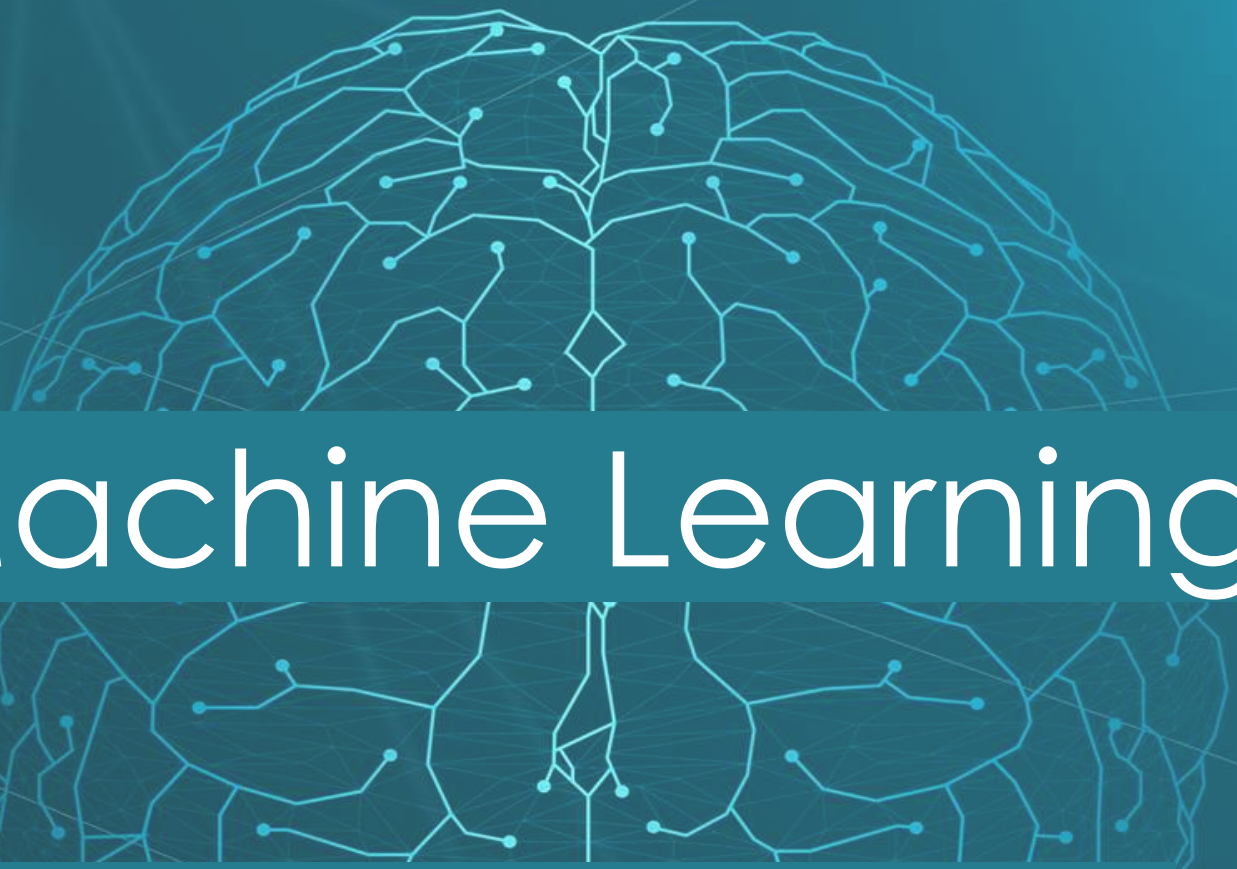
