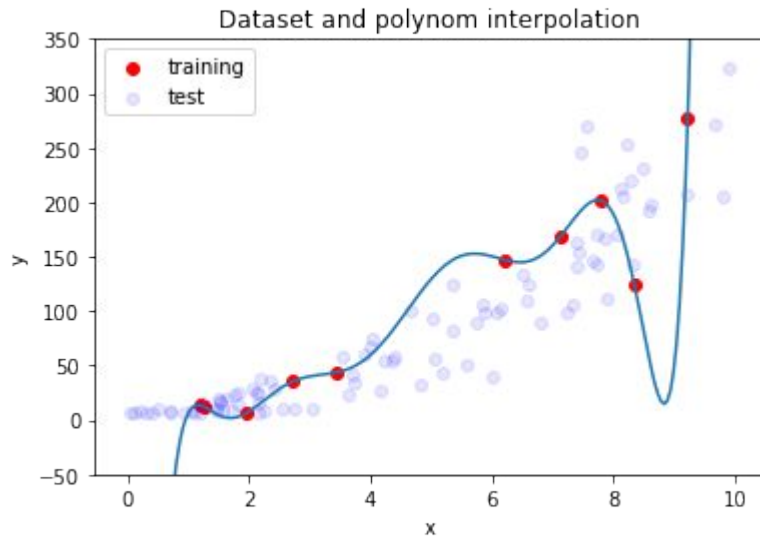
$$J_{\text{train}} = 0$$

$$J_{\text{test}} \approx 252507.67$$



# Detour: Polynomial interpolation

With polynomial interpolation, we can even fit a polynomial exactly to the training set. However, this is not useful in machine learning!



$J_{\text{train}} = 0$

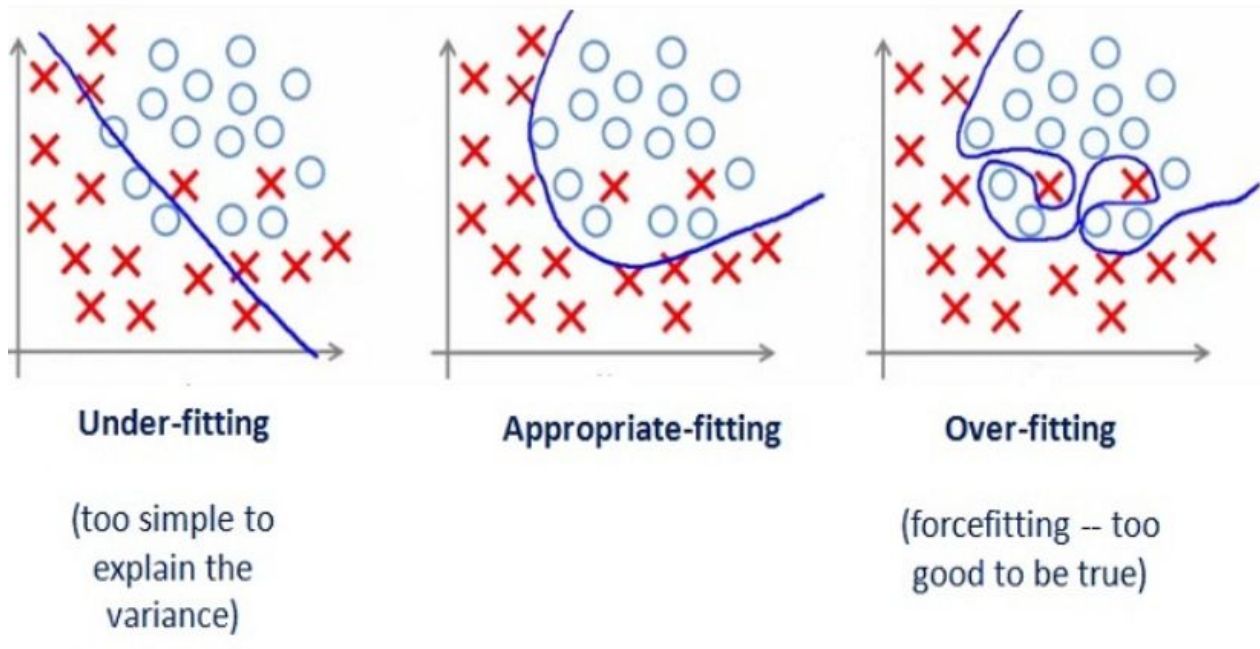
$J_{\text{test}} \approx 252507.67$

A ninth-degree polynomial can be fitted to ten points without any error.

If there is enough training data and the hypothesis function is sufficiently complex, a precise fit can be achieved, but this will be very far from the test data.

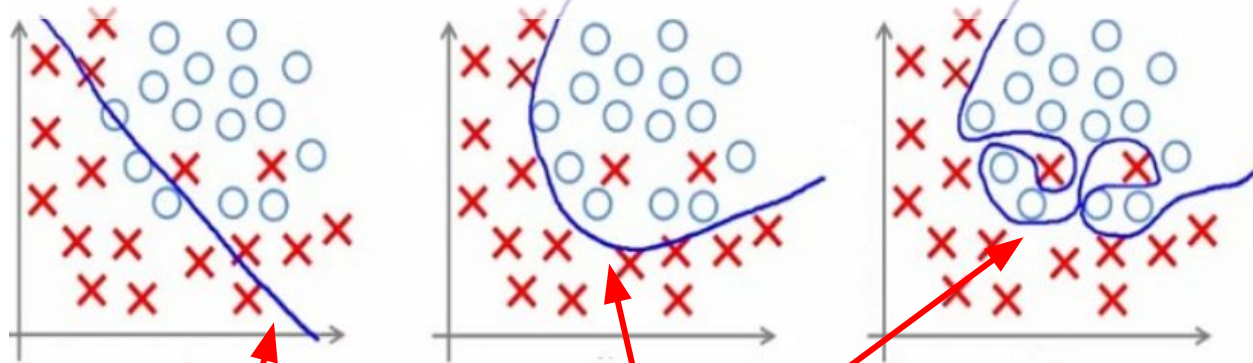
→ “Extreme overfitting”

# Under- and overfitting in case of classification



# Under- and overfitting in case of classification

**Classification with two input variables** (top view of the sample/hypothesis graph)



Under-fitting

Appropriate-fitting

Over-fitting

**The learned decision boundary in three cases:**

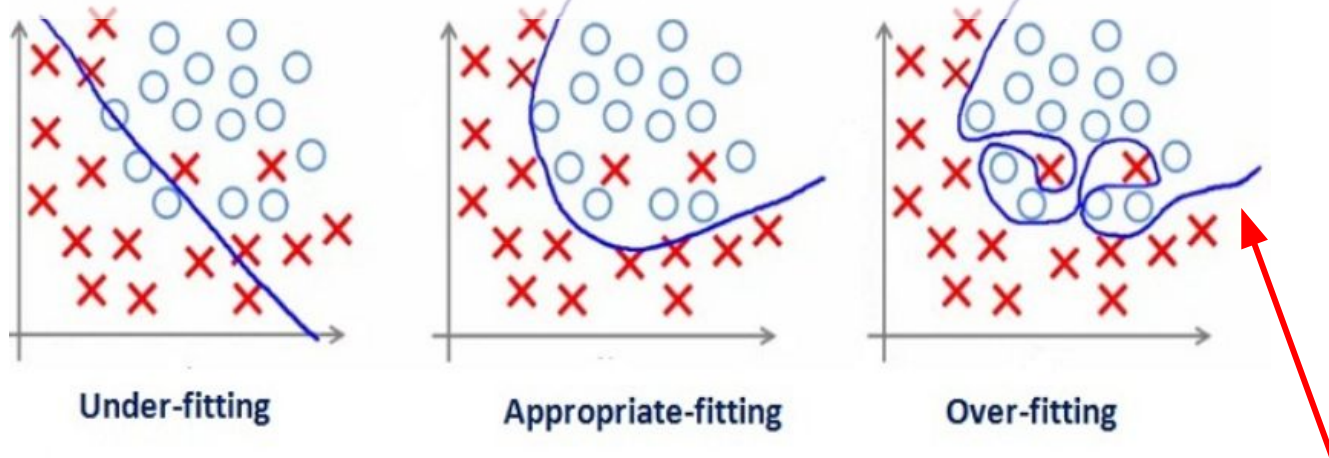
Overly simple, adequately expressive, and overly complex models.

