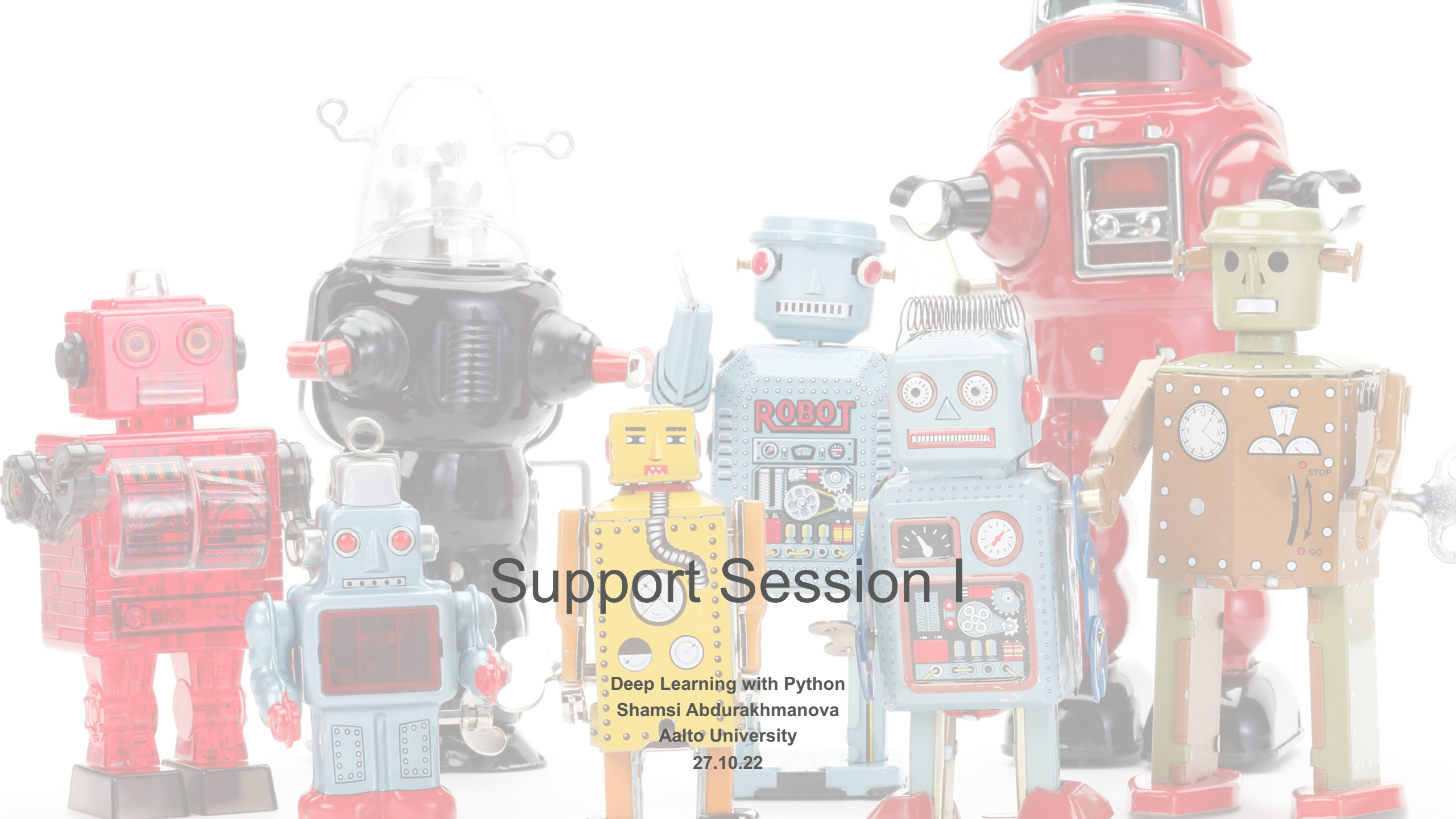



# Support Session I

**Deep Learning with Python**
**Shamsi Abdurakhmanova**
**Aalto University**
**27.10.22**

# ✓ Content

- Python basics. NumPy arrays, Vectors and Matrices
- 3 Components of ML:
    - recap
    - regression example
    - classification example

# Part I:

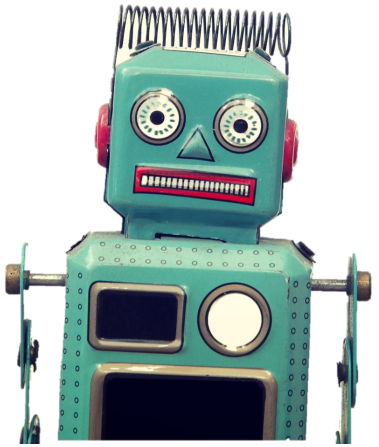
# Python

# Python

- Programming language
- Implementations ( CPython, PyPy, Jython ) /Distribution (CPython: Anaconda)

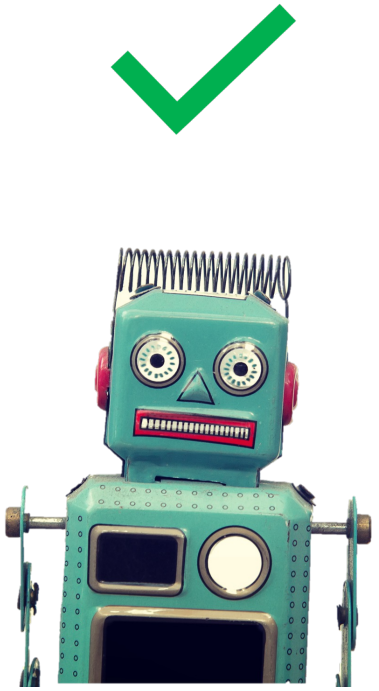
# Python Programming language

```
# The following line will print out some text
print("Hello World!\n")
```



# Python Programming language

```
# The following line will print out some text
print("Hello World!\n")
```

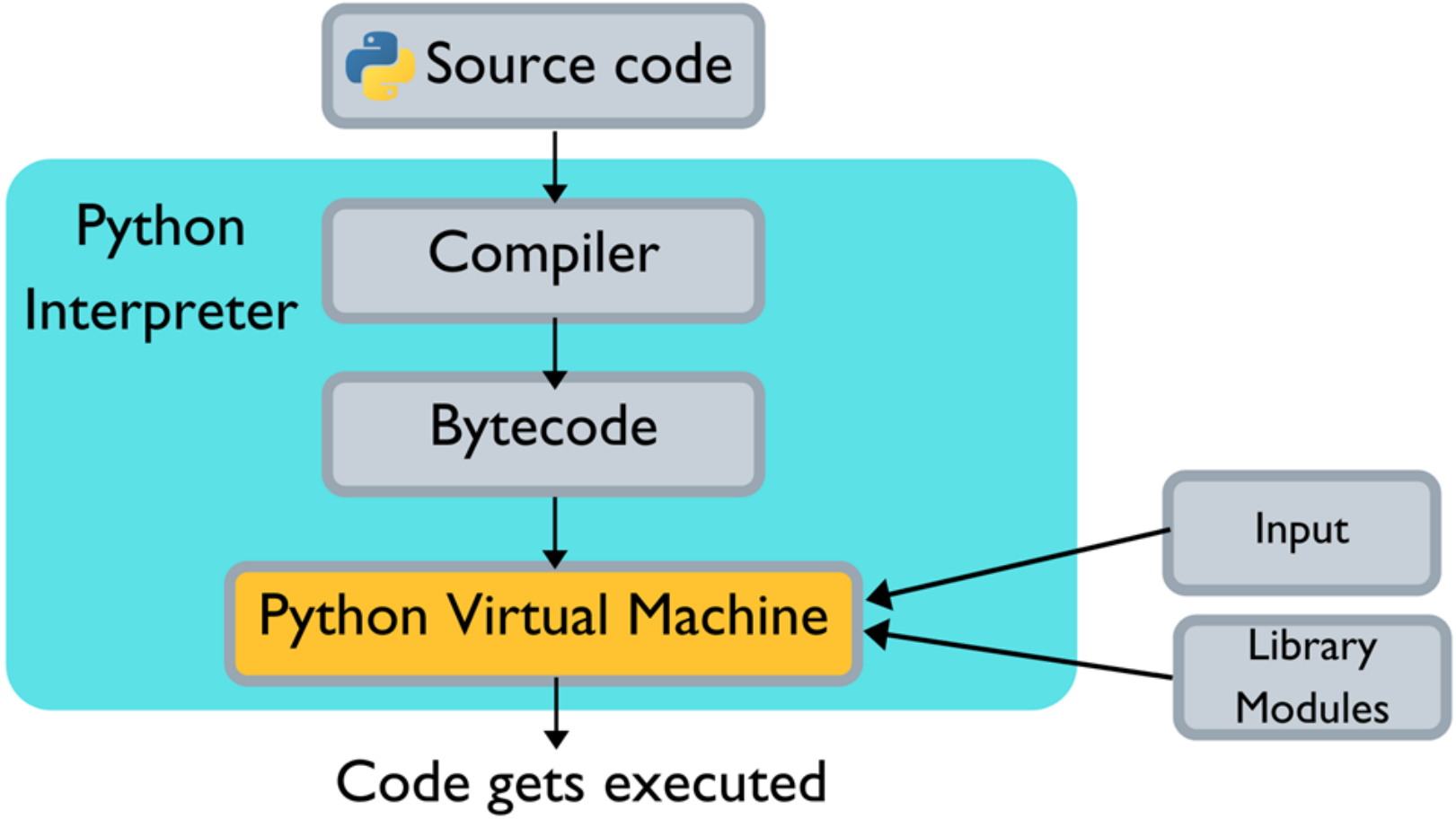
# Python Implementations

```
# The following line will print out some text
print("Hello World!\n")
```

CPython, PyPy, Jython, …

```
10111010 00001110 00000000 10110100
00100001 10111000 00000001 01001100
01010100 01101000 01101001 01110011
01110010 01101111 01100111 01110010
00100000 01100011 01100001 01101110
01110100 00100000 01100010 01100101
01110101 01101110 00100000 01101001
```

# Python Implementation




Source

# Python Programming language

“High-level general-purpose programming language developed by Guido van Rossum in the late 1980s and released in 1991. It is a *dynamically-typed language*, which means we don't have to declare the type of the variable, and the type of the value is known when the code gets executed and the *memory management is done automatically*.”

Source

# Python – where to write/execute code?

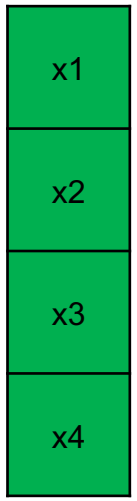
- text editor + terminal

- IDEs (VSCode, Spyder, PyCharm)

- Jupyter/Colab notebook (locally or remotely)

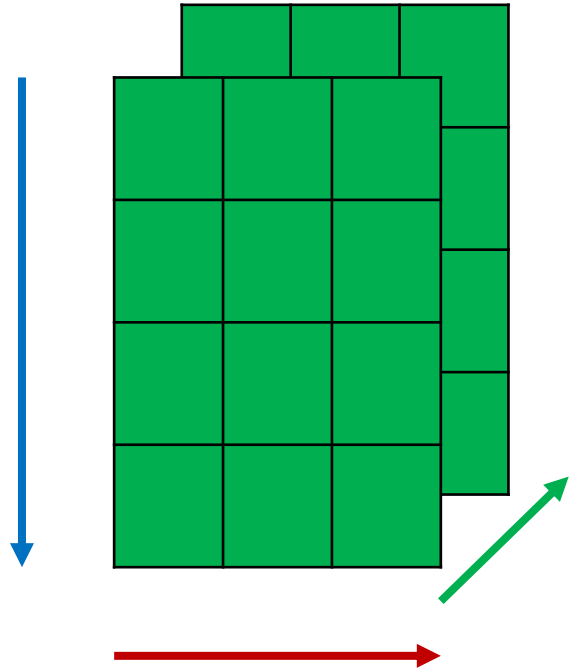
“Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine.”