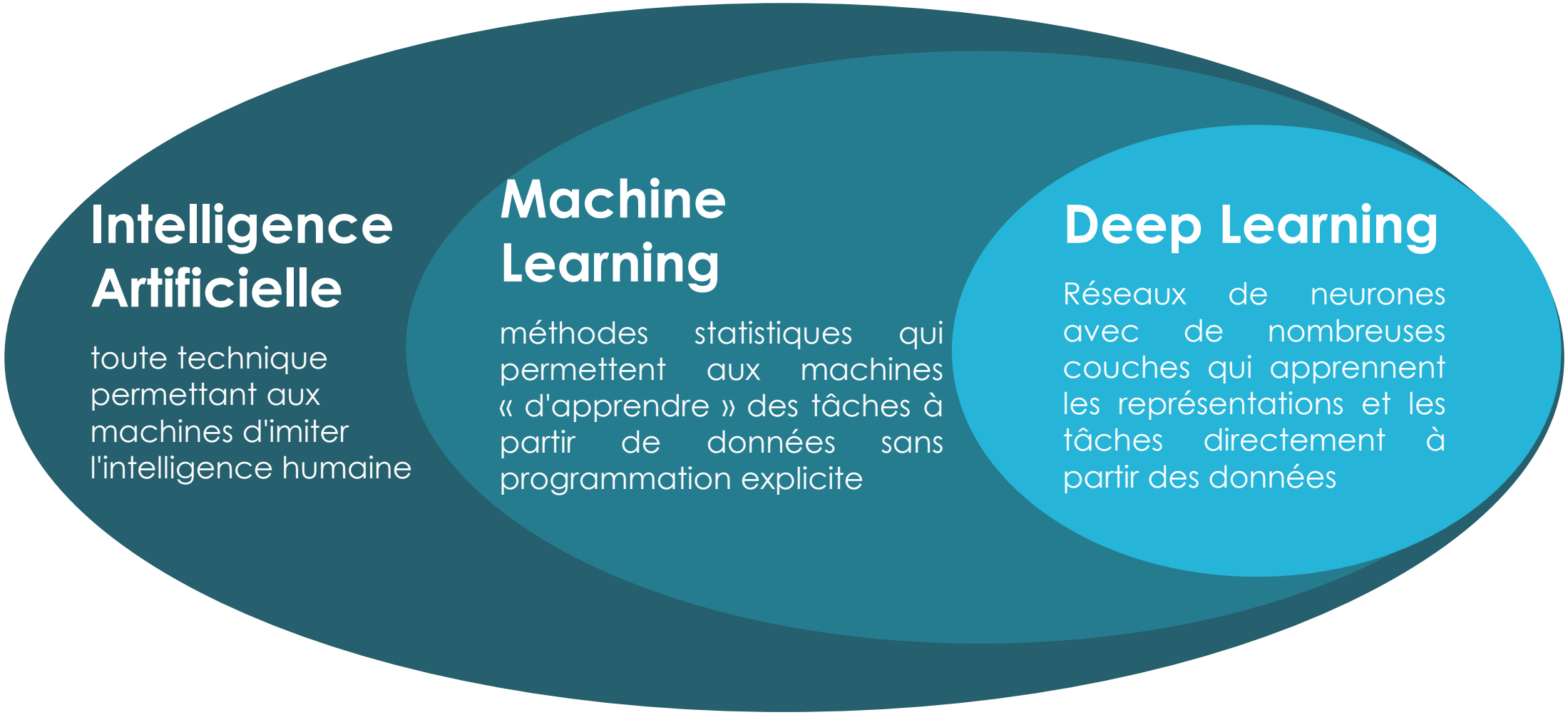
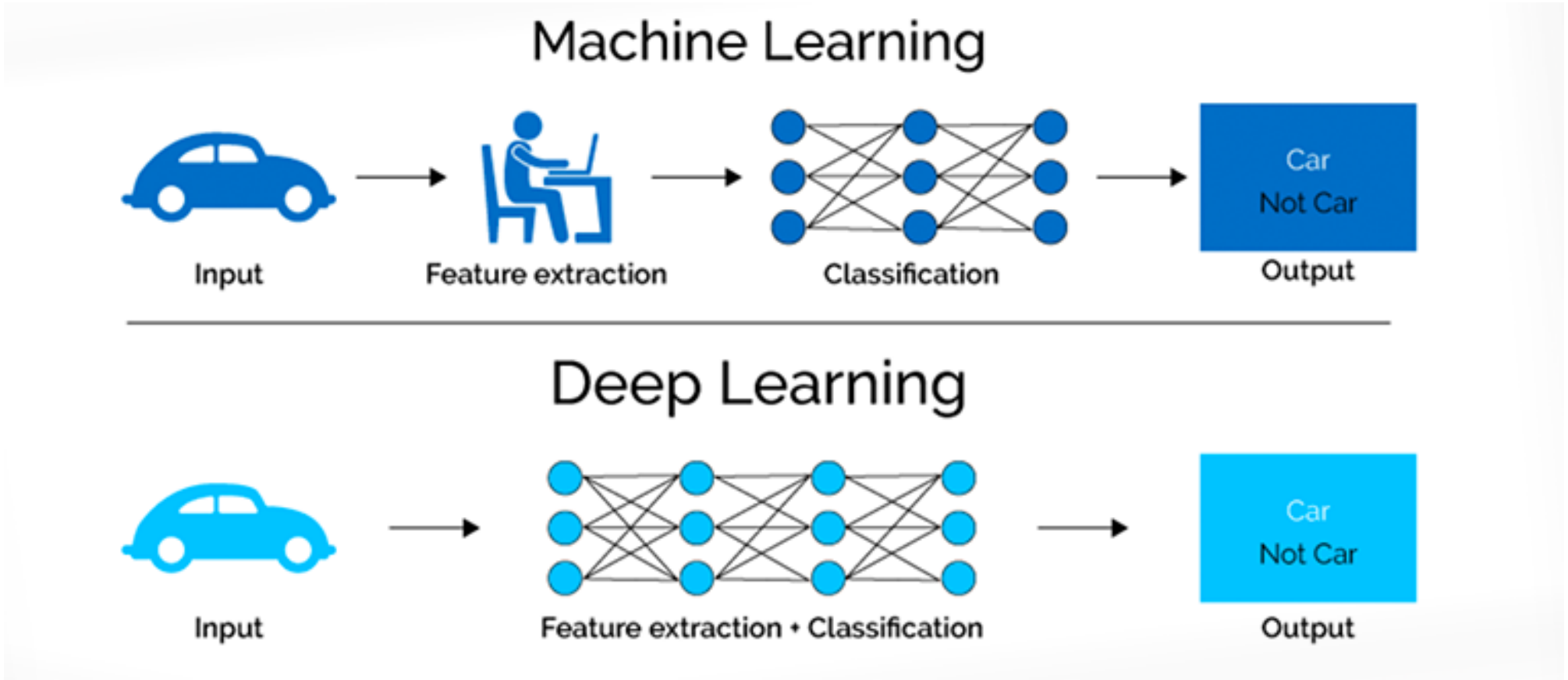