(too simple to capture the variance)

(overfitting -- too good to be true)

# Under- and overfitting in case of classification

**Classification with two input variables** (top view of the sample/hypothesis graph)



Under-fitting

Appropriate-fitting

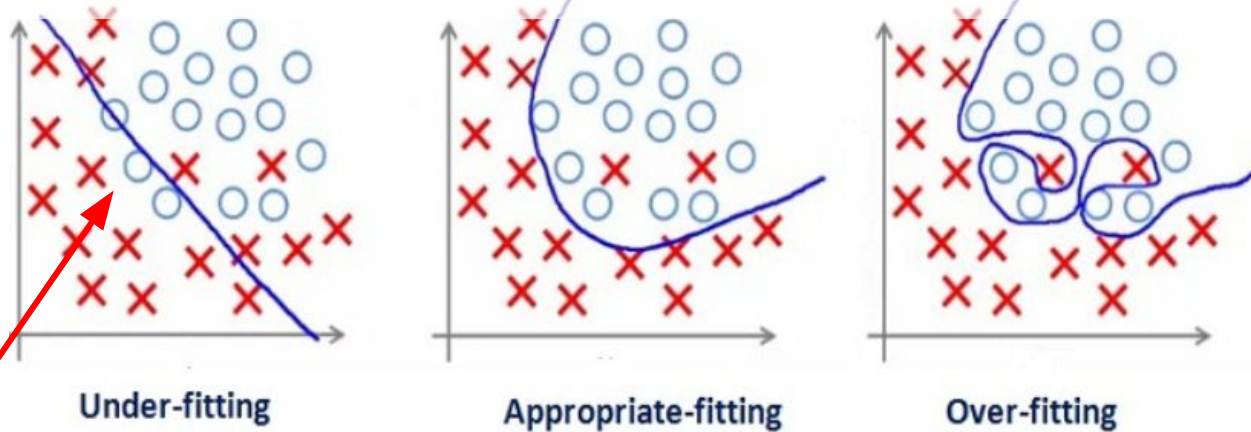
Over-fitting

(too simple to explain the variance)

An overly complex model is capable of "memorizing" individual training examples and precisely shaping the decision boundary around them (**significant overfitting**).

# Under- and overfitting in case of classification

**Classification with two input variables** (top view of the sample/hypothesis graph)



**Logistic regression learns a linear decision surface,**  
(too simple to explain the variance)  
thus overfitting is not expected  
(unless the training set size is minimal).  
(forcefitting -- too good to be true)

# Overfitting in deep neural networks

**Example:** Categorizing photos

If the deep network is powerful enough and has enough learnable parameters, it may learn the task in an undesirable way.



# Overfitting in deep neural networks

**Example:** Categorizing photos

If the deep network is powerful enough and has enough learnable parameters, it may learn the task in an undesirable way.

For example, it **"memorizes"** every single image in the training set with a specific pattern.  
...even a unique JPEG compression artifact!



# Avoiding underfitting and overfitting

Our goal is to ensure that our model does not underfit or overfit. Instead, we want to find the "just right" model!

**How can we avoid underfitting and overfitting?**

# Avoiding underfitting and overfitting

Our goal is to ensure that our model does not underfit or overfit. Instead, we want to find the "just right" model!

**How can we avoid underfitting and overfitting?**

Whether under- or overfit, the trained model will perform poorly when we try to estimate labels for new, unlabeled examples that were not seen during training — therefore, **we want to avoid these phenomena.**

# How to deal with **underfitting**?

**Underfitting:** The complexity of our model is too low to accurately approximate the labels from the input. The task is too difficult.

**Solution:** A more complex model is needed to reduce estimation error.

# How to deal with **underfitting**?

**Underfitting:** The complexity of our model is too low to accurately approximate the labels from the input. The task is too difficult.

**Solution:** A more complex model is needed to reduce estimation error.

**Example:** We can try to use linear regression for complex tasks, such as estimating the age of celebrities from photographs. However, the model is **severely underfitted**, as linear regression can only produce age estimates from a linear combination of pixel brightness, which is not a good approach for age estimation. A more complex model, such as a convolutional neural network, may be more suitable for the task.

# How to deal with **overfitting**?

**Overfitting:** The model is complex enough to accurately learn the specifics of individual elements of the training set, losing its ability to generalize.

**The overfitted model performs poorly on the test set.**

**How to deal with it?**

# How to deal with **overfitting**?

**Overfitting:** The model is complex enough to accurately learn the specifics of individual elements of the training set, losing its ability to generalize.

**The overfitted model performs poorly on the test set.**

**Solutions:**

- Use a simpler model (e.g., fewer parameters)!

# How to deal with **overfitting**?

**Overfitting:** The model is complex enough to accurately learn the specifics of individual elements of the training set, losing its ability to generalize.

**The overfitted model performs poorly on the test set.**

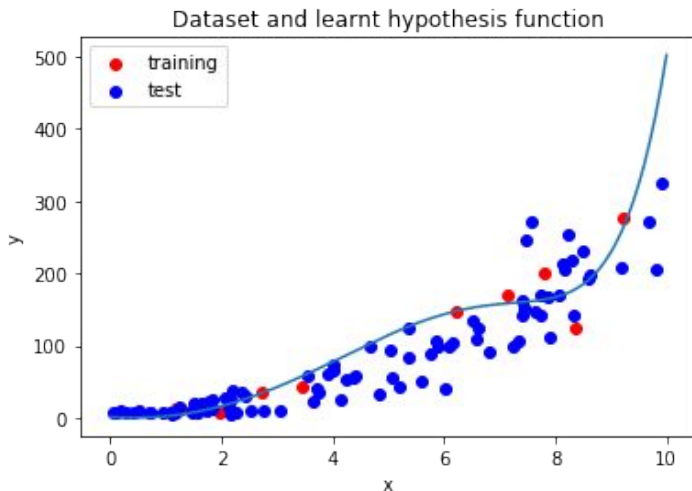
## **Solutions:**

- Use a simpler model (e.g., fewer parameters)!
- Obtain more training data!

# How to deal with **overfitting**?

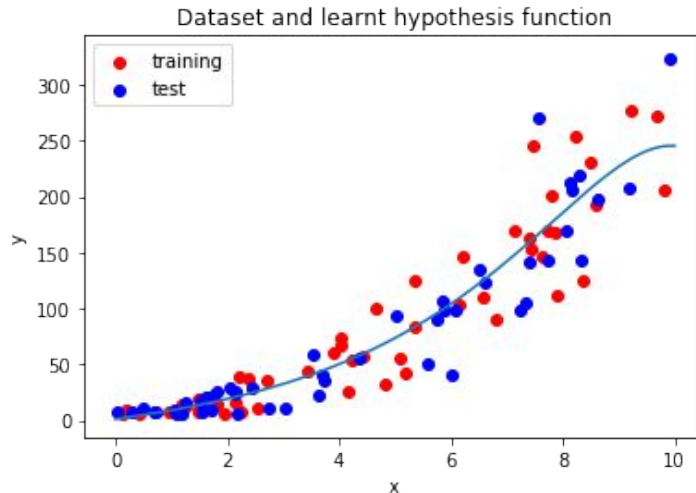
Obtain more training data!

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_9 x^9$$



$$|X_{train}| = 10$$

$$J_{train} \approx 250$$
$$J_{test} \approx 800$$



$$|X_{train}| = 50$$

$$J_{train} \approx 400$$
$$J_{test} \approx 400$$

# How to deal with **overfitting**?

**Obtain more training data!**

**Problem:** Typically, we have no access to enough labeled training data to prevent overfitting in a deep neural network with hundreds of millions of parameters.

**We need alternative methods...**

# How to deal with **overfitting**?

**Overfitting:** The model is complex enough to accurately learn the specifics of individual elements of the training set, losing its ability to generalize.