# NumPy Arrays ($N$-dimensional arrays of numbers)



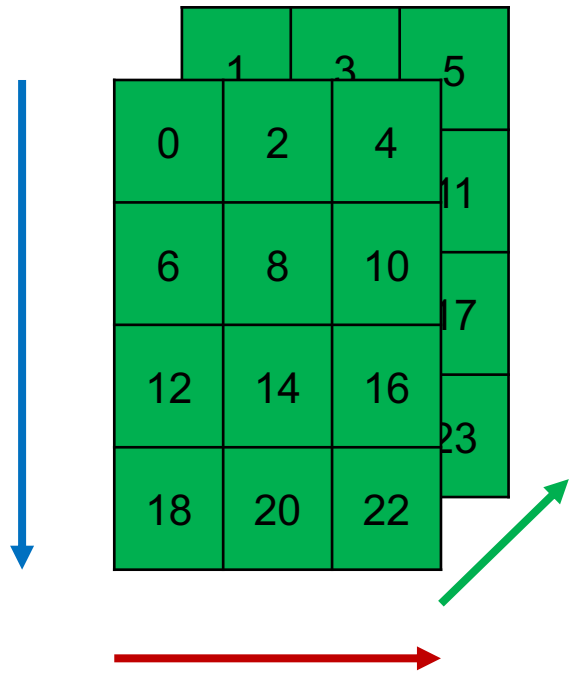

1-dim array
shape (4,)


2-dim array
shape (4,3)



3-dim array
shape (4,3,2)

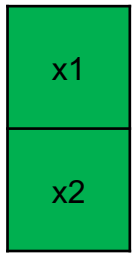
# NumPy Arrays ($N$-dimensional arrays of numbers)



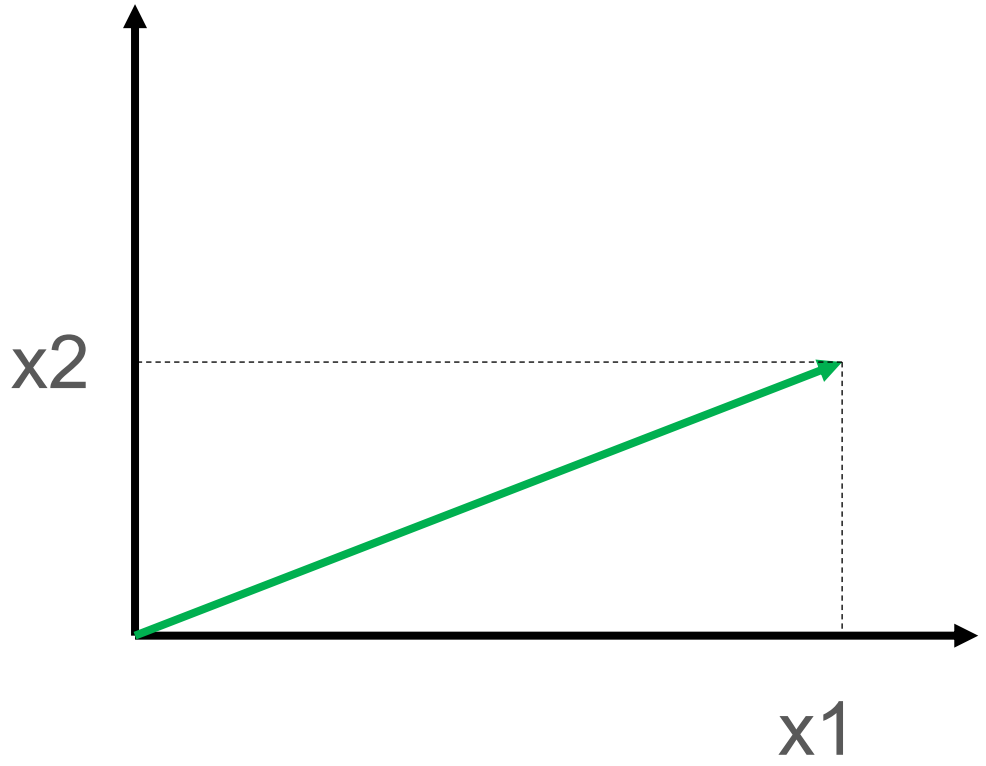

3-dim array
shape (4,3,2)

```
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

       [[ 6,  7],
        [ 8,  9],
        [10, 11]],

       [[12, 13],
        [14, 15],
        [16, 17]],

       [[18, 19],
        [20, 21],
        [22, 23]]])
```

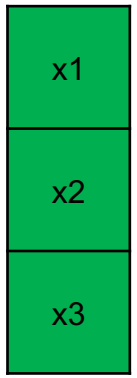
# NumPy Arrays as vectors



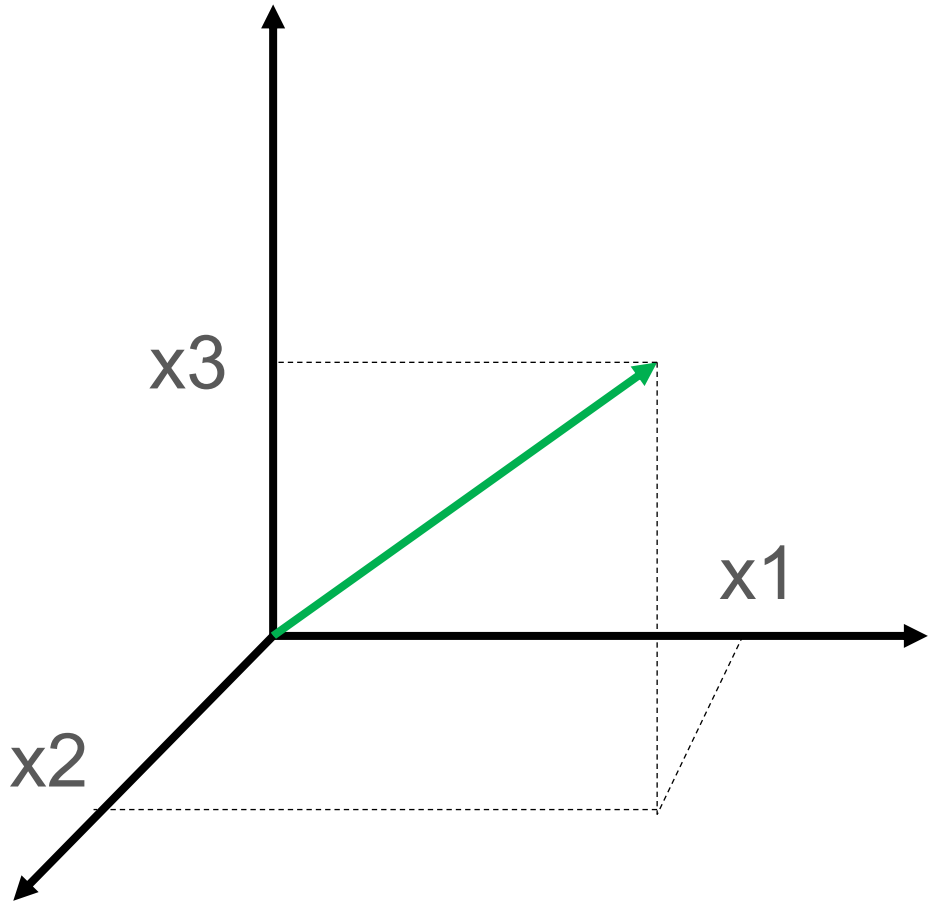

vector **x =** (x1, x2)



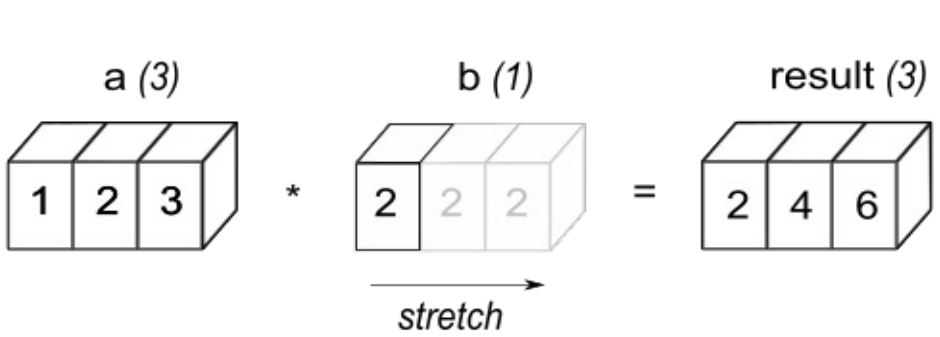
# NumPy Arrays as vectors

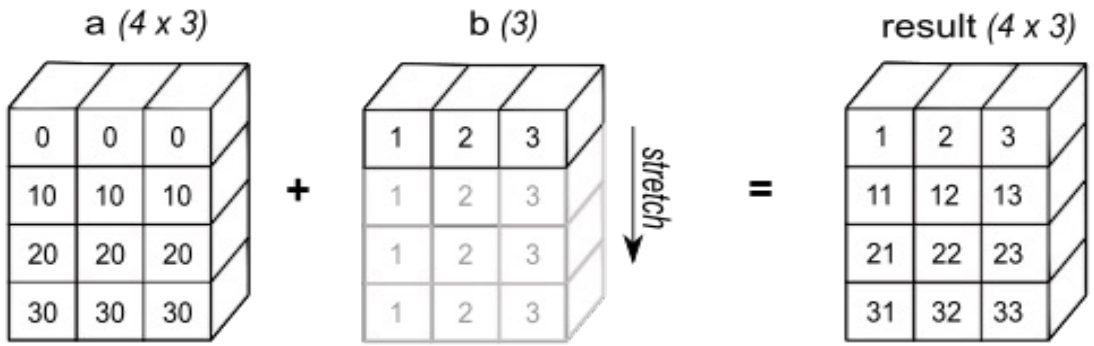



vector **x =** (x1, x2, x3)

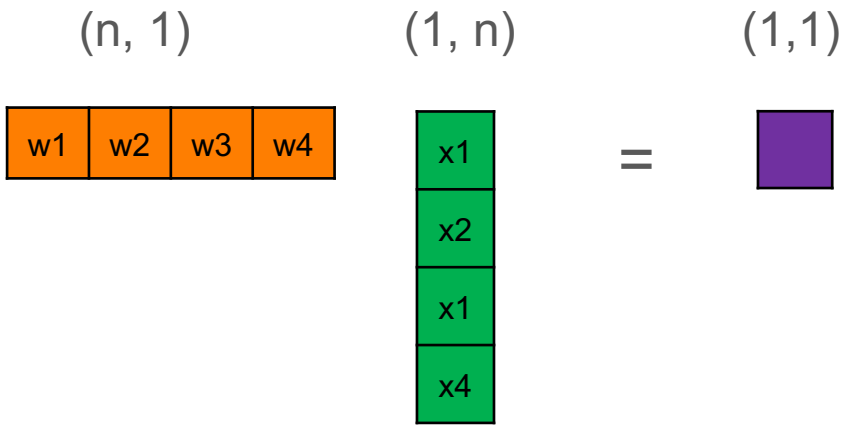


# NumPy Broadcasting



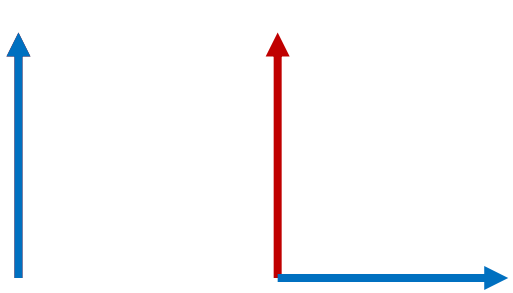



# Vector operations – dot product




$$\mathbf{w^T x} = w_1 x_1 + \ \ldots + w_n x_n = \sum_{i=1}^{n} w_i x_i$$

# Vector operations – dot product



$cos\theta = 1$

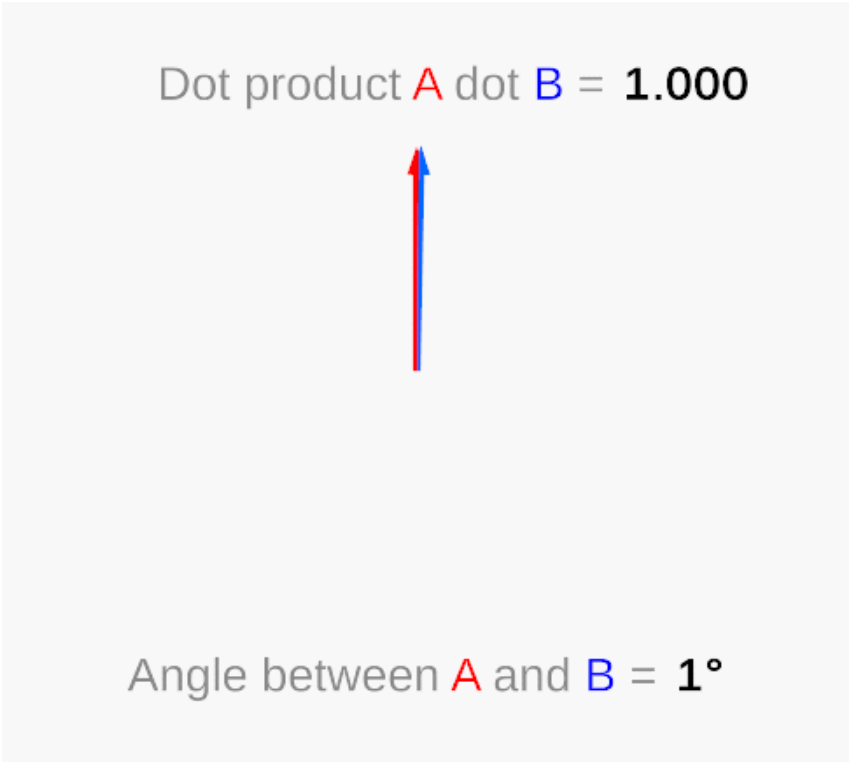


$cos\theta = 0$

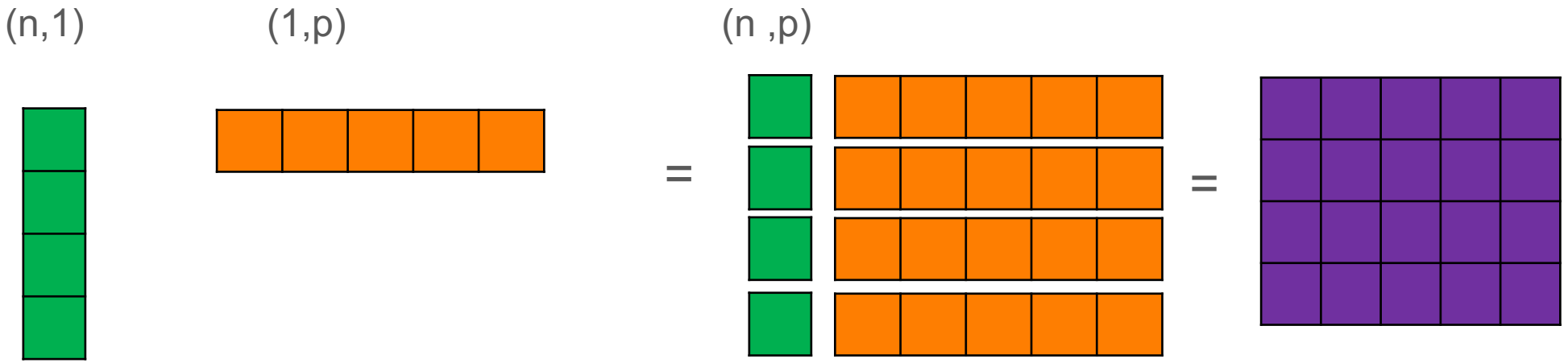
$cos\theta = -1$

for $|\mathbf{A}| = |\boldsymbol{B}| = 1 \rightarrow \boldsymbol{A}^{\mathbf{T}}\mathbf{B} = cos\theta$

$$\boldsymbol{A}^{\mathbf{T}}\mathbf{B} = |\mathbf{A}||\boldsymbol{B}|cos\theta$$