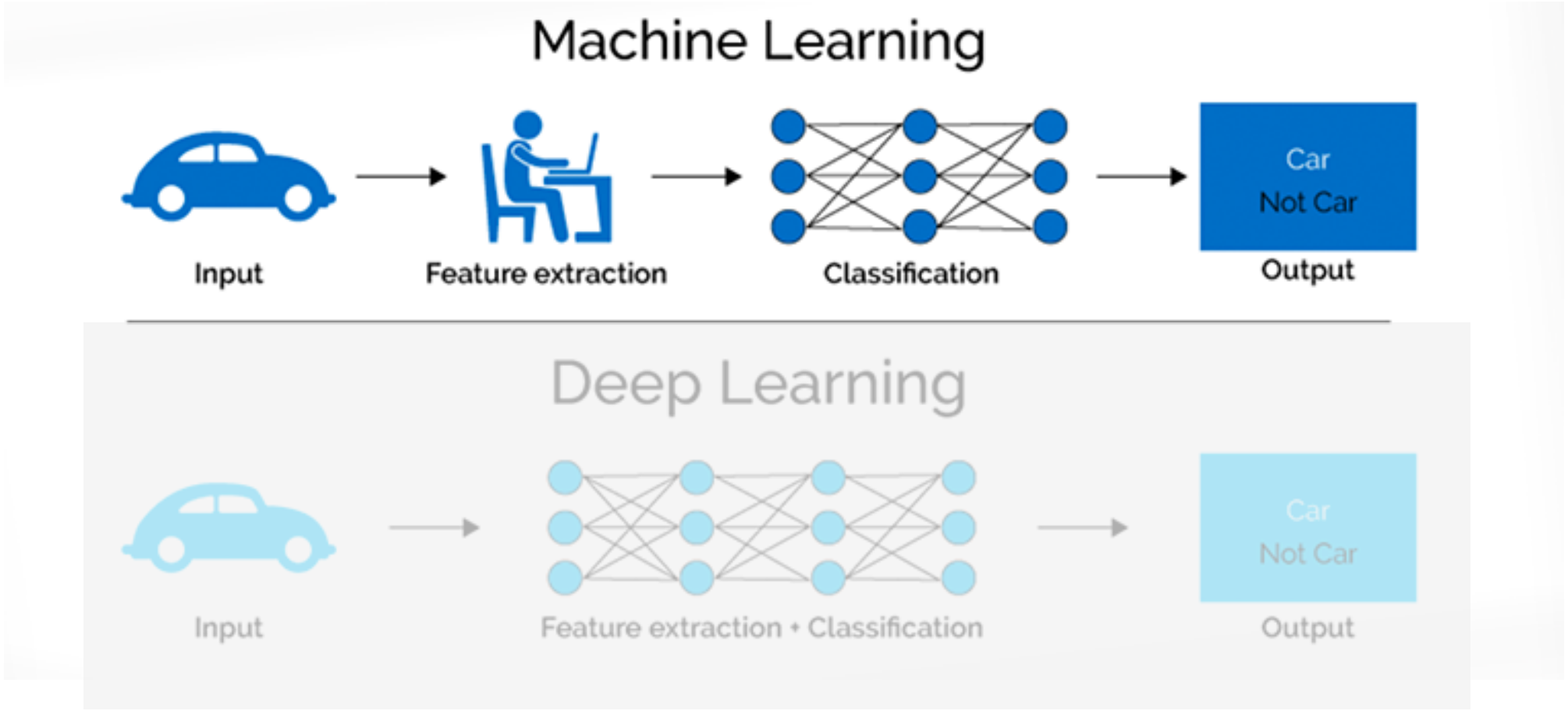