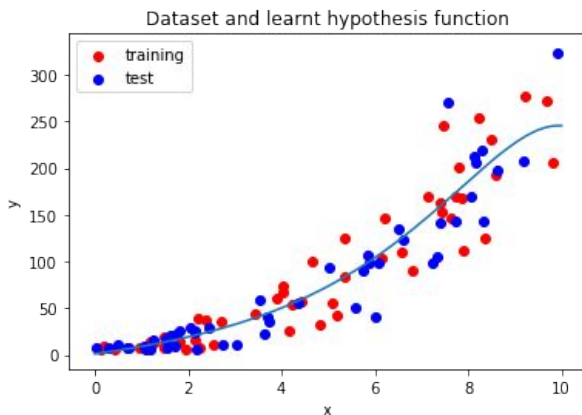
**The overfitted model performs poorly on the test set.**

## **Solutions:**

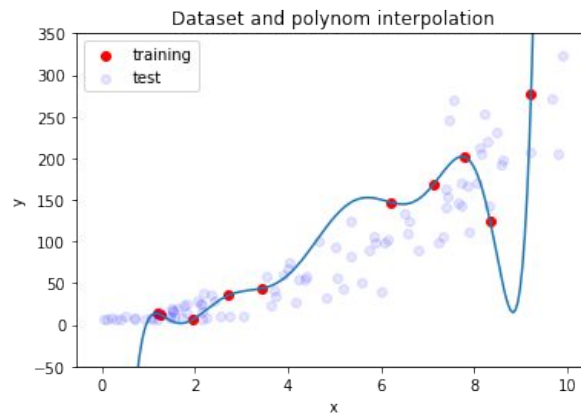
- Use a simpler model (e.g., fewer parameters)!
- Obtain more training data!
- Regularization methods (e.g., L2 regularization)

# How to deal with **overfitting**?

**Observation:** When fitting higher-degree polynomials, the **coefficients** (the  $\theta$  parameters) typically **increase** as the fit becomes more and more accurate.



$$\theta = [1., 7.46, 0.82, 0.06, 0.005, 0.0004, 0., 0., 0., 0.]$$



$$\theta = [[-2229., 7234., -9545., 6756., -2843., 743., -121., 12.05, -0.66, 0.015]]$$

# How to deal with **overfitting**?

**Observation:** When fitting higher-degree polynomials, the **coefficients** (the  $\theta$  parameters) typically **increase** as the fit becomes more and more accurate.

**Penalizing large coefficients** (parameters) **can help!**

**How?**

# How to deal with **overfitting**?

**Observation:** When fitting higher-degree polynomials, the **coefficients** (the  $\theta$  parameters) typically **increase** as the fit becomes more and more accurate.

**Let's include a penalty in the loss function:**

PI.: 
$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_9 x^9$$

$$J(\theta) = \frac{1}{2m} \sum_{j=1}^m (h(x)^{(j)} - y^{(j)})^2 + \lambda \sum_{i=1}^n \theta_i^2$$

# How to deal with **overfitting**?

**Observation:** When fitting higher-degree polynomials, the **coefficients** (the  $\theta$  parameters) typically **increase** as the fit becomes more and more accurate.

**Let's include a penalty in the loss function:**

PI.: 
$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_9 x^9$$

**MSE loss:** Penalizing the label estimation error

$$J(\theta) = \underbrace{\frac{1}{2m} \sum_{j=1}^m (h(x)^{(j)} - y^{(j)})^2}_{\text{MSE loss}} + \lambda \sum_{i=1}^n \theta_i^2$$

**L2-regularization term:**  
Penalizing large parameters

**When  $\lambda = 0$ :** The original model without regularization

**When  $\lambda$  is too high:** It is easier to minimize the loss by learning all zero parameters instead of solving the task...

# How to deal with **overfitting**?

**Observation:** When fitting higher-degree polynomials, the **coefficients** (the  $\theta$  parameters) typically **increase** as the fit becomes more and more accurate.

**Let's include a penalty in the loss function:**

PI.: 
$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_9 x^9$$

**MSE loss:** Penalizing the label estimation error

$$J(\theta) = \frac{1}{2m} \sum_{j=1}^m (h(x)^{(j)} - y^{(j)})^2 + \lambda \sum_{i=1}^n \theta_i^2$$

The **bias** parameter ( $\theta_0$ )  
**is not penalized.**

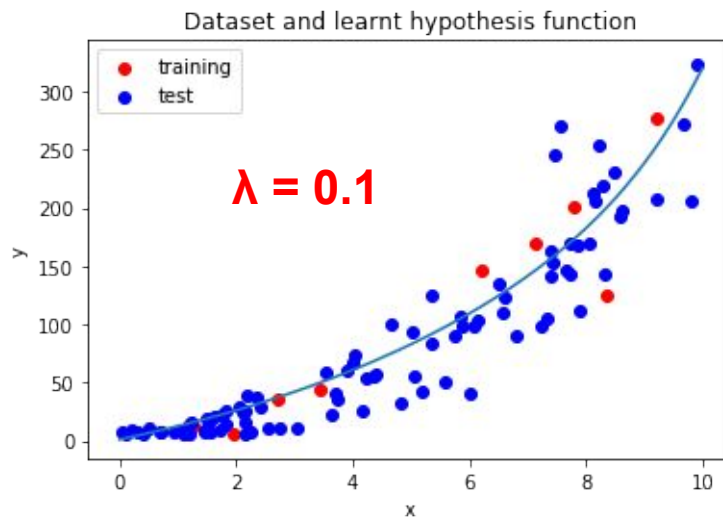
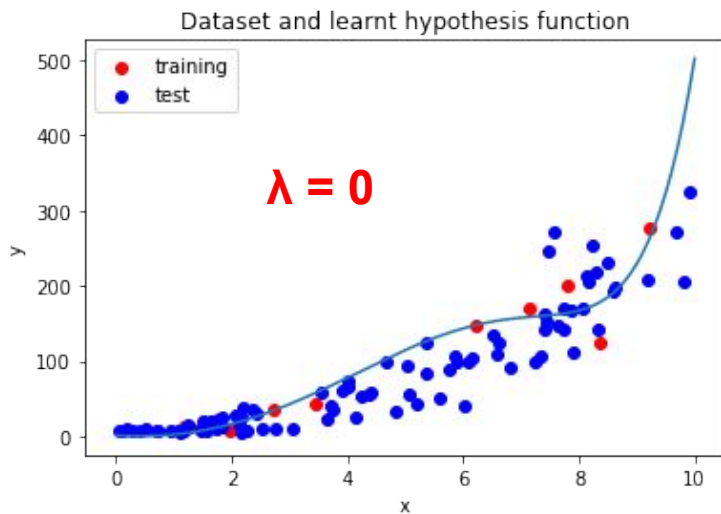
# How to deal with **overfitting**?

L2 regularization:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_9 x^9$$

$$|X_{train}| = 10$$

$$J(\theta) = \frac{1}{2m} \sum_{j=1}^m (h(x)^{(j)} - y^{(j)})^2 + \lambda \sum_{i=1}^n \theta_i^2$$