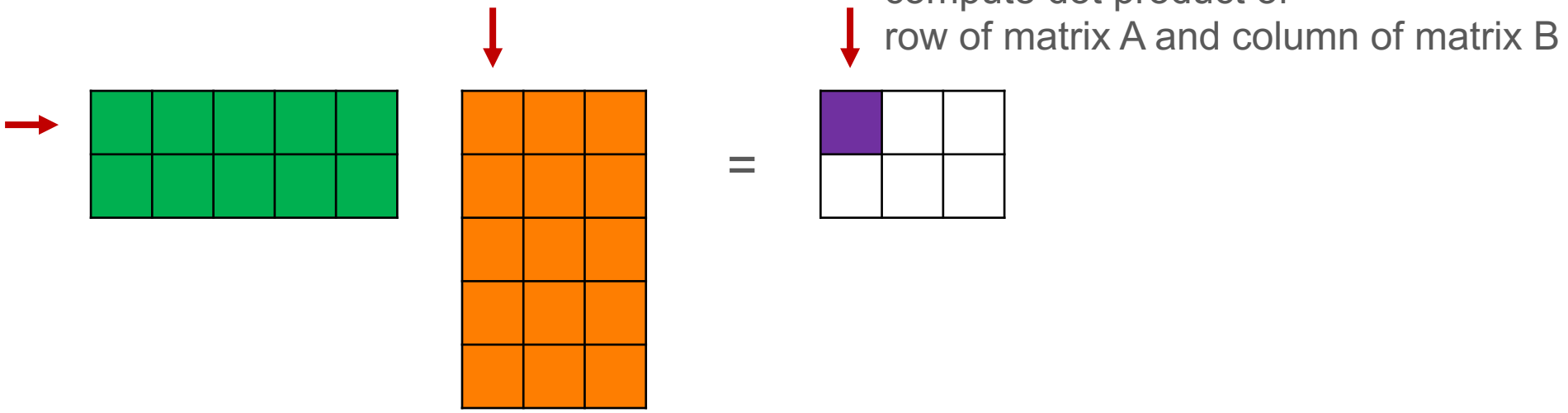
# Vector operations – outer product




$$\mathbf{w} \otimes \mathbf{x} = \mathbf{w}x^{T} = \begin{matrix} w_1x_1 & w_1x_2 & \dots & \dots & w_1x_p \\ \dots & \dots & \dots & \dots & \dots \\ w_nx_1 & w_nx_2 & \dots & \dots & w_nx_p \end{matrix}$$

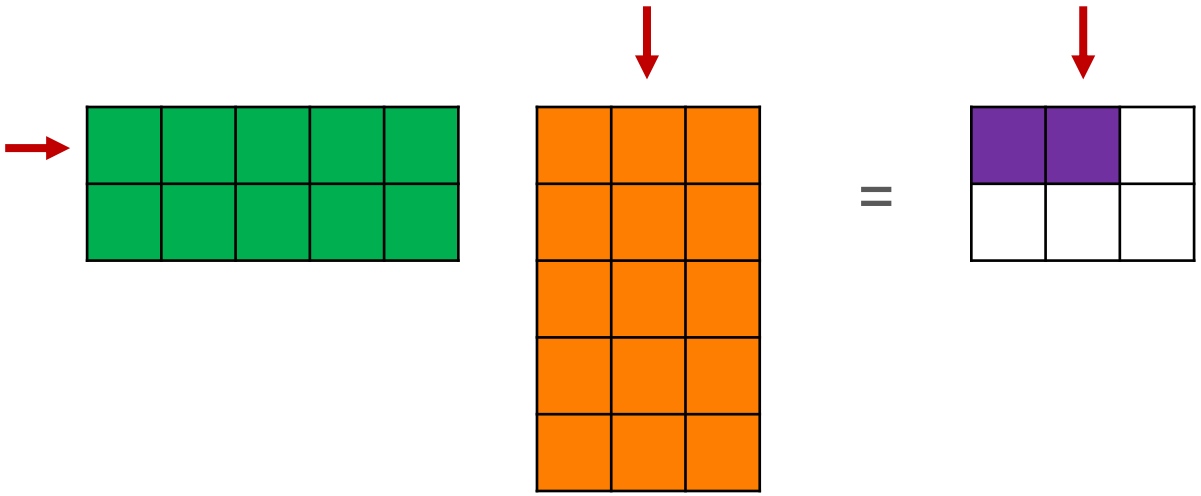
# Matrix multiplication

matrix A (2, 5)  matrix B (5,3)  matrix C (2 ,3)



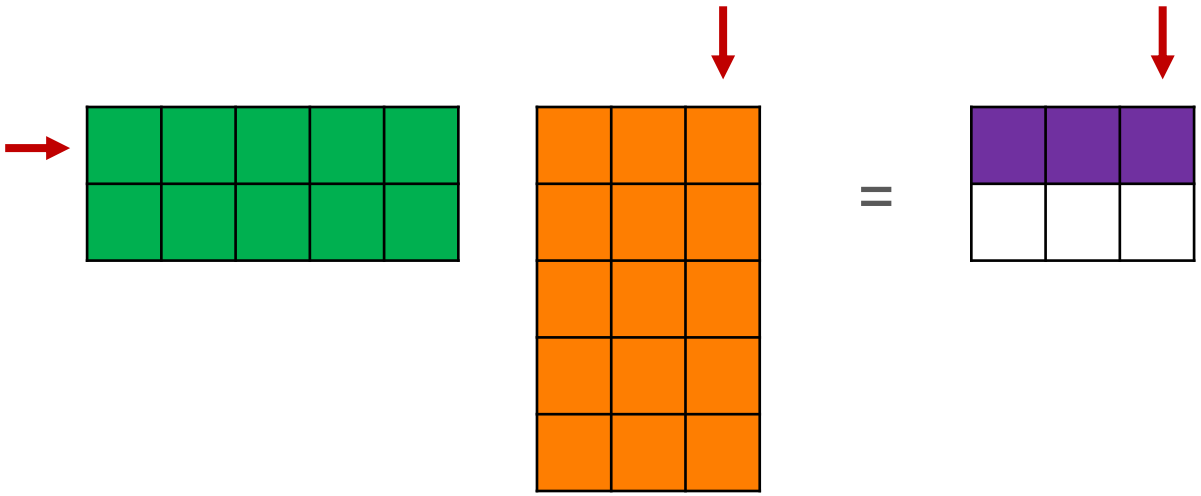
# Matrix multiplication

matrix A (2, 5)  matrix B (5,3)  matrix C (2 ,3)



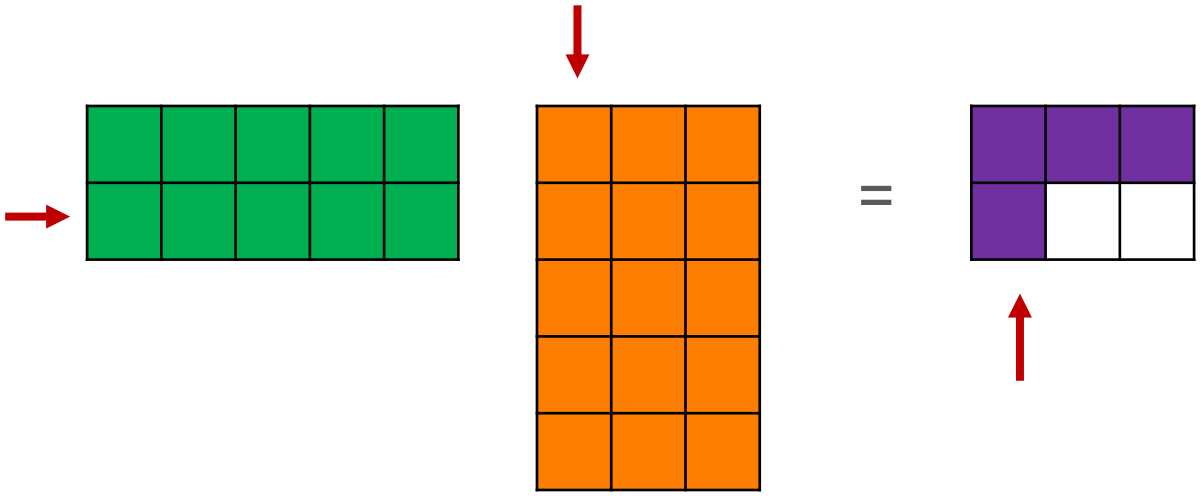
# Matrix multiplication

matrix A (2, 5)  matrix B (5,3)  matrix C (2 ,3)



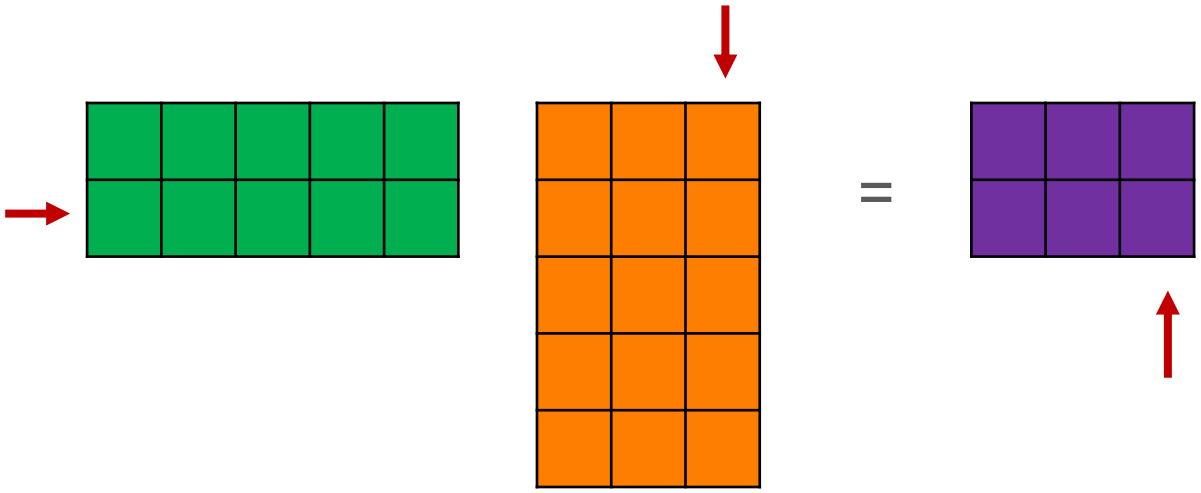
# Matrix multiplication

matrix A (2, 5)  matrix B (5,3)  matrix C (2 ,3)