Machine Learning

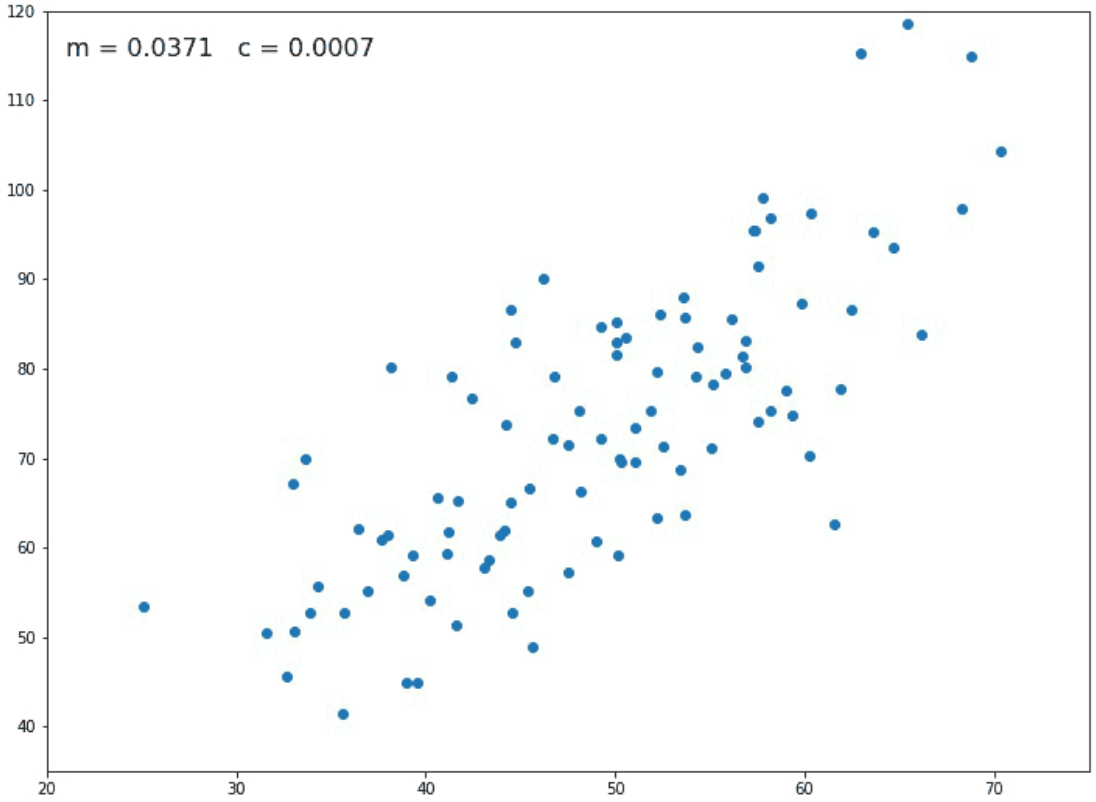
Abdelmoudjib BENTERKI
abdelmoudjib.benterki@estaca.fr


Intelligence Artificielle
toute technique permettant aux machines d'imiter l'intelligence humaine
Machine Learning
méthodes statistiques qui permettent aux machines « d'apprendre » des tâches à partir de données sans programmation explicite
Deep Learning
Réseaux de neurones avec de nombreuses couches qui apprennent les représentations et les tâches directement à partir des données