# How to deal with **overfitting**?

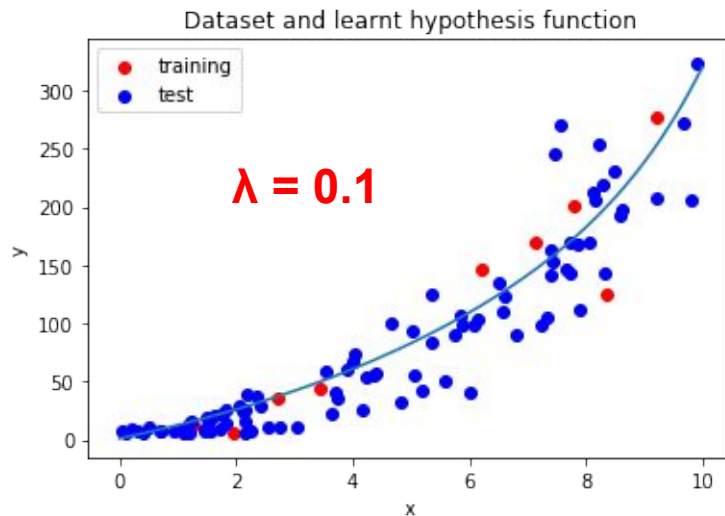
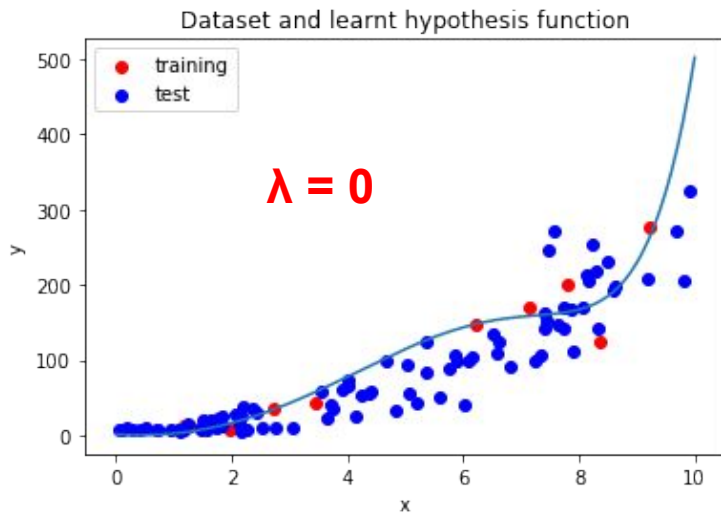
L2 regularization:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_9 x^9$$

$$|X_{train}| = 10$$

$$J(\theta) = \frac{1}{2m} \sum_{j=1}^m (h(x)^{(j)} - y^{(j)})^2 + \lambda \sum_{i=1}^n \theta_i^2$$

**Can also help when training data is scarce.**



# How to deal with **overfitting**?

L2 regularization:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_9 x^9$$

$$J(\theta) = \frac{1}{2m} \sum_{j=1}^m (h(x)^{(j)} - y^{(j)})^2 + \lambda \sum_{i=1}^n \theta_i^2$$

**L2 regularization is not only effective for polynomial regression.**

Experience shows that, in general, weights (parameters) also increase in the case of neural networks when overfitting occurs.

**L2 regularization is just one example.** There are many types of regularization methods that reduce overfitting by applying some kind of constraint during learning.

# How to deal with **overfitting**?

**Overfitting:** The model is complex enough to accurately learn the specifics of individual elements of the training set, losing its ability to generalize.

**The overfitted model performs poorly on the test set.**

## **Solutions:**

- Use a simpler model (e.g., fewer parameters)!
- Obtain more training data!
- Regularization methods (e.g., L2 regularization)
- Early stopping

# How to deal with **overfitting**?

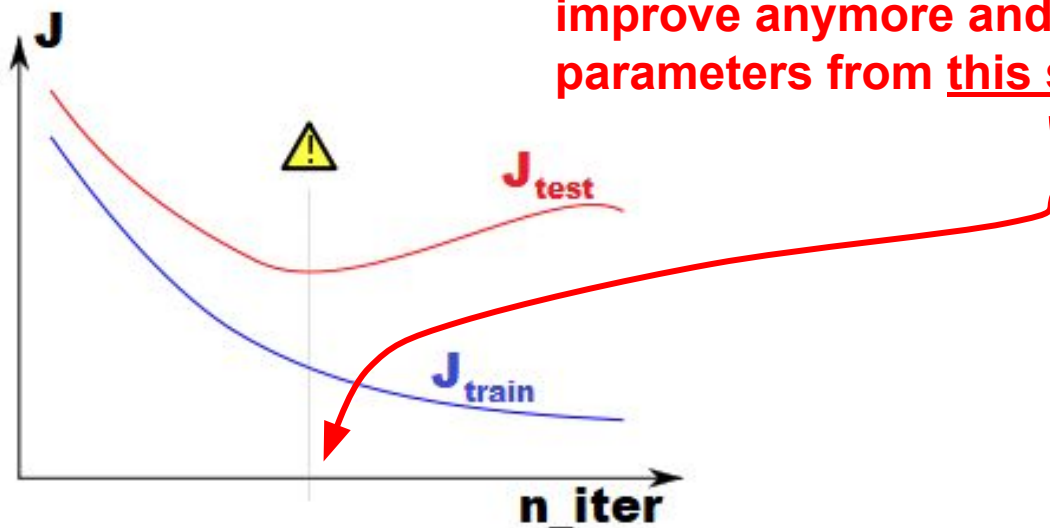
**Early stopping:** Overfitting often inevitably occurs after a certain number of iterations when training deep neural networks. **Early stopping** is generally an effective solution.



# How to deal with **overfitting**?

**Early stopping:** Overfitting often inevitably occurs after a certain number of iterations when training deep neural networks. **Early stopping** is generally an effective solution.

**Stop training if  $J_{\text{test}}$  does not improve anymore and use the parameters from this state**

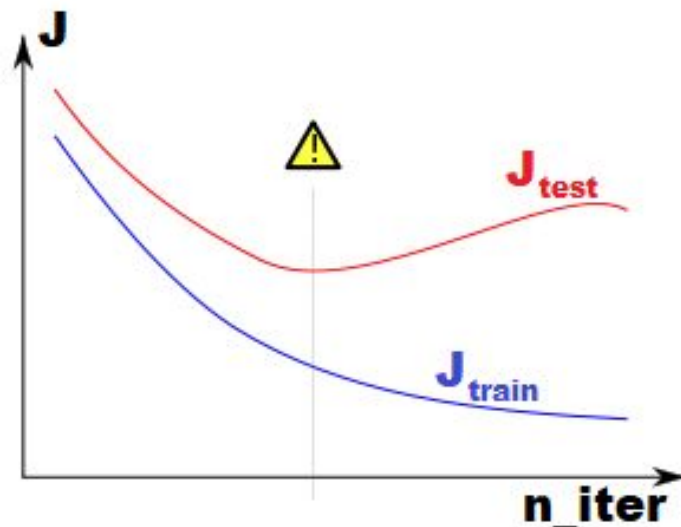


# How to deal with **overfitting**?

Early stopping:

**New loop condition:** Loop, while  $J_{\text{test}}$  keeps reducing.

```
repeat until convergence {  
  for  $i \leftarrow 1 \dots n$  {  
     $grad_i = \frac{\partial}{\partial \theta_i} J(\theta)$   
  }  
  for  $i \leftarrow 1 \dots n$  {  
     $\theta_i = \theta_i - \alpha grad_i$   
  }  
}
```



# How to deal with **overfitting**?

Early stopping:

```
repeat until convergence {  
  for i ← 1...n {  
     $grad_i = \frac{\partial}{\partial \theta_i} J(\theta)$   
  }  
  for i ← 1...n {  
     $\theta_i = \theta_i - \alpha grad_i$   
  }  
}
```

**New loop condition:** Loop, while  $J_{\text{test}}$  keeps reducing.

(We should be **patient** for a while and not stop on the first sign of a plateauing test loss!)