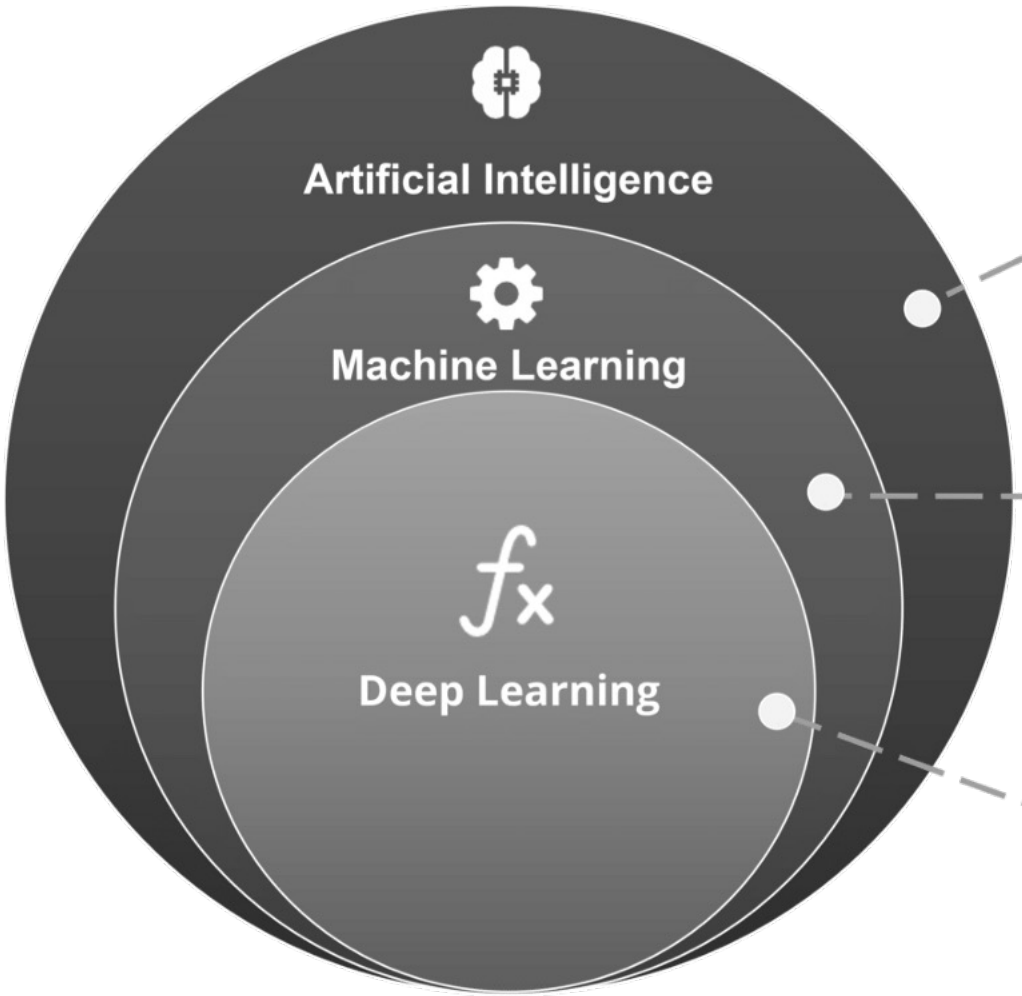
# Matrix multiplication

matrix A (2, 5)  matrix B (5,3)  matrix C (2 ,3)



# Part II:

# Three Components of ML




## ARTIFICIAL INTELLIGENCE

A technique which enables machines to mimic human behaviour

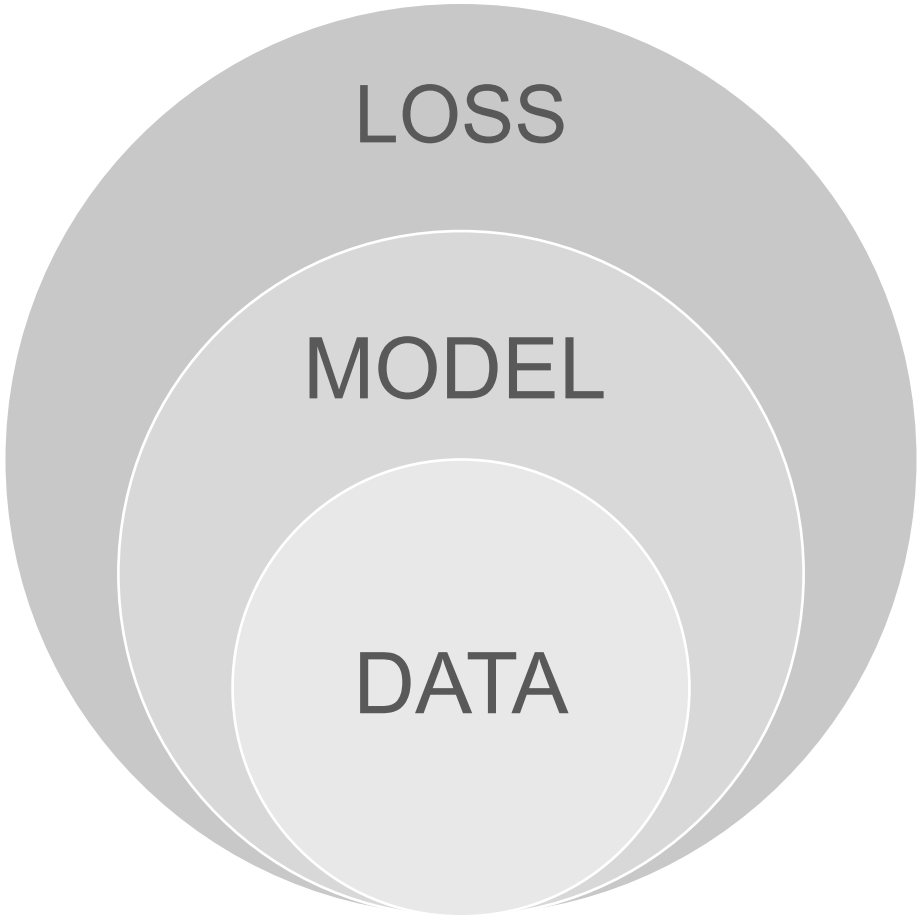
## MACHINE LEARNING

Subset of AI technique which use statistical methods to enable machines to improve with experience

## DEEP LEARNING

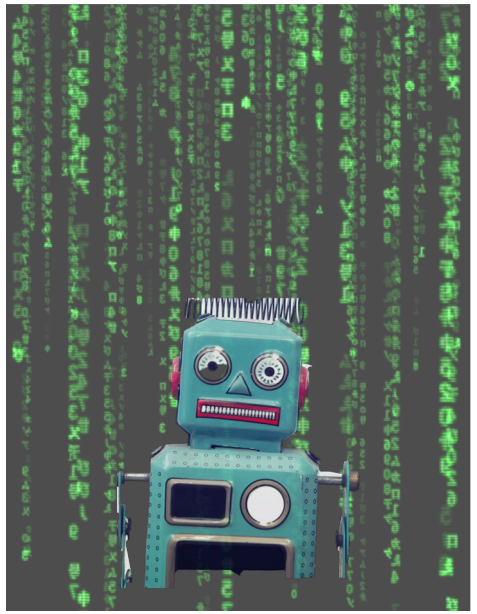
Subset of ML which make the computation of multi-layer neural network feasible

# Three Components of ML




ML Method

# Data

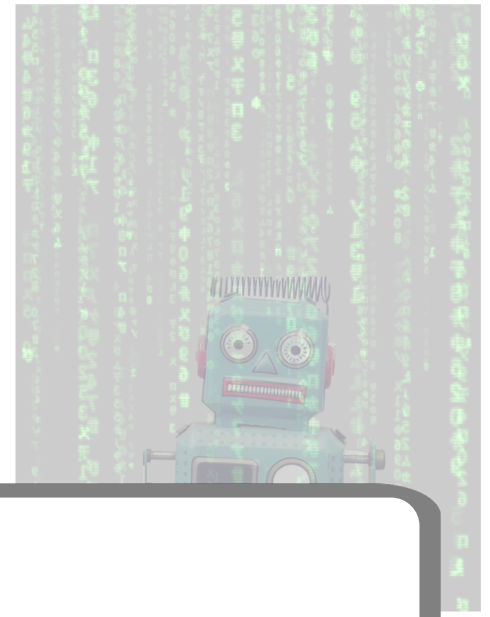


- collections of individual datapoints

- datapoint -  any object that conveys information

- datapoint is characterized by **features** and **label(s)**

# Data



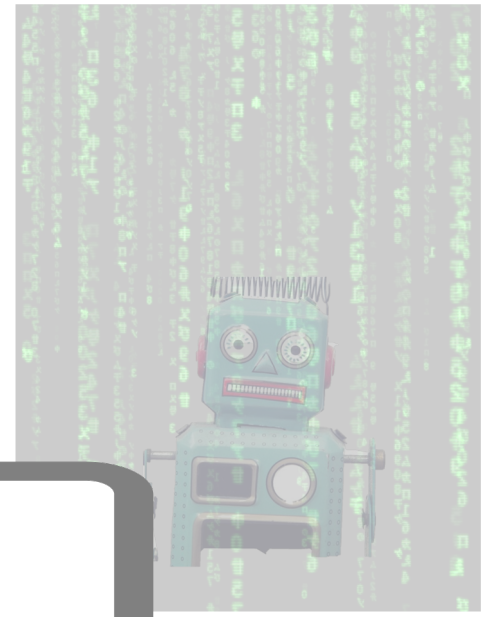
- collections of individual datapoints

- datapoints - atomic units

***Useful*** properties of a datapont that can be “easily” collected

- datapoint is characterized by **features** and **label(s)**

# Data



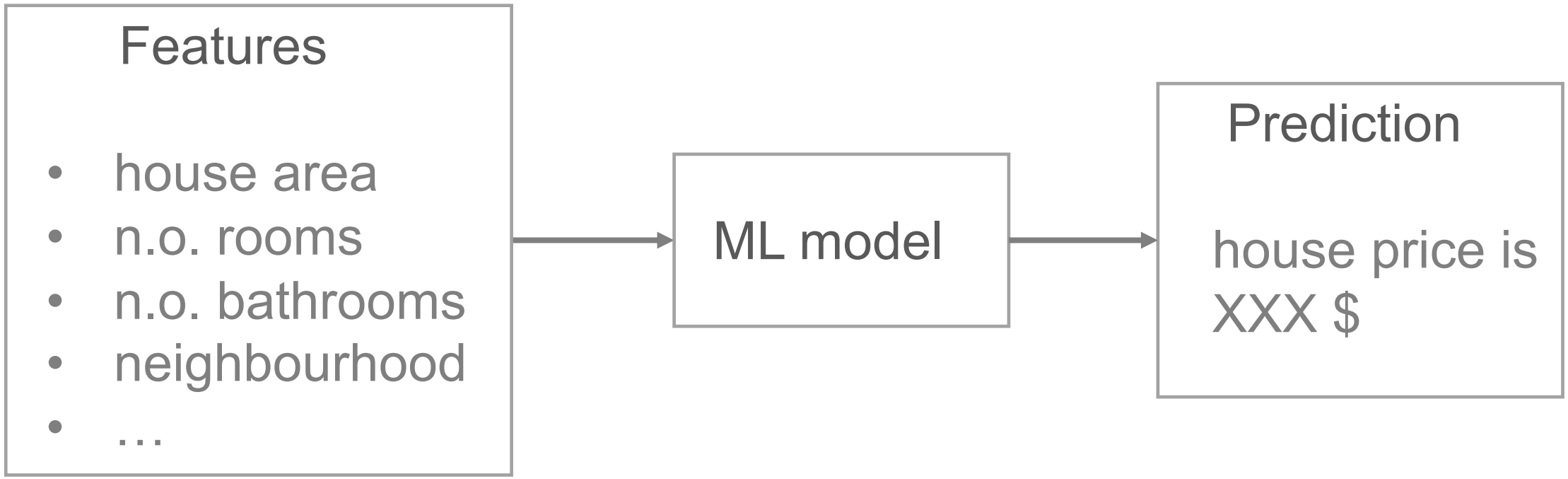
- collections of indivi
- datapoints - atom

Property of a datapont that we want predict.
“Quantity of interest”

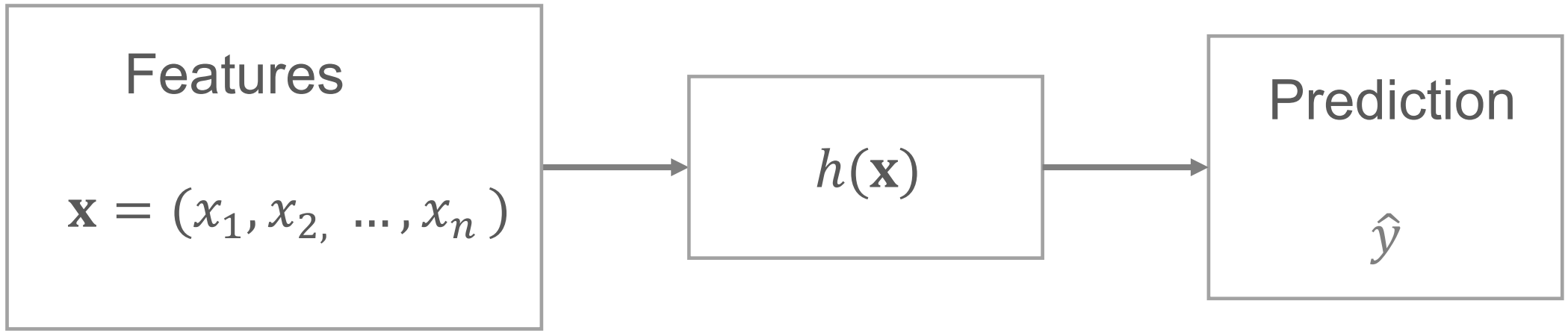
- datapoint is characterized by **features** and **label(s)**

| DATA POINT: | FEATURES/ ATTRIBUTES/ CHARACTERISTICS: | LABEL: |
| --- | --- | --- |
| • house | → number of rooms | → price |
| • text | → word “cat” count in the text | → article topic |
| • genome sequence | → frequency of AATCAGTT motif | → cell type |
| • image | → pixel values | → identify objects on image |
| • speech | → frequency spectrum | → emotional state |