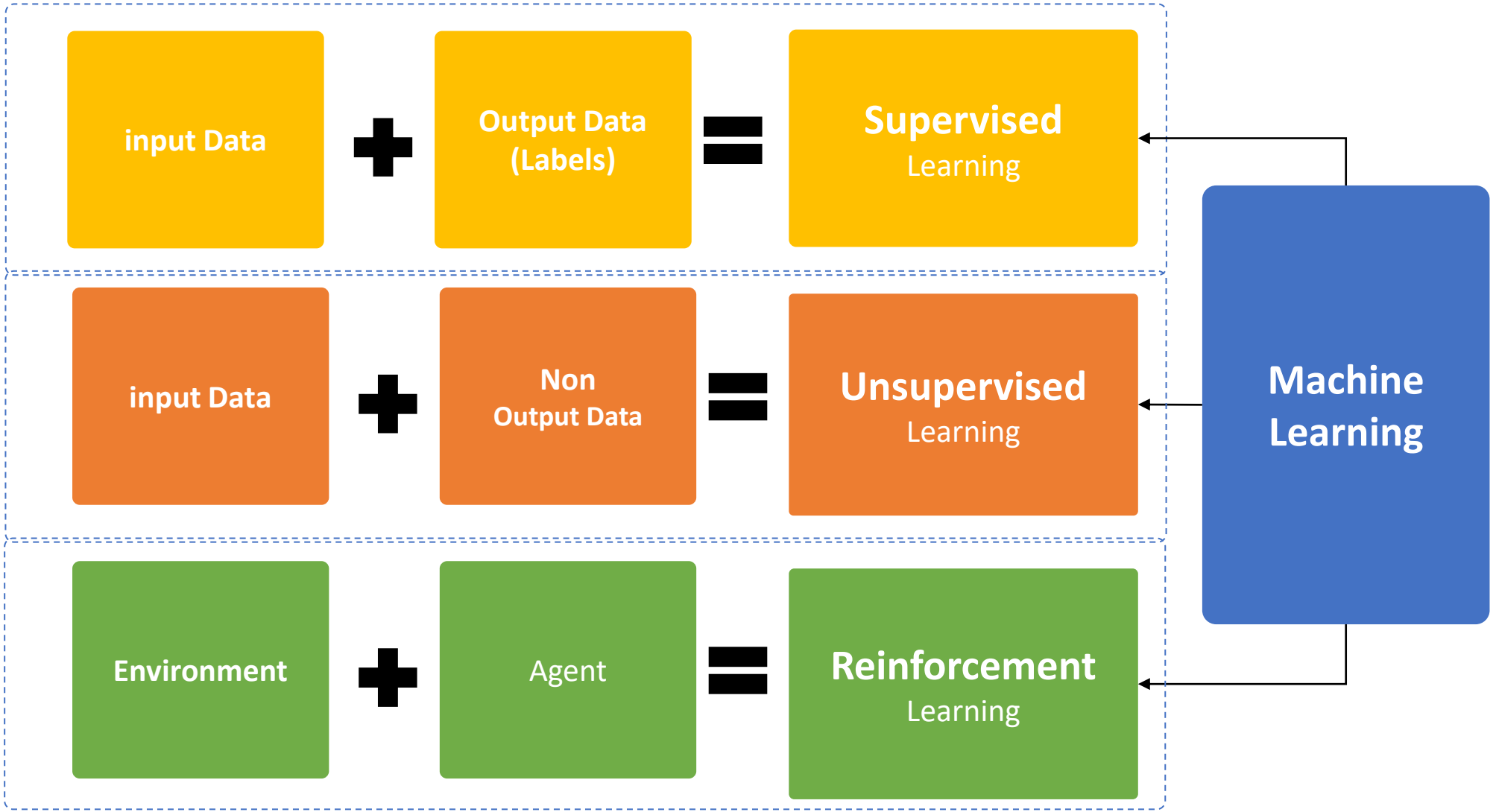

Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

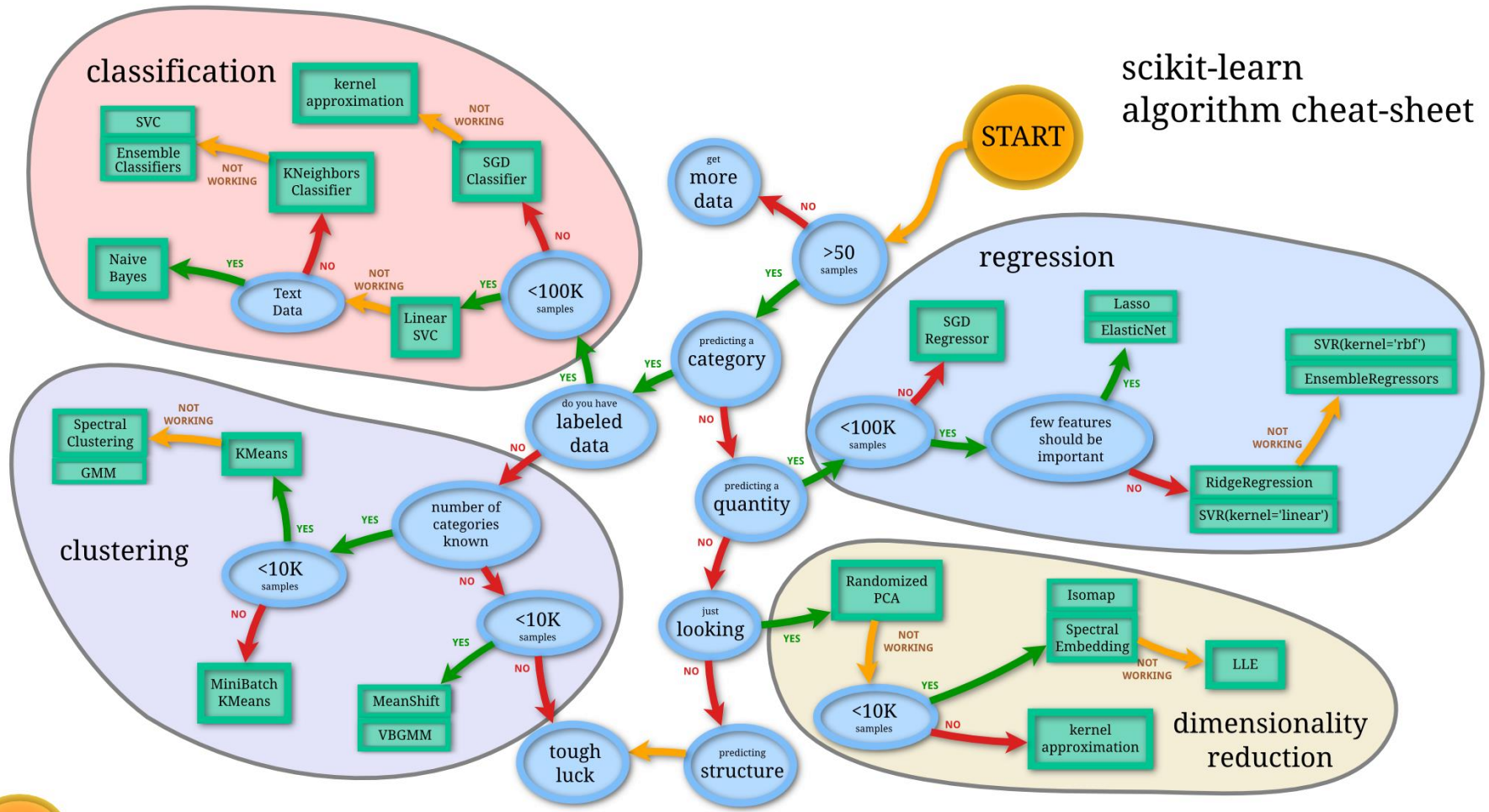

Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

Develop a MODEL from DATA is a ML



Data → **Preprocessing** → **Training** → **Testing** → **Prediction**


input Data
Output Data (Labels)
Supervised Learning
input Data
Non Output Data
Unsupervised Learning
Environment
Agent
Reinforcement Learning
Machine Learning