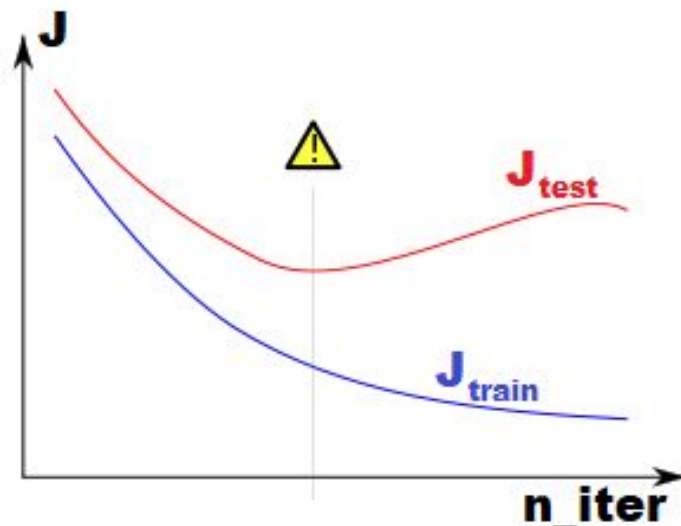


# How to deal with **overfitting**?

Early stopping:

**New loop condition:** Loop, while  $J_{\text{test}}$  keeps reducing.

```
repeat until convergence {  
  for  $i \leftarrow 1 \dots n$  {  
     $grad_i = \frac{\partial}{\partial \theta_i} J(\theta)$   
  }  
  for  $i \leftarrow 1 \dots n$  {  
     $\theta_i = \theta_i - \alpha grad_i$   
  }  
}
```

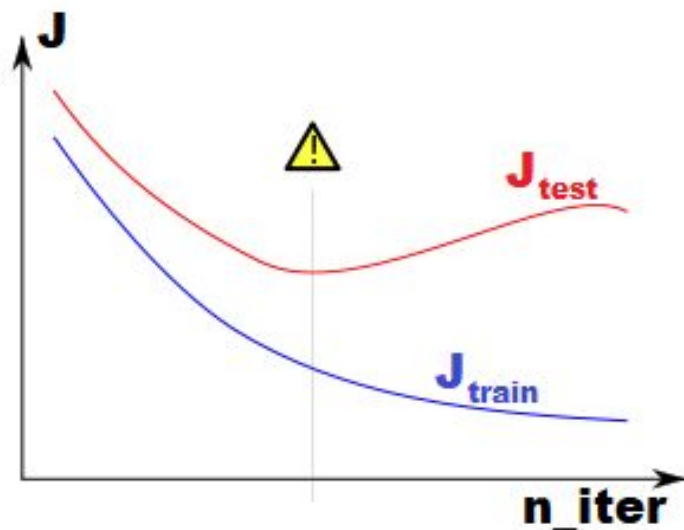


**Can we see any problem with this technique?**

# Early stopping

We stated that we should not train the model using the test set.

**What is actually happening?**

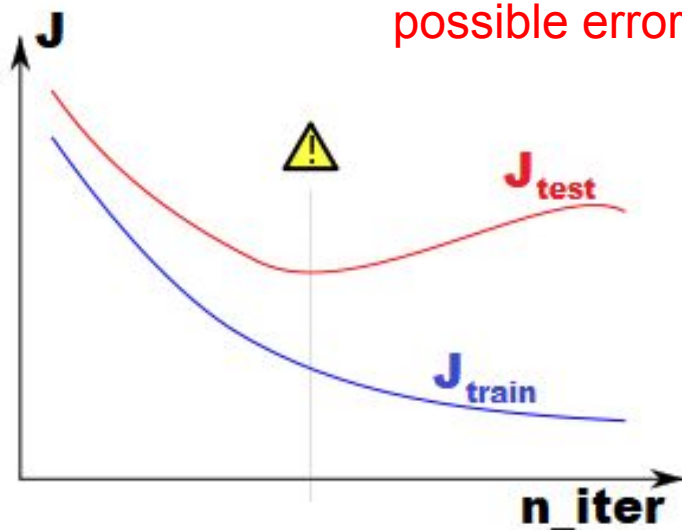


# Early stopping

We stated that we should not train the model using the test set.

What is actually happening?

**We adjust the model to the test set, as we stop training when we have presumably achieved the smallest possible error on the test set.**



**This is cheating...**

# Splitting the sample

**New approach:**

Training set, **validation set**, test set

For example, 50%, 25%, 25% of the sample

# Splitting the sample

**New approach:**

Training set, **validation set**, test set

For example, 50%, 25%, 25% of the sample

**In critical applications**, proportions can change, e.g., 20%, 10%, 70%.  
We expect the test set to enable us to estimate the future performance of the trained model as accurately as possible, based on data that was unseen during training.

# Validation set, hyperparameters

**We will use the validation set to optimize the following parameters:**

- *Learning rate (alpha)*
- *Polynomial degree, or neural network architecture (layers, number of neurons, etc.)*
- *Number of iterations for the gradient method (early stopping)*
- ...

**Such parameters are called hyperparameters.**

**We will use the validation set to find optimal hyperparameters.**

# Hyperparameters

Finding the model with the lowest errors thus consists of two optimization tasks.

Until now:  $\theta^* = \operatorname{argmin}_{\theta} J(\theta)$

From now:  $\psi^*, \theta^* = \operatorname{argmin}_{\psi} \operatorname{argmin}_{\theta} J_{\psi}(\theta)$

$\psi^*$ : Optimal hyperparameters

$\psi$ : hyperparameters

# Model training procedure

**The task:**  $\psi^*, \theta^* = \operatorname{argmin}_{\psi} \operatorname{argmin}_{\theta} J_{\psi}(\theta)$

- 1) Select a new hyperparameter configuration  $\Psi$  and create the model with that.
- 2) Optimize the model parameters ( $\theta$ ) on the **training set** with gradient descent.
- 3) Evaluate the trained model on the **validation set**, then GOTO 1

**Finally:**

- $\Psi^*$  := The hyperparameter configuration with the best performance on the validation set.
- $\theta^*$  := The trained model parameters with hyperparameters  $\Psi^*$ .
- Evaluate model with parameters  $\theta^*$  and hyperparameters  $\Psi^*$  on the **test set**.

# Optimizing hyperparameters

**Gradient descent cannot generally be used for this purpose**

(It is not always possible to specify a differentiable loss function w.r.t. hyperparameters).

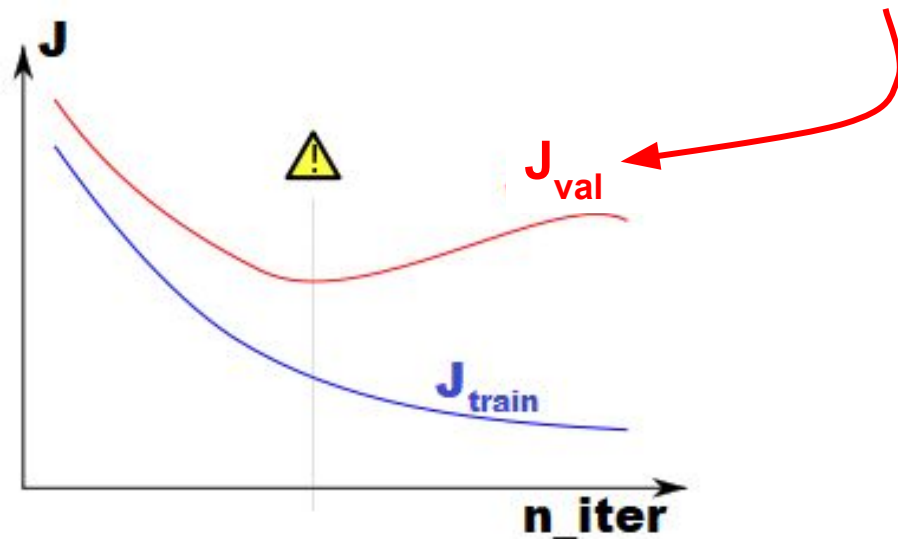
**Common techniques:**

- Manual trial and error
- Grid search
- Random search
- Bayesian optimization
- Evolutionary/genetic algorithms
- ...