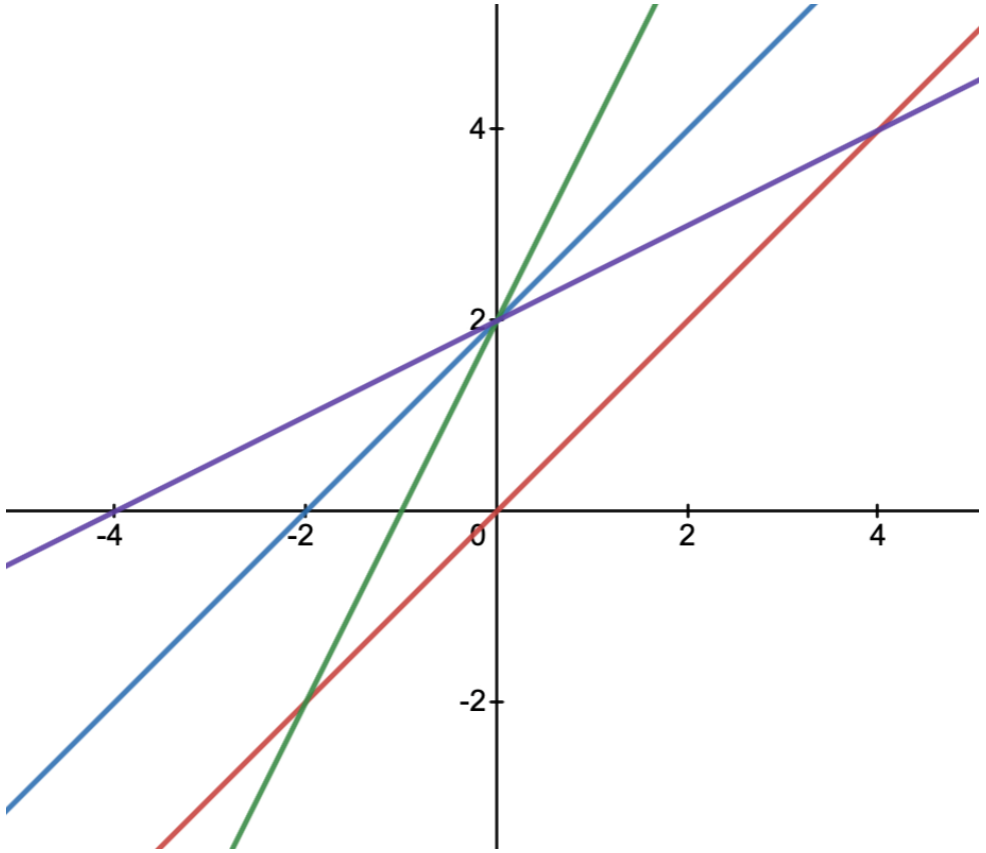
# Model


Features
house area
n.o. rooms
n.o. bathrooms
neighbourhood
…
ML model
Prediction
house price is
XXX $

# Model



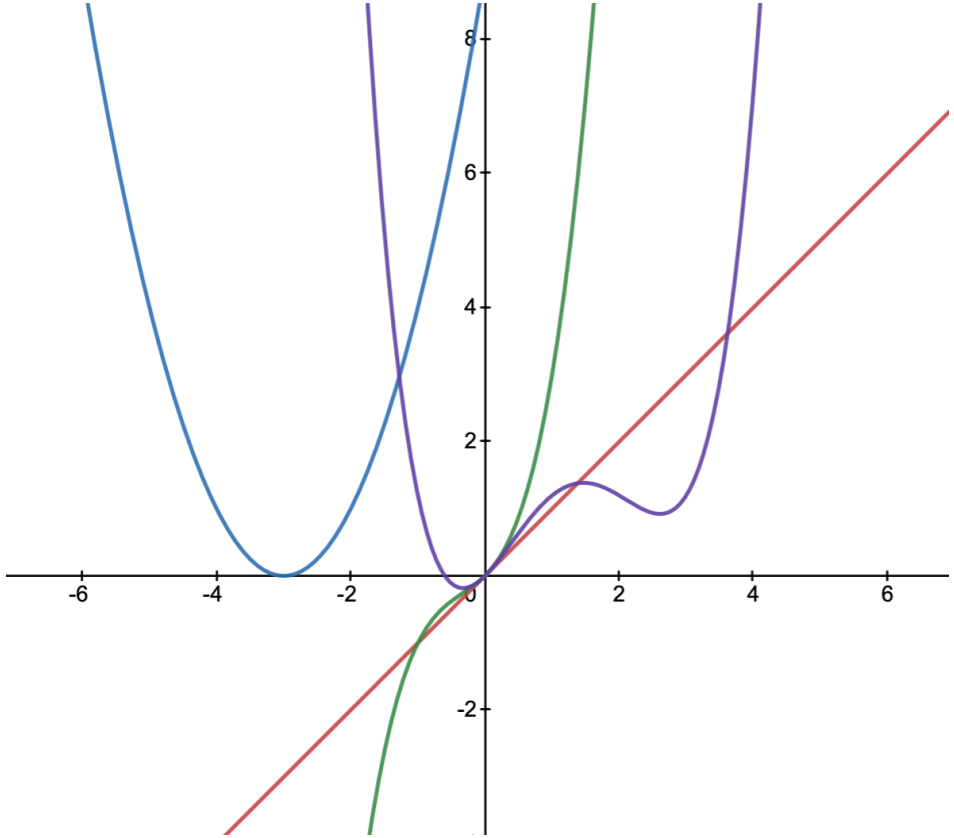
# Model ≈ specific type of function




Linear model

- $h(\mathbf{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
- Domain – real numbers
- Unbounded
- n.o. parameters (args) = n.o. features + 1

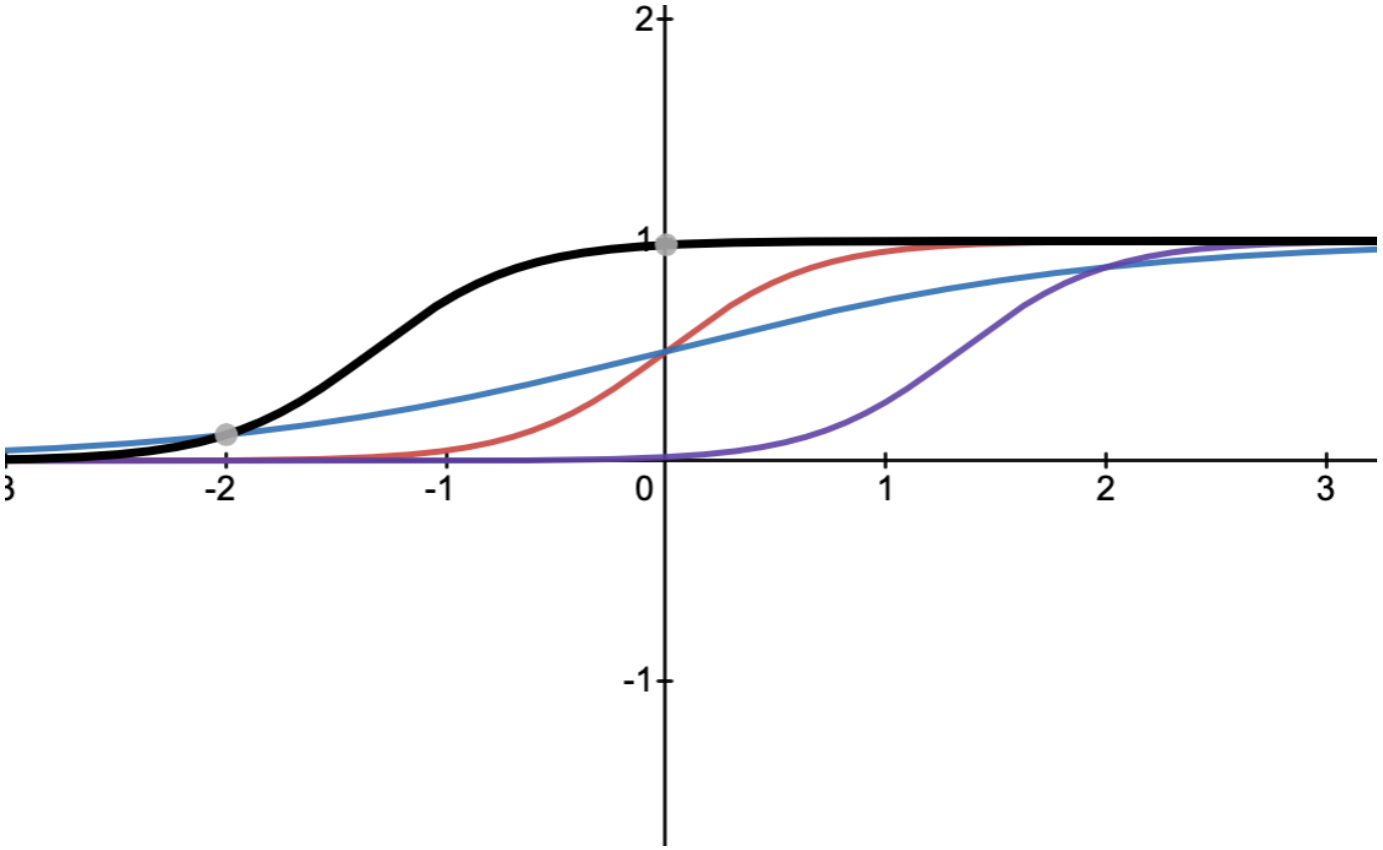
# Model ≈ specific type of function




Polynomial model

- $h(\mathbf{x}) = w_1 x_1 + w_2 x_1^2 + w_3 x_1^3 + \cdots$
- Domain – real numbers
- Unbounded
- n.o. parameters (args) – depends on degree and n.o. features

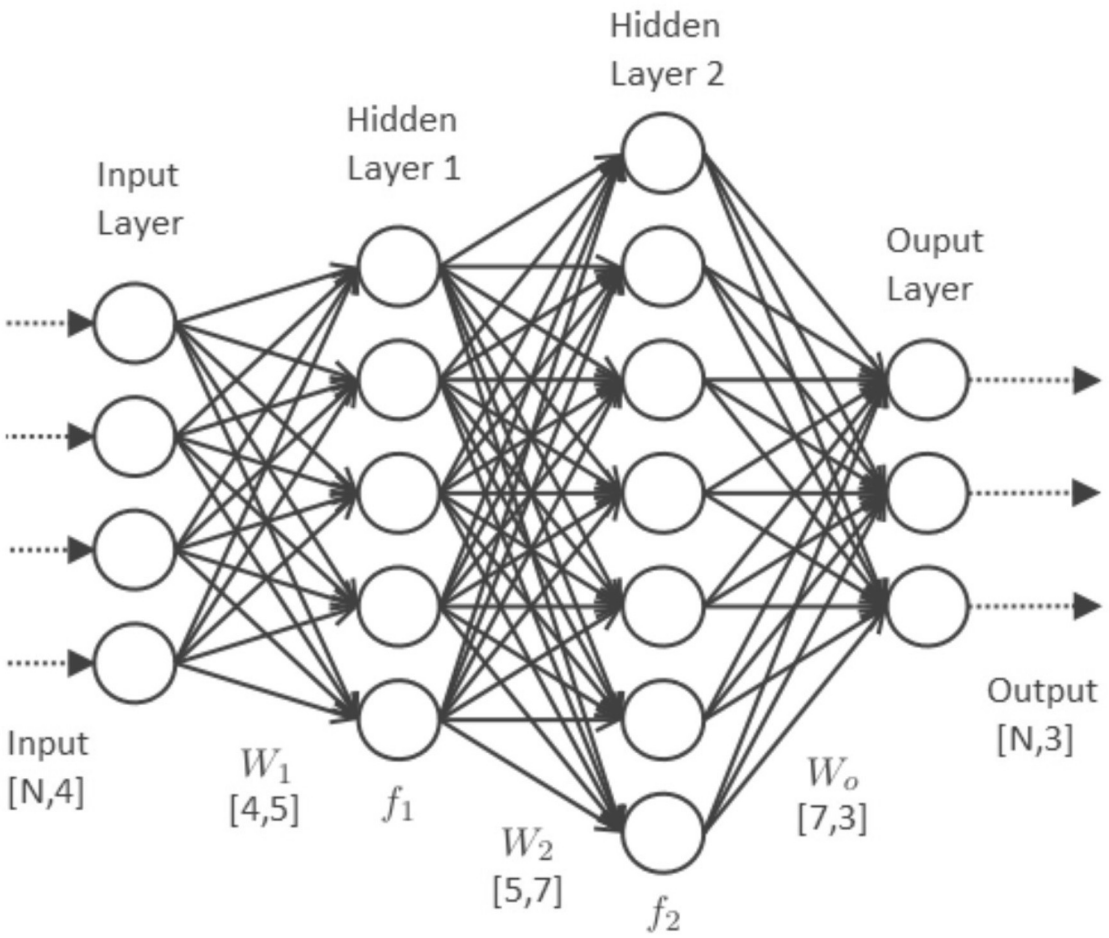
# Model ≈ specific type of function




Logistic function

- still using linear function $h(\mathbf{x})$
- $\hat{y} = \frac{1}{1+e^{-h(x)}}$
- Domain – real numbers
- Features - Unbounded
  Label - Bounded [0,1]