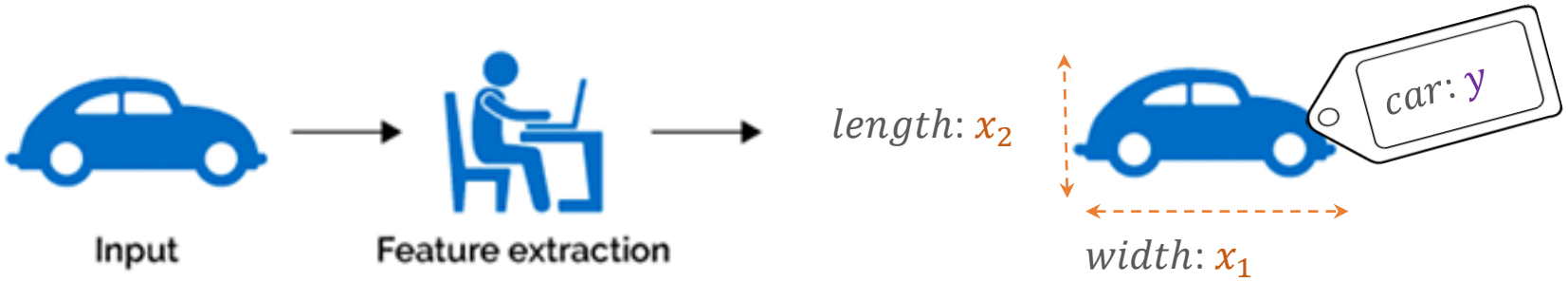

scikit-learn
algorithm cheat-sheet
START
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
NO
Naive Bayes
YES
NO
Text Data
NOT WORKING
Linear SVC
YES
<100K samples
NO
get more data
NO
>50 samples
YES
predicting a category
YES
NO
do you have labeled data
YES
NO
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
NO
<100K samples
YES
few features should be important
YES
NOT WORKING
RidgeRegression
SVR(kernel='linear')
NO
predicting a quantity
YES
NO
clustering
Spectral Clustering
NOT WORKING
GMM
KMeans
YES
number of categories known
YES
<10K samples
NO
NO
<10K samples
YES
MiniBatch KMeans
MeanShift
VBGMM
NO
just looking
YES
NO
Randomized PCA
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
YES
<10K samples
NO
kernel approximation
dimensionality reduction
tough luck
predicting structure


python™


Google colab
GETTING STARTED WITH
GOOGLE COLAB

Back

COLLECT DATA | PRE-PROCESSING | TRAINING | PREDICTION | TESTING

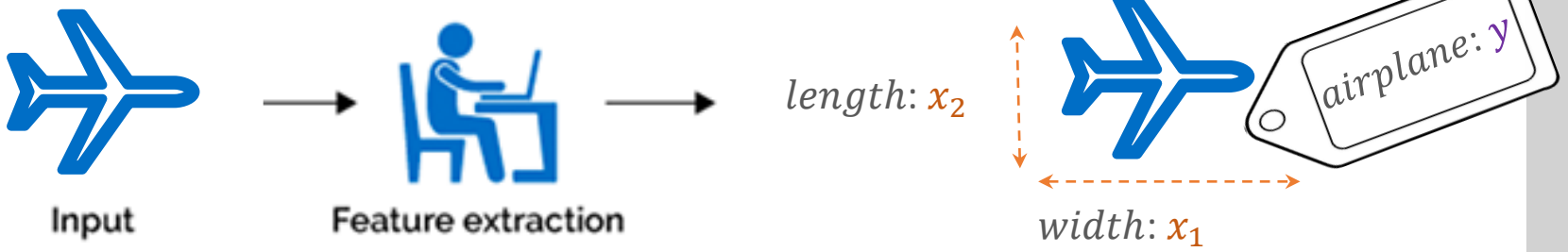


DATASET

Features | Labels or Target

samples	x_1	x_2	y
1	5.3	5.9	car
2	10.0	2.3	car
3	6.2	5.9	car

COLLECT DATA | PRE-PROCESSING | TRAINING | PREDICTION | TESTING



DATASET

	Features		Labels or Target
samples	x_1	x_2	y
1	5.3	5.9	car
2	10.0	2.3	car
3	6.2	5.9	car
4	50	Nan	airplane
5	60	18	airplane
6	56	32	airplane

Machine Learning
Life Cycle

COLLECT DATA | PRE-PROCESSING | TRAINING | PREDICTION | TESTING

Types of Data

- Numerical Data
- Categorical Data
- Time Series Data
- Text Data

Various Sources of Dataset

- Google Dataset Search Engine
- Microsoft Dataset
- Computer Vision Dataset
- Kaggle Dataset
- Amazon Dataset
- Open Images Dataset v4
- Google-Landmarks Dataset

Open Datasets for Autonomous Driving Projects

- KITTI Vision Benchmark Suite
- nuScenes Dataset
- A2D2 Dataset
- ApolloScape Dataset
- Argoverse Dataset
- Berkeley DeepDrive Dataset
- CityScapes Dataset
- Comma2k19 Dataset
- LeddarTech PixSet Dataset
- Level 5 Open Data
- Oxford Radar RobotCar Dataset
- PandaSet

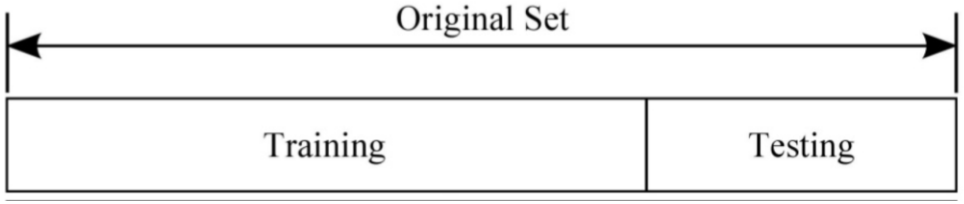
1- Manage missing data « Nan (Not a Number)»:

- supprimer la ligne
- remplir par la moyenne, valeur qui se répète le plus, ou …

2- Manage Categorical Variables« car, airplane »

- remplacer « car » par « 0 »
- Remplacer « airplane » par « 1 »

3- Split the dataset between Training and Test set



4- Apply Feature Scaling

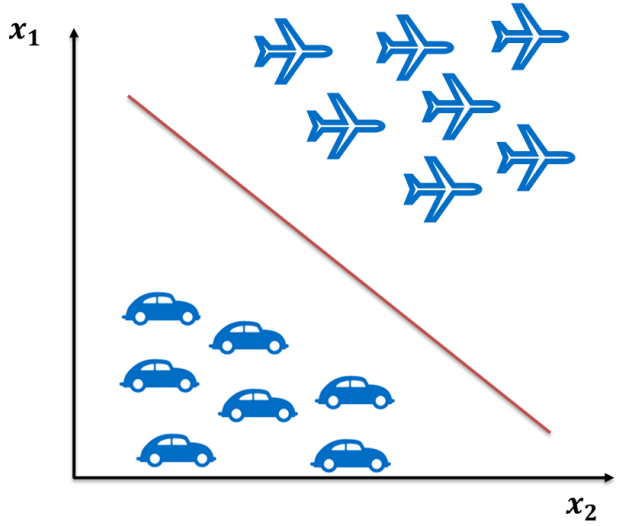
- Normaliser les donnée entre [0, 1] ou [-1, 1]

samples	x_1	x_2	y
1	5.3	5.9	car
2	10.0	2.3	car
3	6.2	5.9	car
4	**Nan**	5.9	car
5	10.0	2.3	car
6	6.2	5.9	car
7	6.2	5.9	car
8	50	**Nan**	airplane
9	60	18	airplane
10	56	32	airplane
11	50	20	airplane
⋮			
n	56	32	airplane

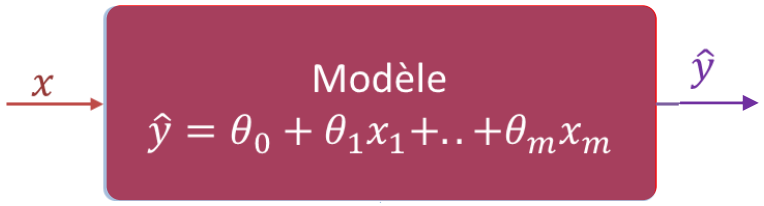
COLLECT DATA | PRE-PROCESSING | **TRAINING** | PREDICTION | TESTING

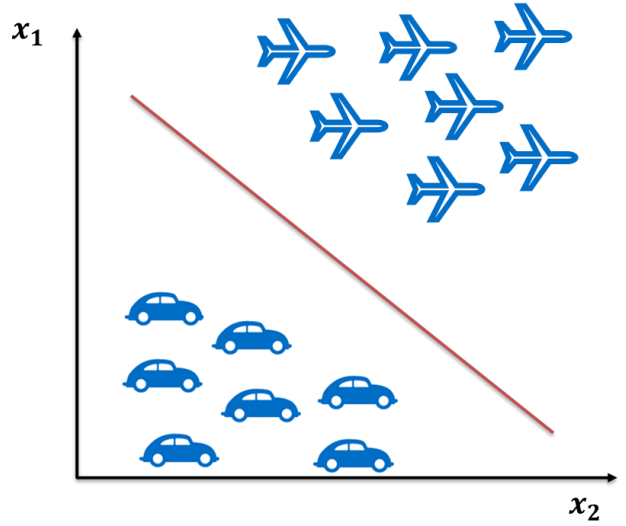
samples	x_1	x_2	y
1	5.3	5.9	0
20	10.0	2.3	0
3	6.2	5.9	0
42	**6.2**	5.9	0
5	10.0	2.3	0
6	6.2	5.9	0
73	6.2	5.9	0
8	50	**5.9**	1
90	60	18	1
10	56	32	1
120	50	20	1
⋮			
n_tr	56	32	1

COLLECT DATA | PRE-PROCESSING | **TRAINING** | PREDICTION | TESTING



samples	x_1	x_2	y
1	5.3	5.9	0
20	10.0	2.3	0
3	6.2	5.9	0
42	**6.2**	5.9	0
5	10.0	2.3	0
6	6.2	5.9	0
73	6.2	5.9	0
8	50	**5.9**	1
90	60	18	1
10	56	32	1
120	50	20	1
⋮			
n_tr	56	32	1