# Early stopping done correctly

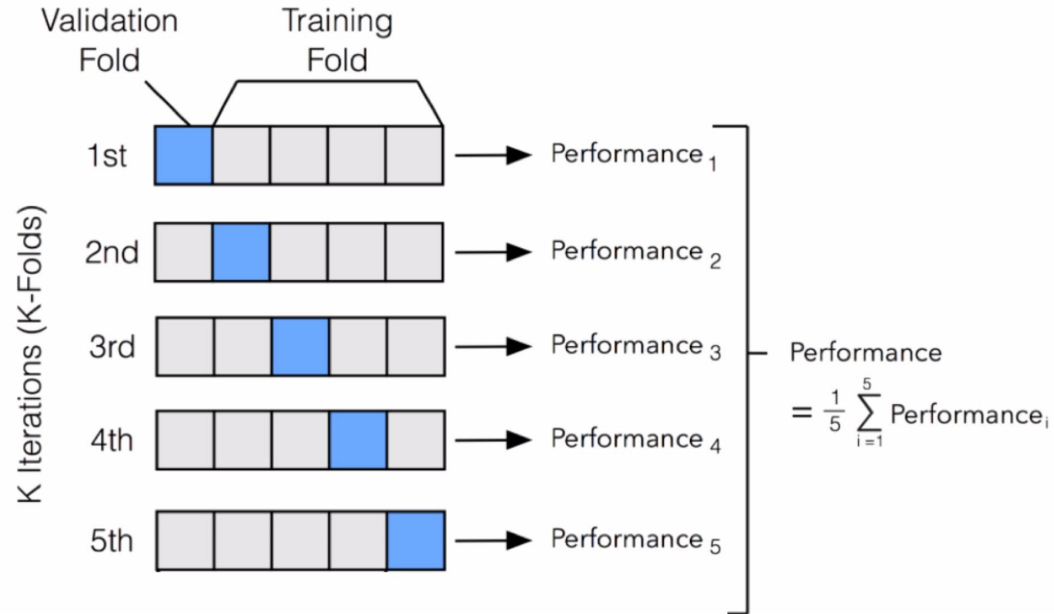
Early stopping done correctly:

Instead of the loss measured on the test set,  
**we observe the validation loss!**



# Selecting the validation set

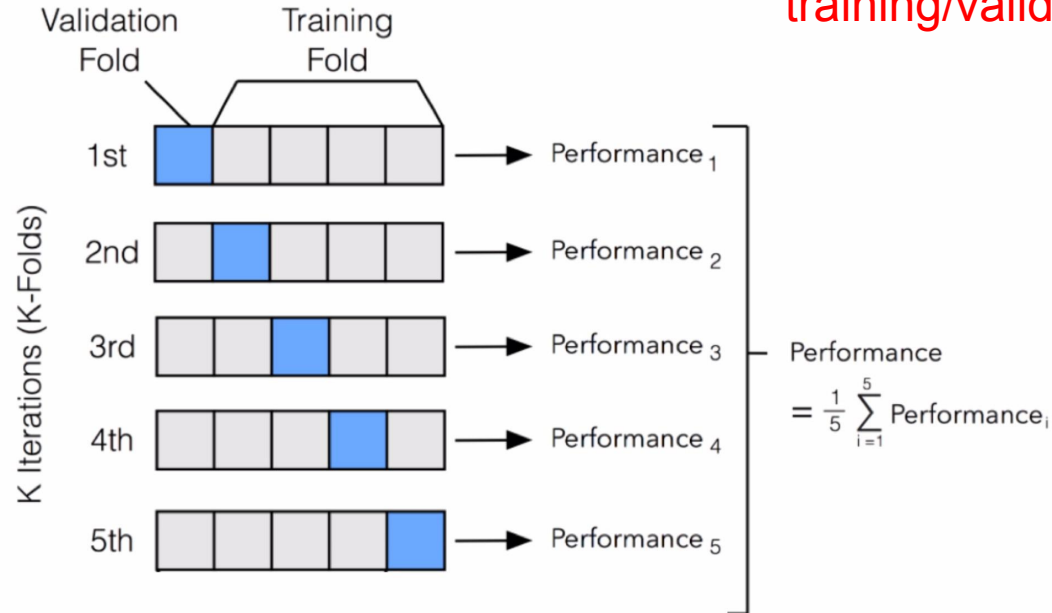
## Cross validation



# Selecting the validation set

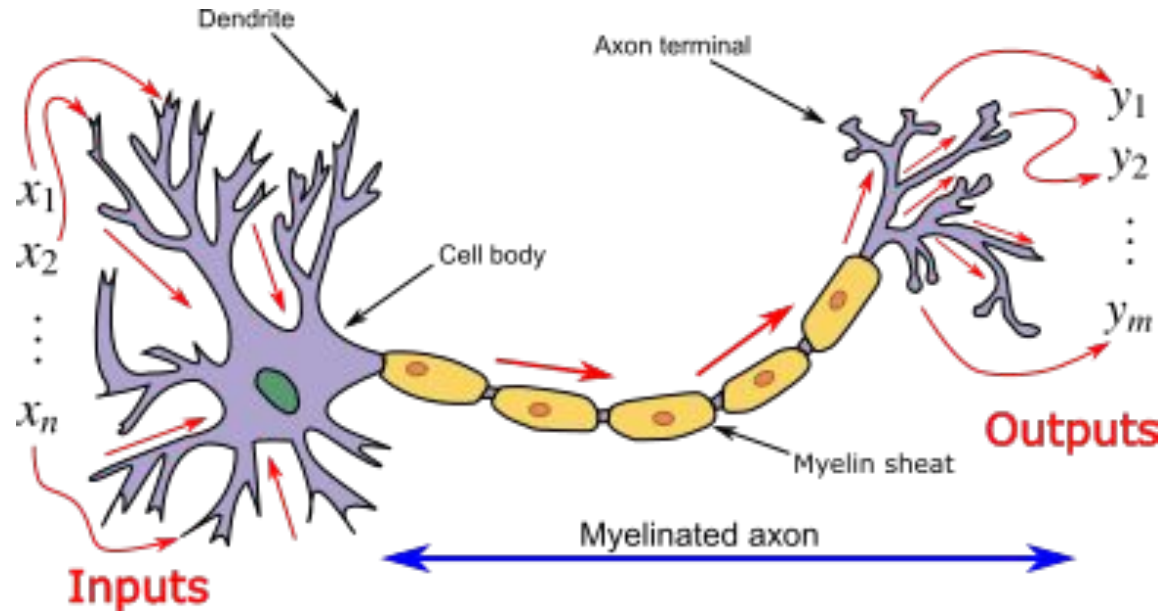
## Cross validation

A popular technique for evaluating hyperparameters. It is most useful **when we have little data available** for training/validation.