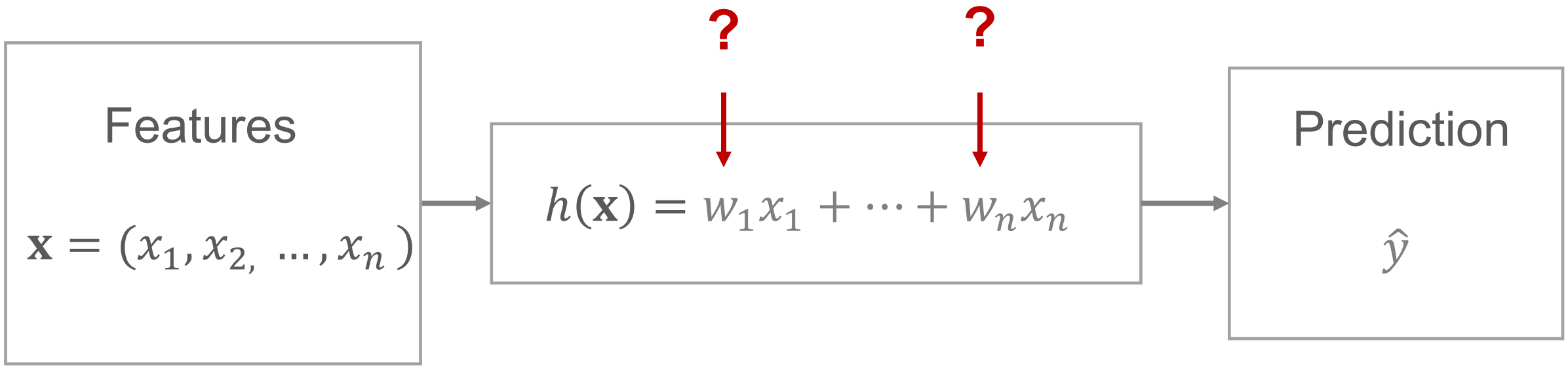
# Model ≈ specific type of functions



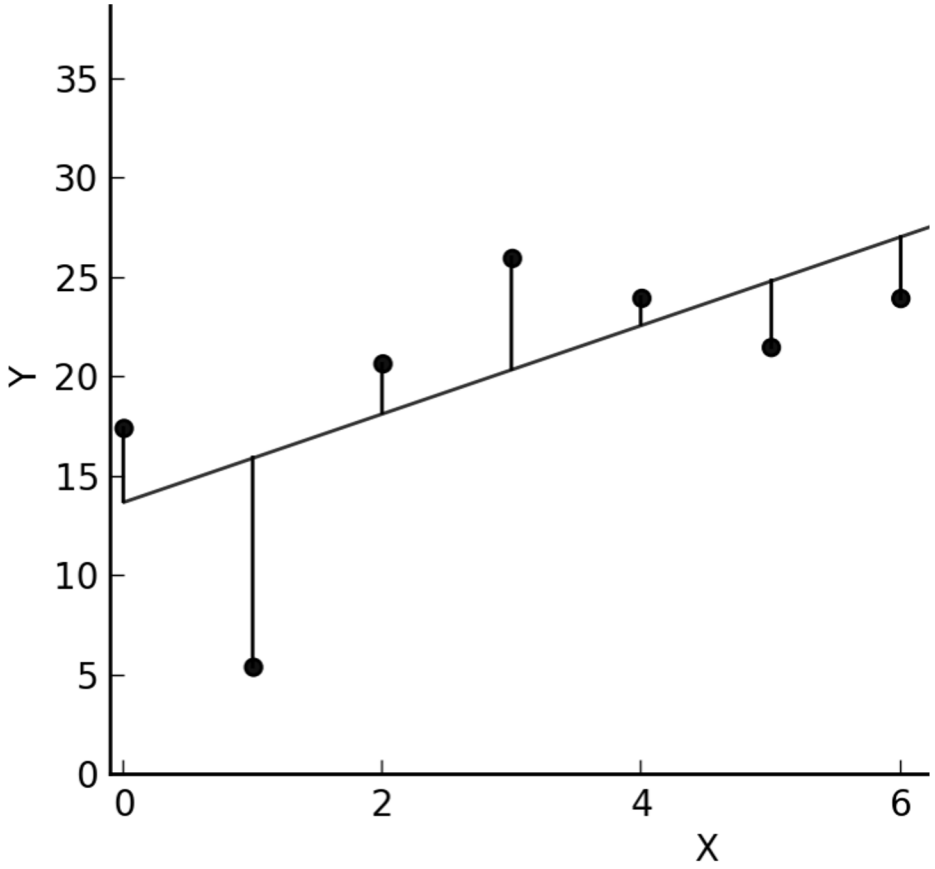

Artificial Neural Network

- $h(\mathbf{x}) = ???$
- Domain – real numbers
- Features - Unbounded
  Label - Bounded/ Unbounded

# How to choose best model?



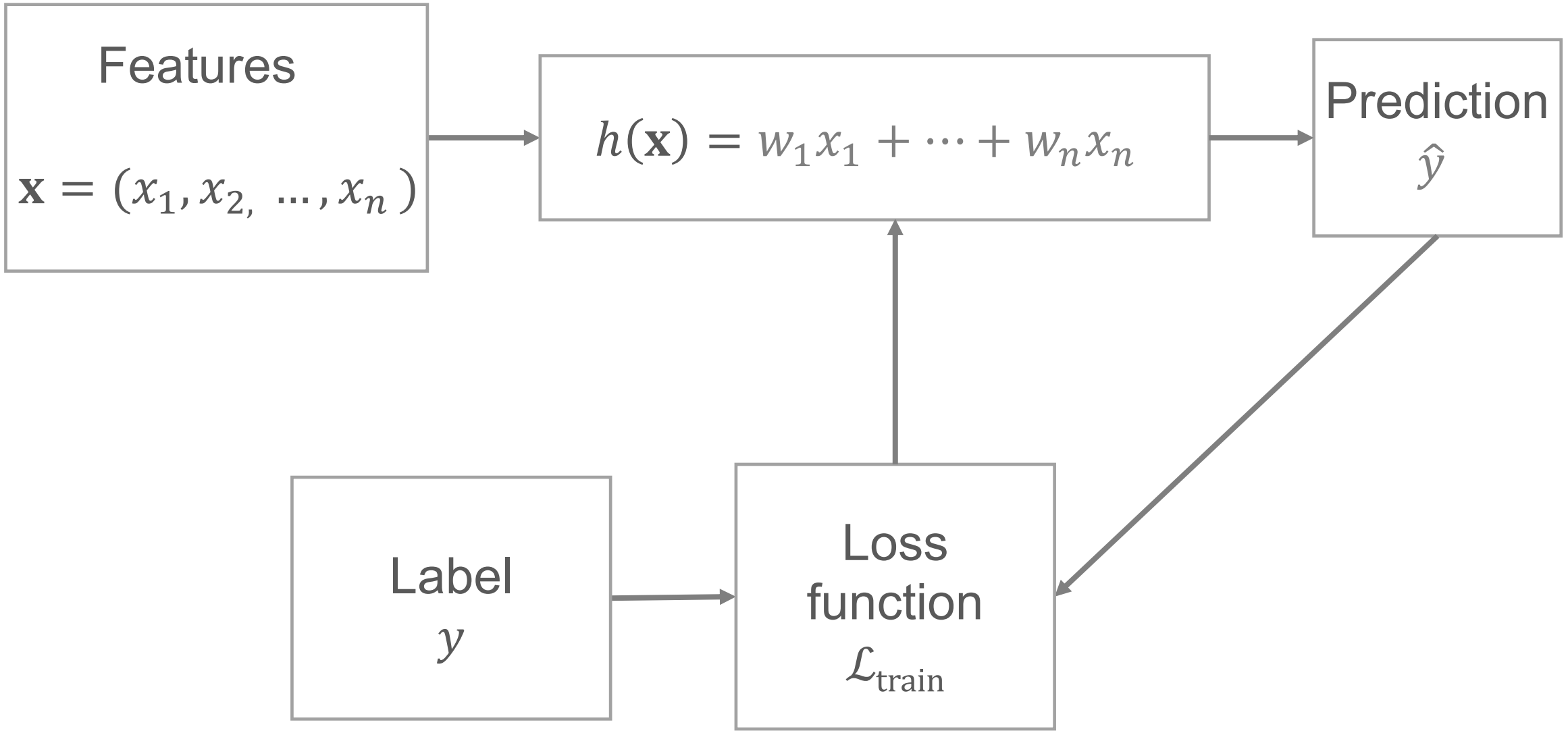
# How to choose best model?




Prediction should be close to labels

$$\hat{y} \approx y$$

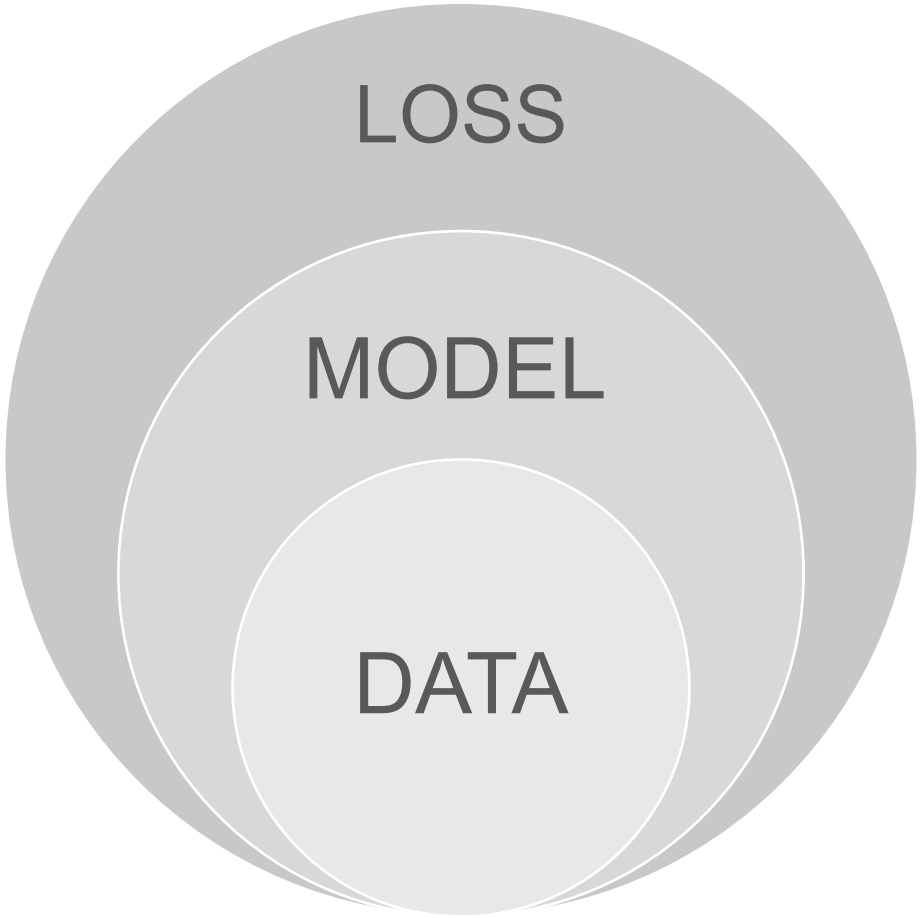
* less straightforward for unlabeled data


Features
$\mathbf{x} = (x_1, x_2, \ldots, x_n)$
$h(\mathbf{x}) = w_1 x_1 + \cdots + w_n x_n$
Prediction
$\hat{y}$
Label
$y$
Loss
function
$\mathcal{L}_{\text{train}}$

# Loss function

- Loss/ cost function - used to evaluate a quality of a model
- Compute training error (loss) $\mathcal{L}_{\text{train}}$
- Guide ML/ optimization algorithm to find best parameters of a model

# Three Components of ML




Finds best map

Maps features → label(s)

Consist of datapoints
Datapoint has features & label(s)

# Training a model is an iterative process

- Feed input (features) to model

Repeat block many times

- Compute prediction
- Compute training error (training loss) by comparing prediction and label
- Optimization algorithm uses training error (loss) to adjust model parameters (and hopefully improve a model)

# Linear regression - Data

- Data point – house
- Features (house area, n.o. rooms, …) - real-valued numbers
- Label (house price is XXX $) - real-valued number
- Assumption – features and label are linearly related