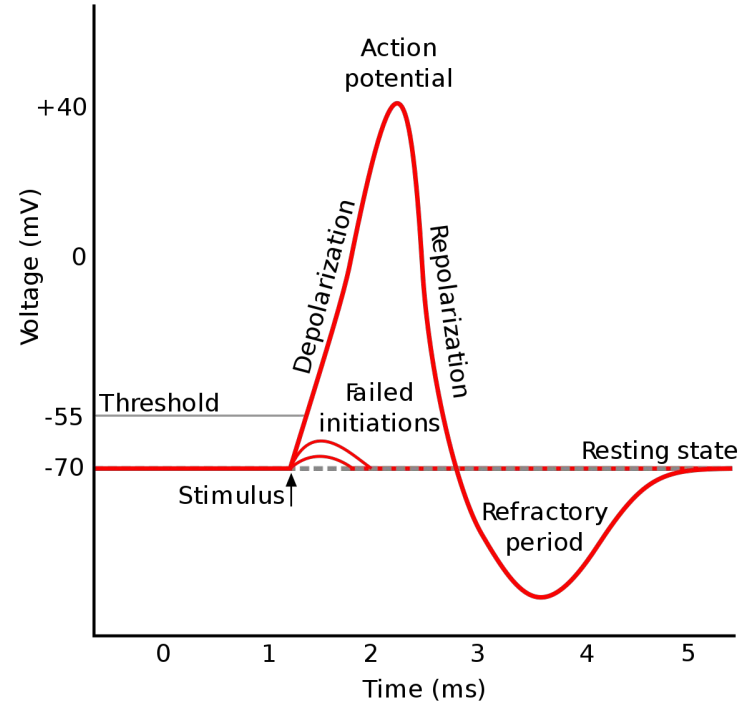
# The biological neuron model





# The functioning of biological neurons simplified

## The functioning of brain cells





# The functioning of biological neurons simplified

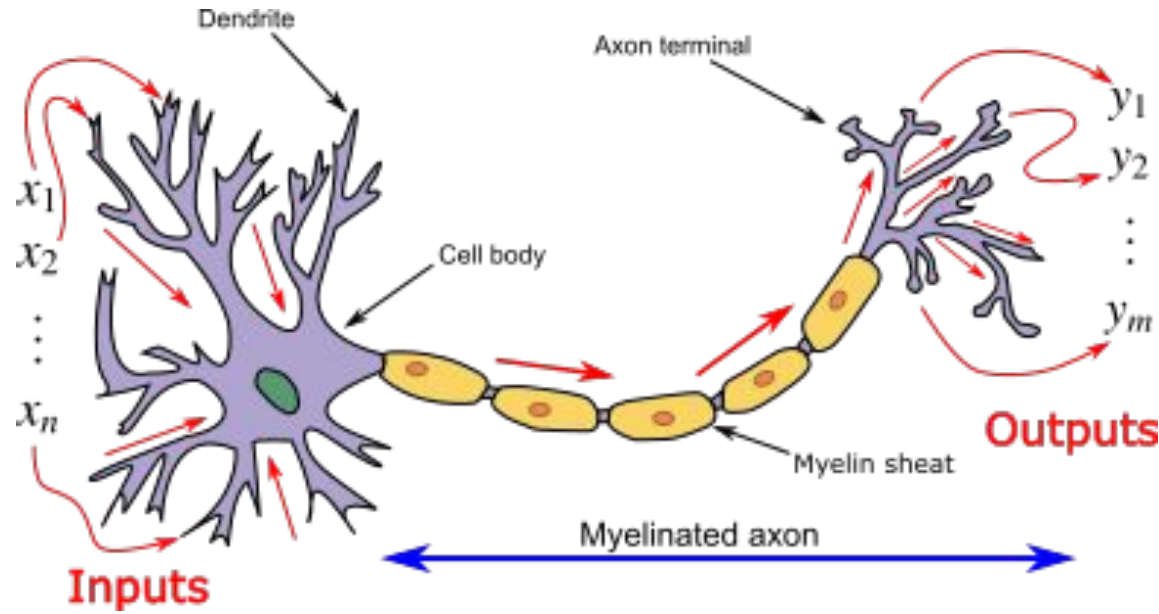
## The functioning of brain cells

- **Membrane potential:** The voltage difference between the inner and outer walls of the cell
- **In the absence of input**, the membrane potential **continuously decreases** to a resting level.
- The membrane potential **increases in response to input**.
- **Inputs** arriving at different branches (dendrites) are amplified or attenuated (can also be negative) with **different weights**.
- When the membrane potential **reaches a threshold** (specific to each neuron), the neuron **“fires”** and charge passes through the output.

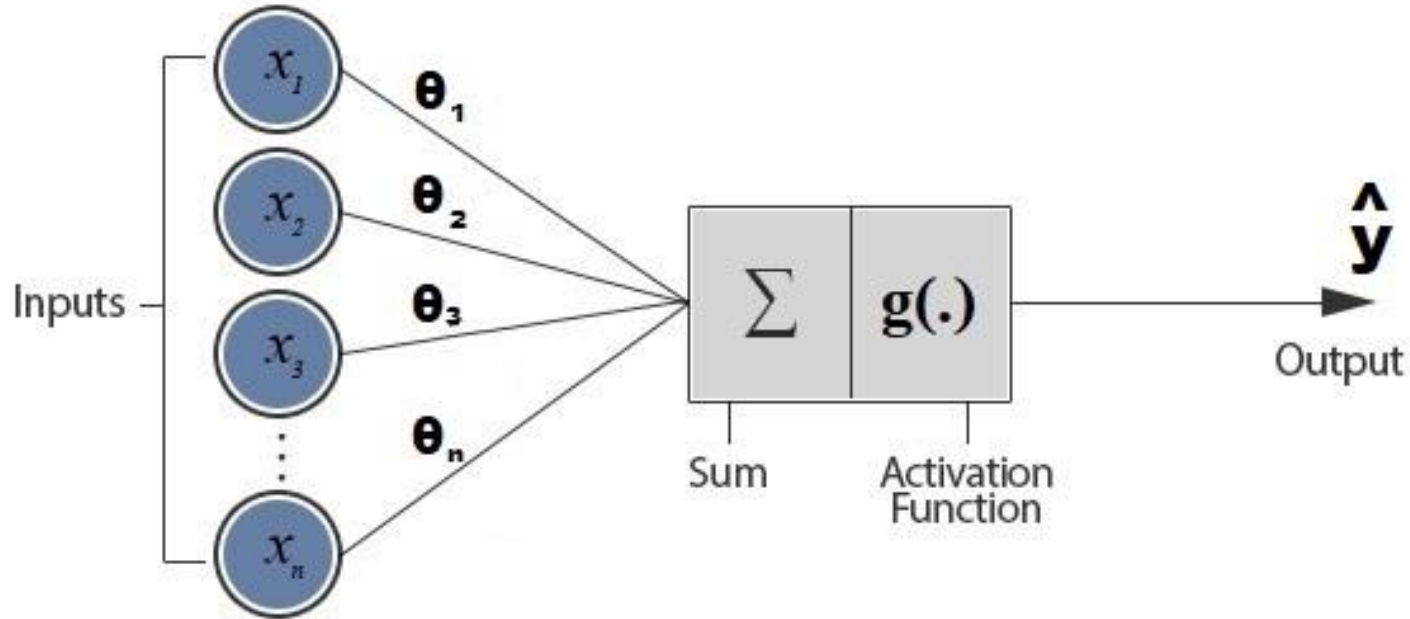


# The biological neuron model

**What does its functioning resemble?**

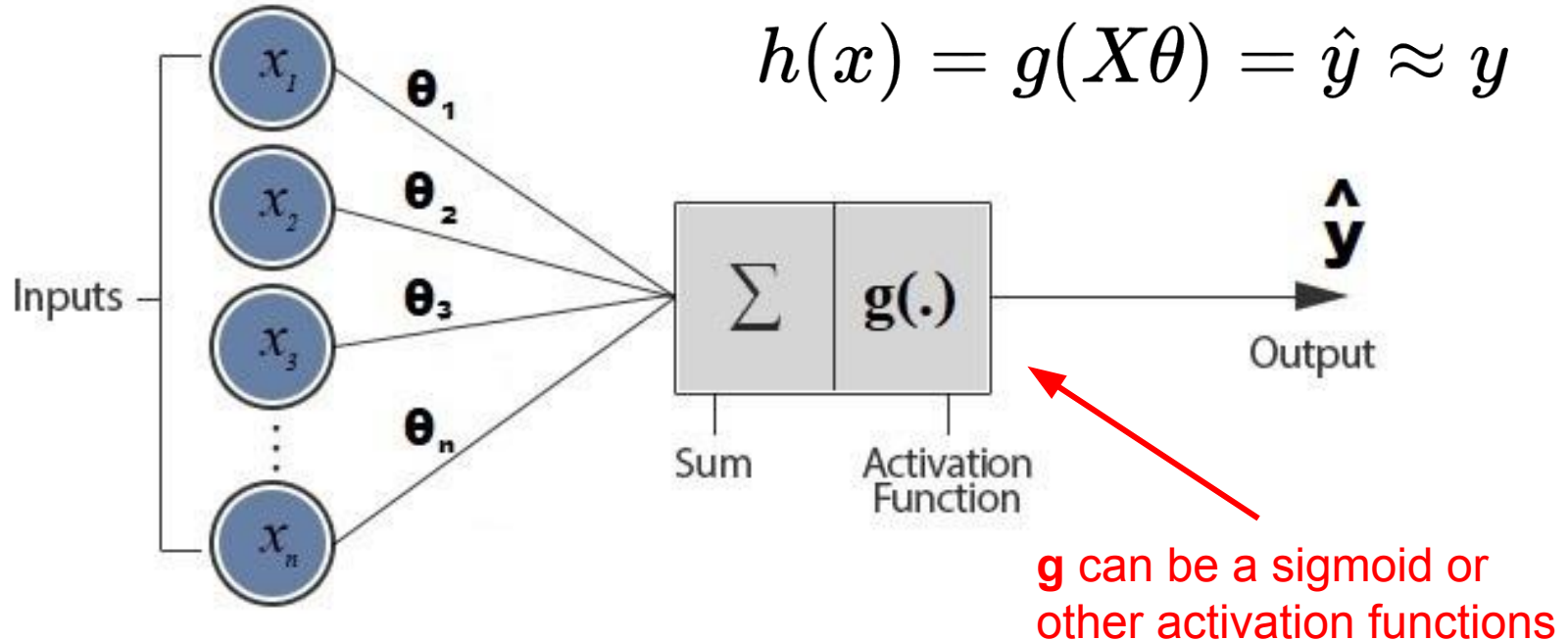


# The artificial neuron model (Rosenblatt, 1958)



The simplified, discretized model of a biological neuron. **What does it resemble?**