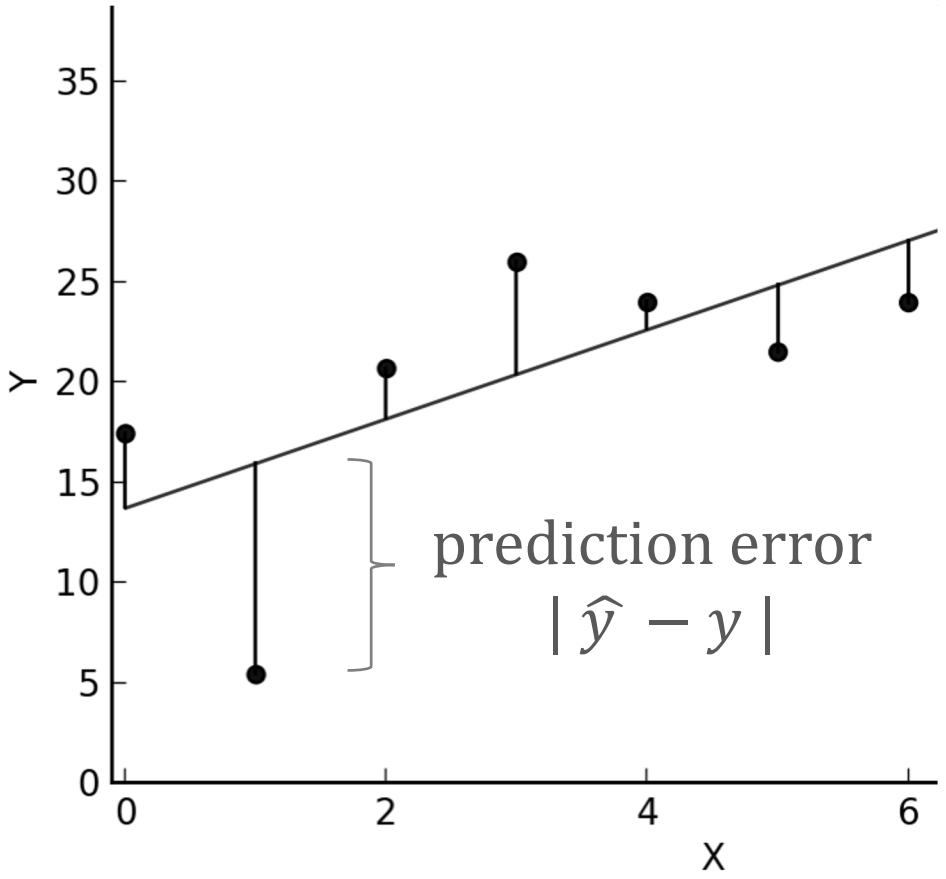
# Linear regression - Model

- Based on our assumption the model is

$$h(\mathbf{x}) = w_1 x_1 + \cdots + w_n x_n = \hat{y}$$

- Model is parametric (with params $w_1, \ldots, w_n$)
- N.o. parameters = n.o. features + 1 (bias or intercept)

# Linear regression - Loss

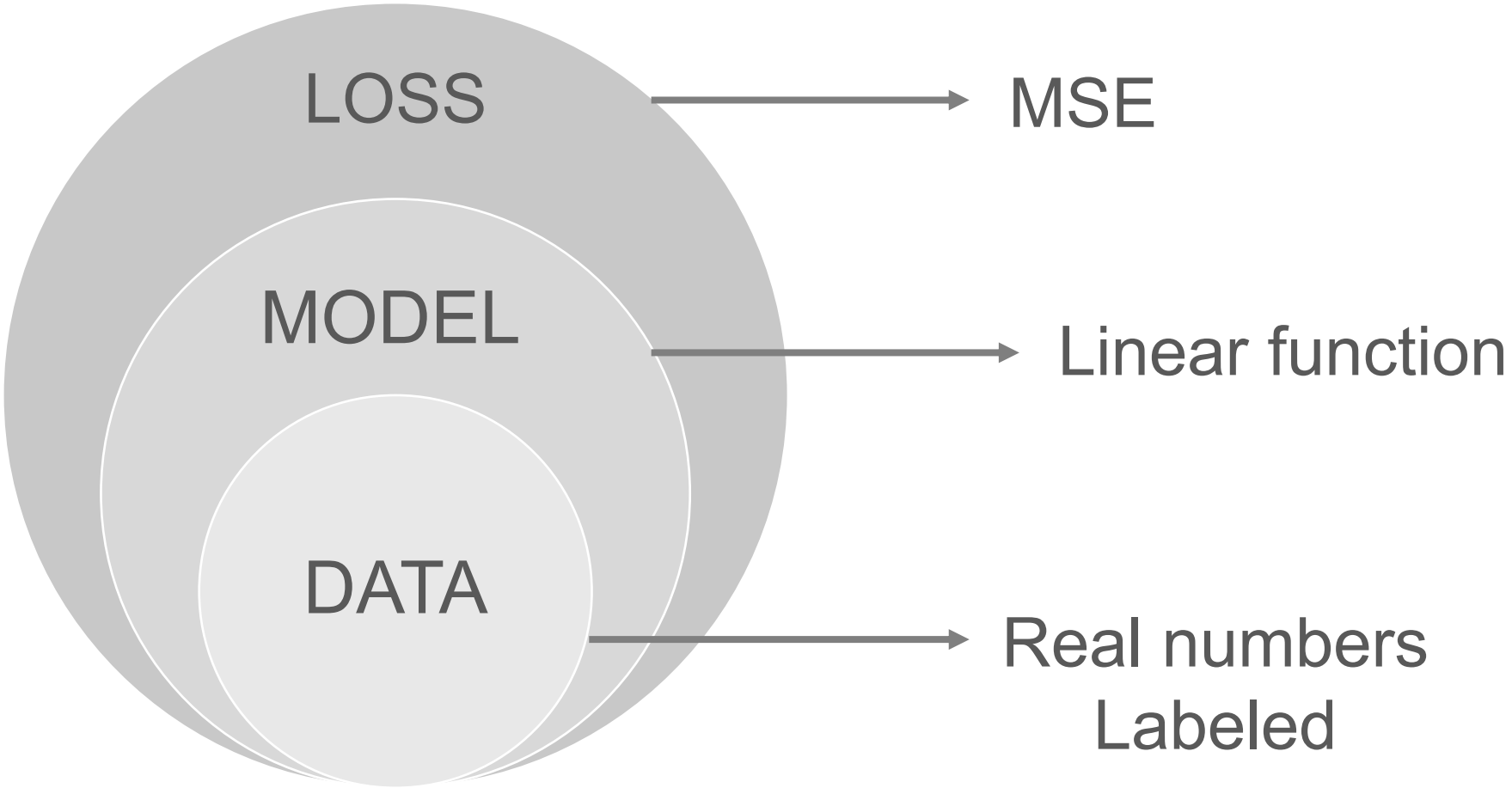



- Goal: $\hat{y} \approx y$
- Mean Squared Error (MSE)

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m}(\widehat{y_i} - y_i)^2$$

- Minimize MSE

# Linear regression



# Logistic regression - Data

- Data point – patient
- Features (BMI, blood test results, …) - real-valued numbers
- Label (disease or no disease) - categorical
- Assumption – features and log odds are linearly related

# Logistic regression - Data

Assumption – features and log odds are linearly related

$$\log \frac{p^{(i)}}{1-p^{(i)}} = h(\mathbf{x}^{(i)}) = w_1 x_1^{(i)} + \cdots + w_n x_n^{(i)}$$

- $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$ - feature vector for $\boldsymbol{i}$-th datapoint
- $p^{(i)}$ - probability of an event (“patient has disease”) for $\boldsymbol{i}$-th datapoint

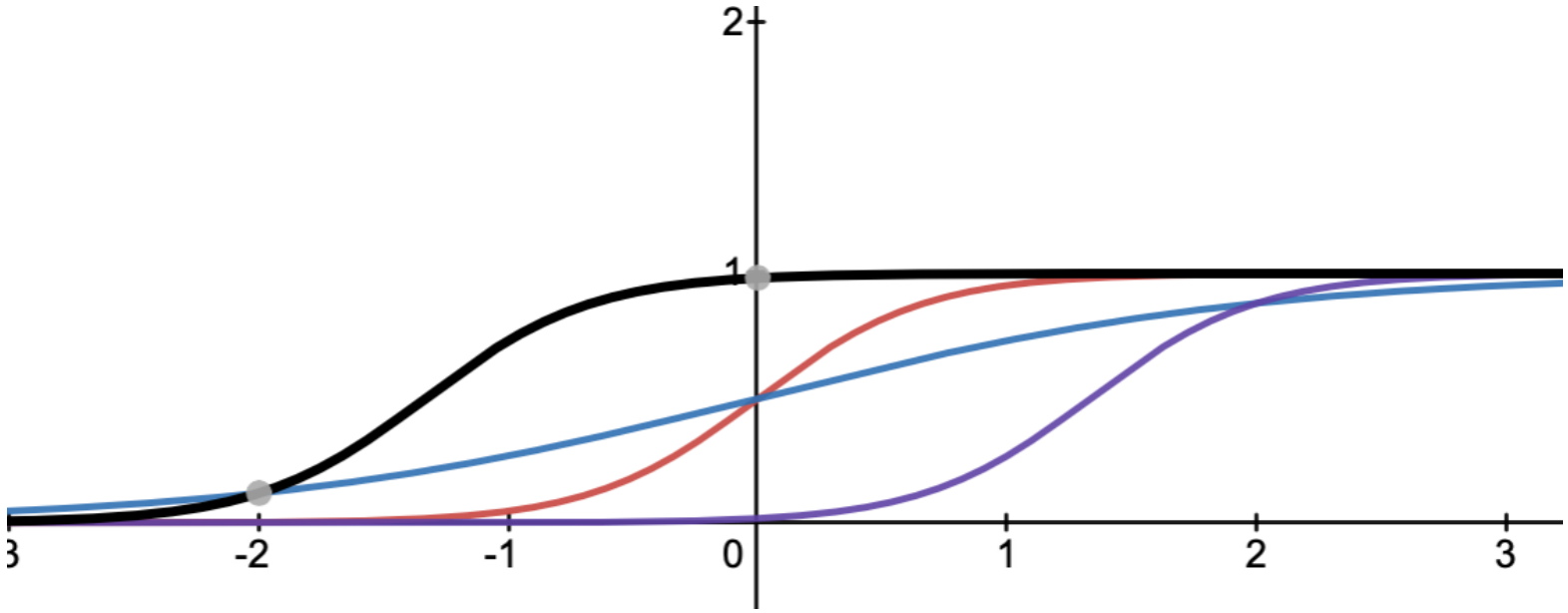
# Logistic regression - Model

Assumption – features and log odds are linearly related

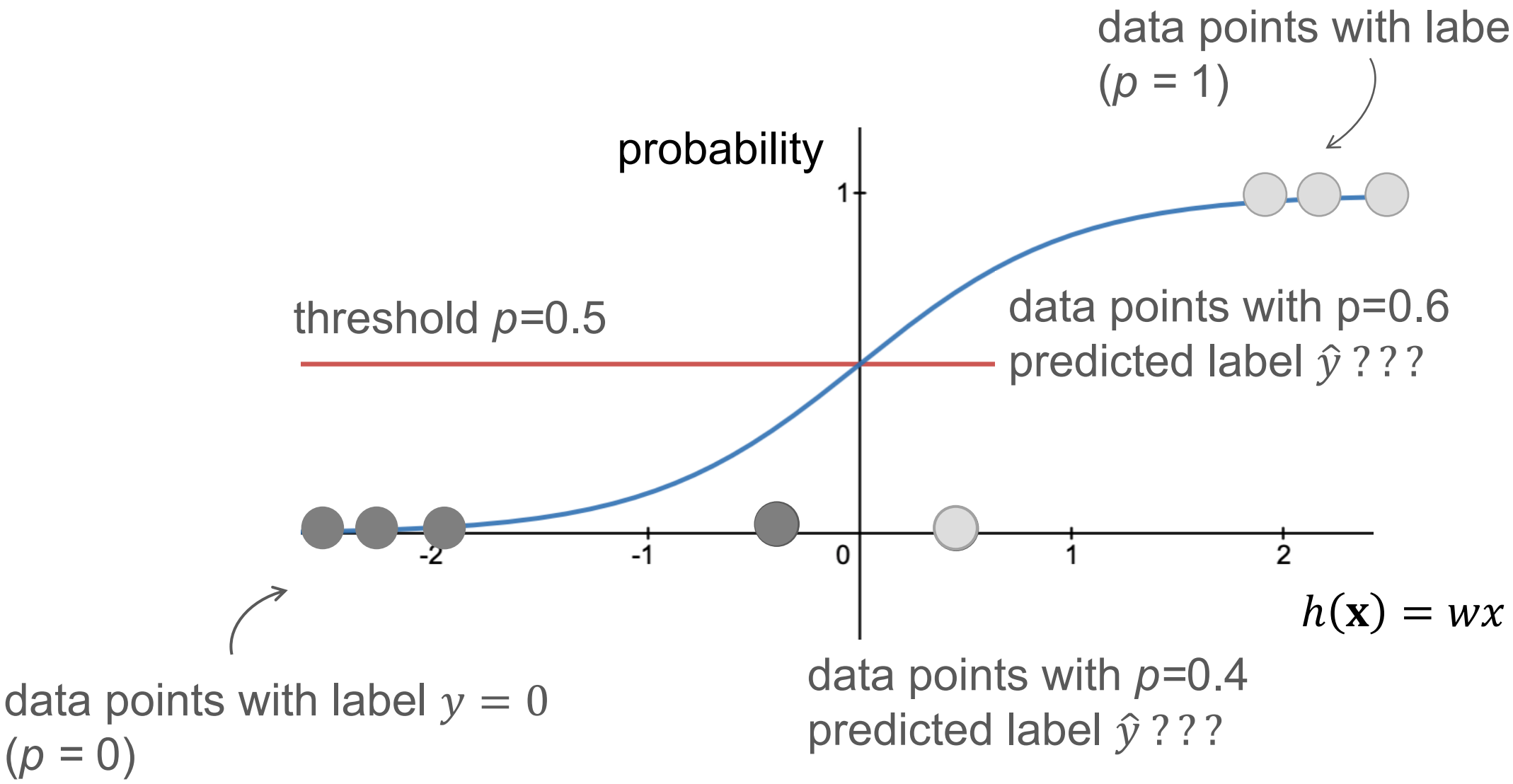
$$\ln \frac{p}{1-p} = h(\mathbf{x}) = w_1 x_1 + \cdots + w_n x_n$$

→ $p = \frac{1}{1+e^{-h(\mathbf{x})}}$ - logistic/ sigmoid function

# Logistic regression - Model




logistic/ sigmoid function
range – [0,1]


data points with label $y = 1$
($p$ = 1)
probability
1
threshold $p$=0.5
data points with p=0.6
predicted label $\hat{y}$ ? ? ?
-2
-1
0
1
2
$h(\mathbf{x}) = wx$
data points with label $y = 0$
($p$ = 0)
data points with $p$=0.4
predicted label $\hat{y}$ ? ? ?