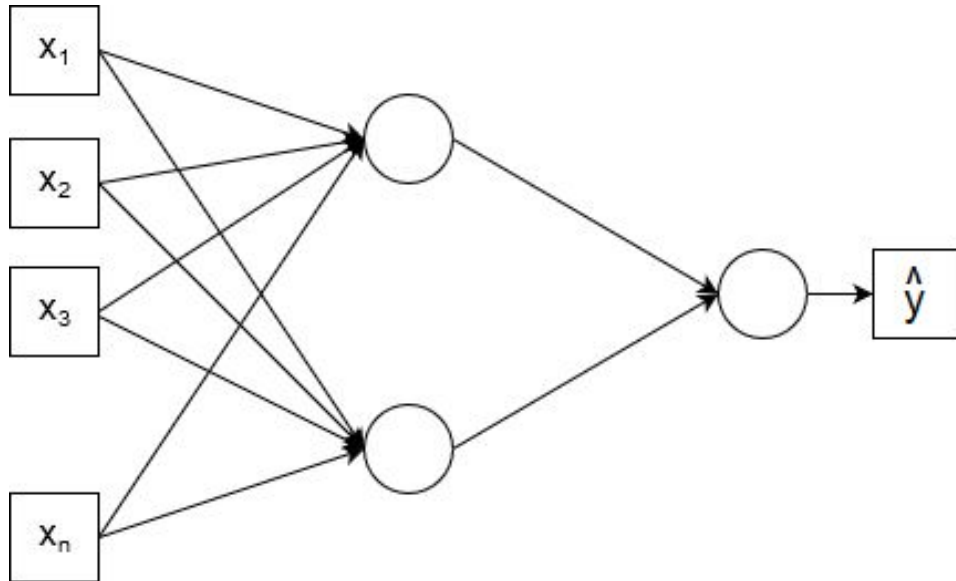
# The artificial neuron model (Rosenblatt, 1958)



**An artificial neuron is a (multivariate) logistic regression if  $g$  is sigmoid!**

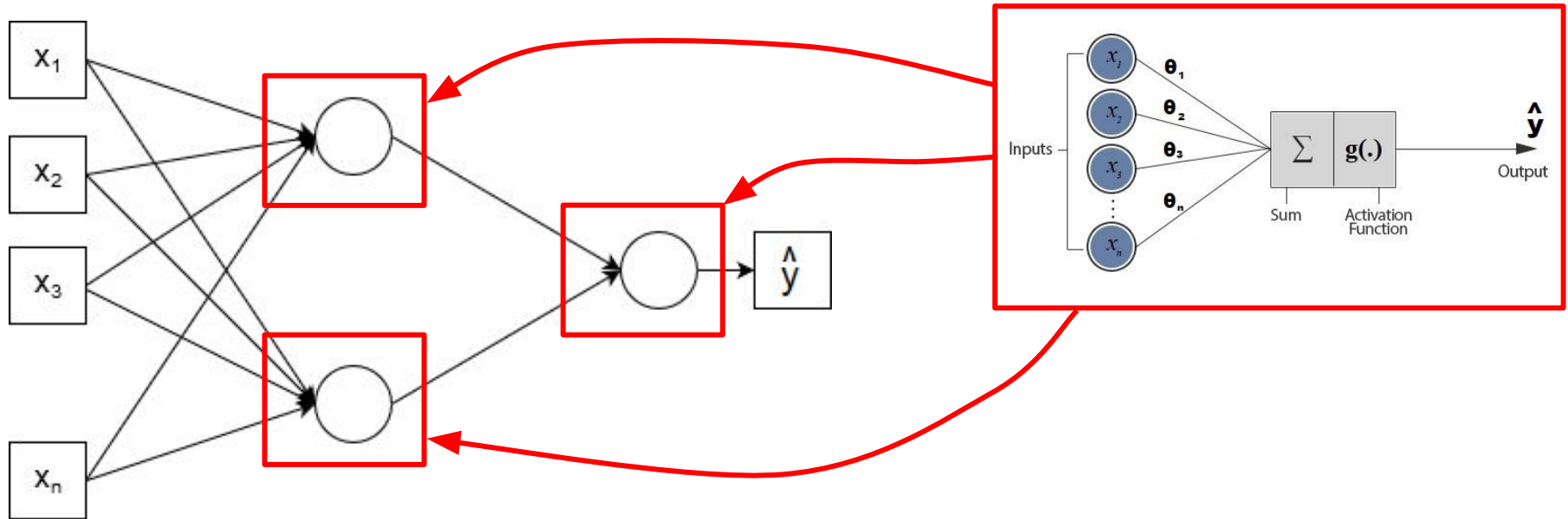
# Building blocks of neural networks

**Artificial neurons are the building blocks of one of the basic types of artificial neural networks (the Multilayer Perceptron, MLP).**



# Building blocks of neural networks

Artificial neurons are the building blocks of one of the basic types of artificial neural networks (the Multilayer Perceptron, MLP).

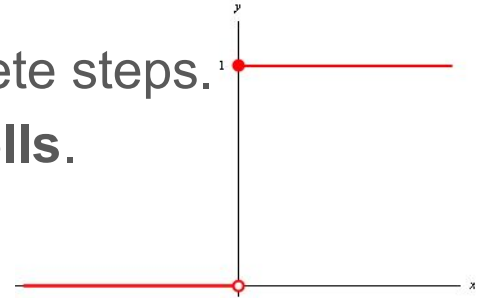




# The functioning of biological neurons simplified

## Differences from the artificial neuron model:

- Continuous signal in the nerve cell instead of discrete steps.  
→ **Sum in artificial neurons, integral in nerve cells.**

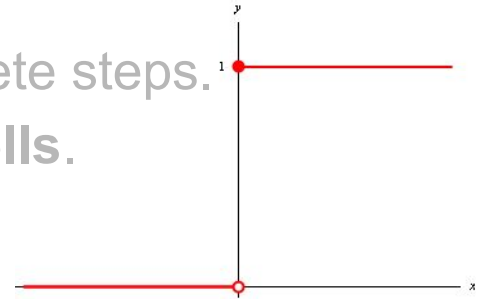




# The functioning of biological neurons simplified

## Differences from the artificial neuron model:

- Continuous signal in the nerve cell instead of discrete steps.  
→ **Sum in artificial neurons, integral in nerve cells.**
- In a nerve cell, nonlinearity does not have to be continuous (Heaviside step function).  
→ **In an artificial neuron, continuous, a differentiable nonlinearity is required.**



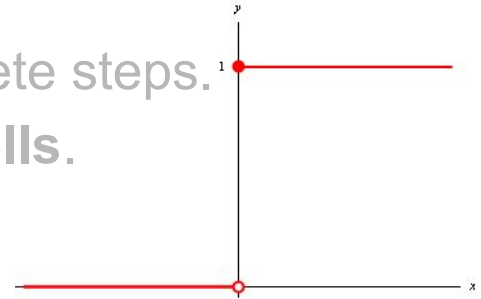
(and the derivative should not be zero everywhere...)



# The functioning of biological neurons simplified

## Differences from the artificial neuron model:

- Continuous signal in the nerve cell instead of discrete steps.  
→ **Sum in artificial neurons, integral in nerve cells.**
- In a nerve cell, nonlinearity does not have to be continuous (Heaviside step function).  
→ **In an artificial neuron, continuous, a differentiable nonlinearity is required.**
- The weight of a nerve cell is either always negative or always positive; the sign does not change (excitator vs. inhibitor)  
→ **In an artificial neuron, the sign of the weight can change.**



# Summary

## **May be included in tests / exam:**

- Identifying and handling of under- and overfitting, hyperparameters, validation set
- The artificial neuron (Perceptron) model

## **Will not be included in tests / exam:**

- Polynomial regression
- Polynomial interpolation
- The biological neuron model and differences from the artificial model