# Logistic regression - Loss

- Goal: $\hat{y} \approx y$
- For data points with label $y = 1$ ("disease"):

  choose model parameters $w_1, w_2, \ldots, w_n$
  s.t. probability $p$ as close to 1 as possible

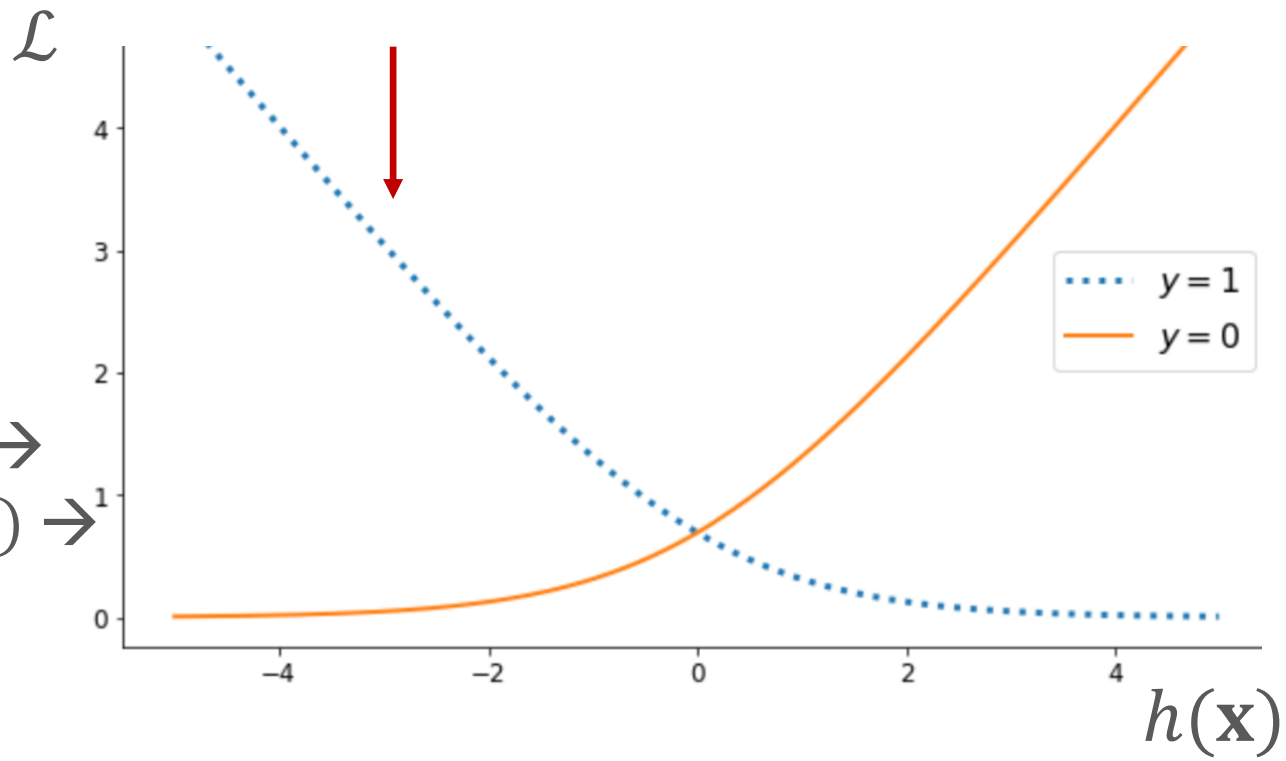
- For data points with label $y = 0$ ("no disease"):

  choose model parameters $w_1, w_2, \ldots, w_n$
  s.t. probability $p$ as close to 0 as possible

# Logistic Loss

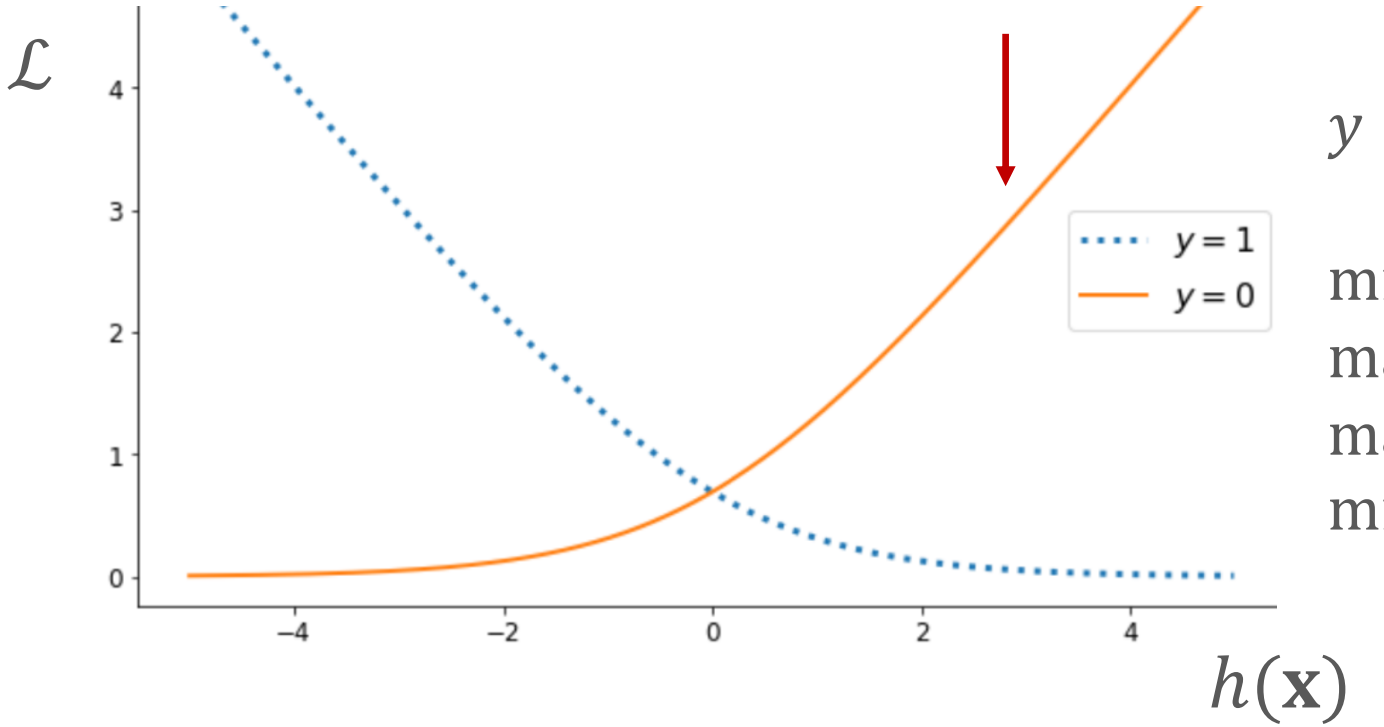
$y = 1$ (“disease”):

maximize $p$ →
maximize $ln(p)$ →
minimize $-ln(p)$ →




$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\, ln\left(p^{(i)}\right) - \left(1 - y^{(i)}\right) ln\left(1 - p^{(i)}\right)$$

# Logistic Loss

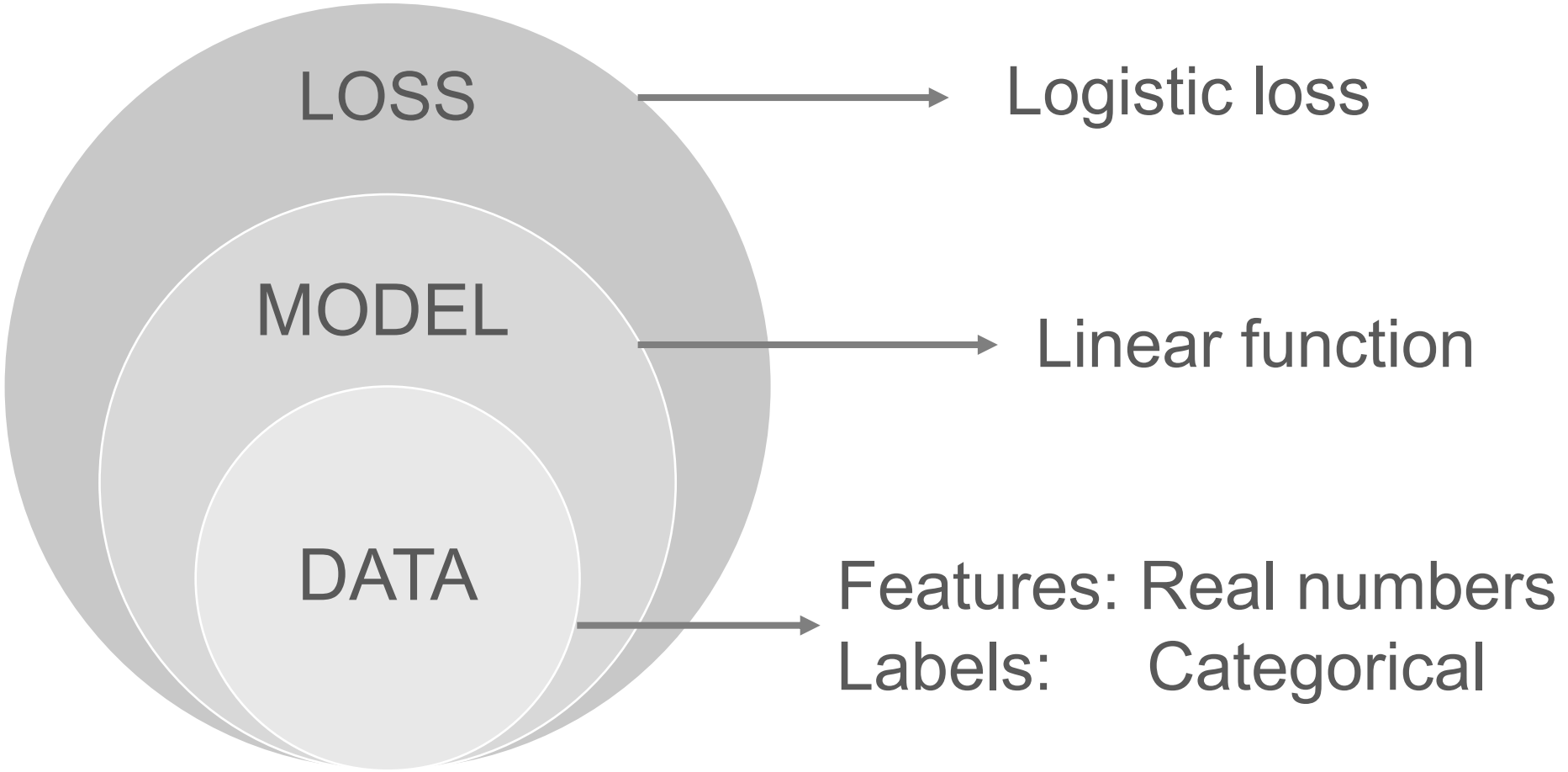



$y = 0$ (“no disease”):

| | |
|---|---|
| minimize | $p \rightarrow$ |
| maximize | $(1 - p) \rightarrow$ |
| maximize | $ln(1 - p) \rightarrow$ |
| minimize. | $-ln(1 - p)$ |

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\, ln\left(p^{(i)}\right) - \left(1 - y^{(i)}\right) ln\left(1 - p^{(i)}\right)$$

# Logistic regression



# ”Hello world” of ML: Iris dataset

**iris setosa**


**iris versicolor**


**iris virginica**

# ”Hello world” of ML: Iris dataset


Data →

Model + Loss →

??? →

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression

# load data
iris = datasets.load_iris()
iris['feature_names'], iris['target_names']
```

```
(['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 array(['setosa', 'versicolor', 'virginica'], dtype='<U10'))
```

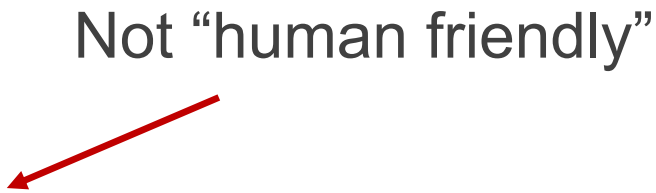
```
# features & labels
X, y = iris.data, iris.target

# model
clf = LogisticRegression(random_state=0, max_iter=500).fit(X, y)
clf.predict(X[:2, :])

clf.predict_proba(X[:2, :])
clf.score(X, y)
```

```
0.9733333333333334
```

# Loss function

Loss/ cost function - used to evaluate a quality of a model.

For labelled data:




- Compute training error
- Guide ML/ optimization algorithm to find best parameters of a model

# Score/ Metrics

- Compute training error (loss)
- Easy to interpret
- Often **cannot** be used for model training (optimization)

# Score/ Metrics

- Regression: $R^2$, MAE, RMSE, …
- Classification: Accuracy, F-score, precision, confusion matrix, …