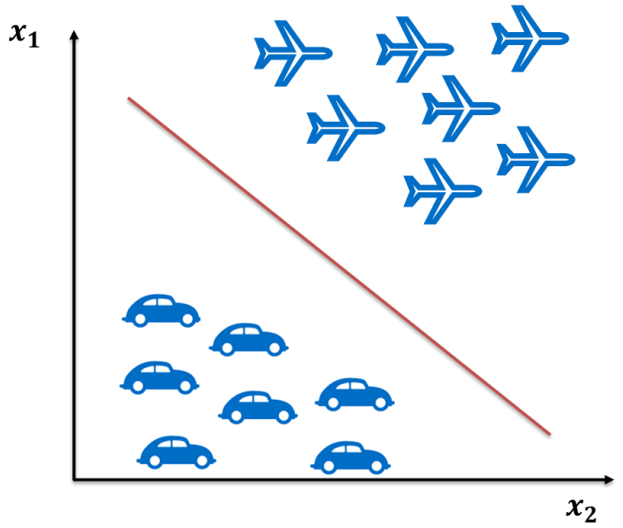


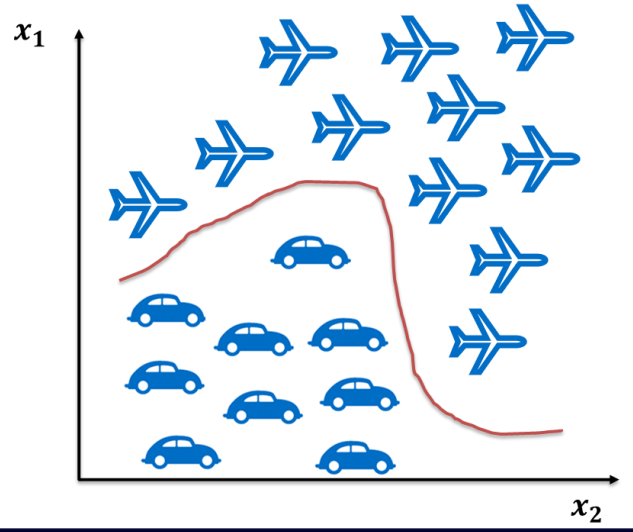
samples	x_1	x_2	y
1	5.3	5.9	0
20	10.0	2.3	0
3	6.2	5.9	0
42	**6.2**	5.9	0
5	10.0	2.3	0
6	6.2	5.9	0
73	6.2	5.9	0
8	50	**5.9**	1
90	60	18	1
10	56	32	1
120	50	20	1
⋮			
n_tr	56	32	1

COLLECT DATA | PRE-PROCESSING | **TRAINING** | PREDICTION | TESTING

x → Modèle $\hat{y} = \theta_0 + \theta_1 x_1 + .. + \theta_m x_m$ → $\hat{y}$

x → Modèle $\hat{y} = f(x, \theta)$ → $\hat{y}$





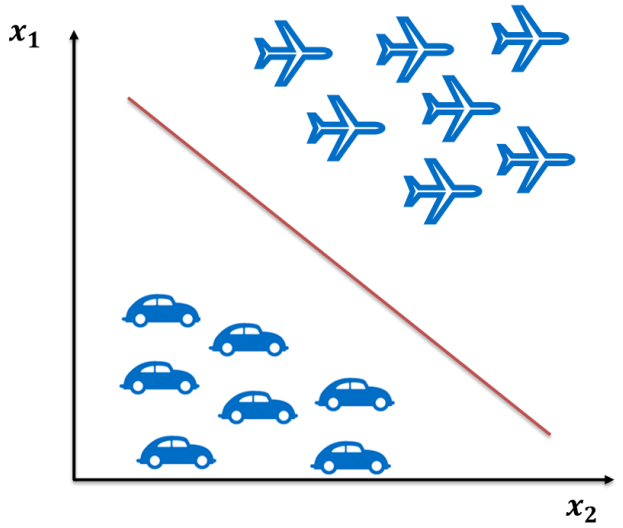
samples	x_1	x_2	y
1	5.3	5.9	0
20	10.0	2.3	0
3	6.2	5.9	0
42	**6.2**	5.9	0
5	10.0	2.3	0
6	6.2	5.9	0
73	6.2	5.9	0
8	50	**5.9**	1
90	60	18	1
10	56	32	1
120	50	20	1
⋮			
n_tr	56	32	1

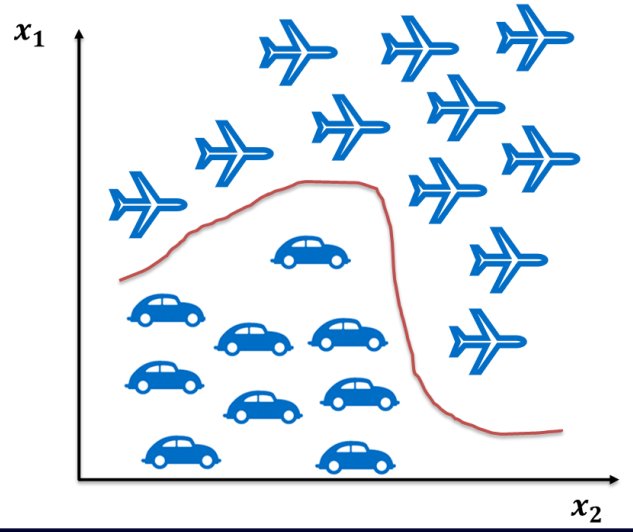
COLLECT DATA | PRE-PROCESSING | **TRAINING** | PREDICTION | TESTING

x → Modèle $\hat{y} = \theta_0 + \theta_1 x_1 + .. + \theta_m x_m$ → $\hat{y}$

How to find the optimal parameters Θ^*?

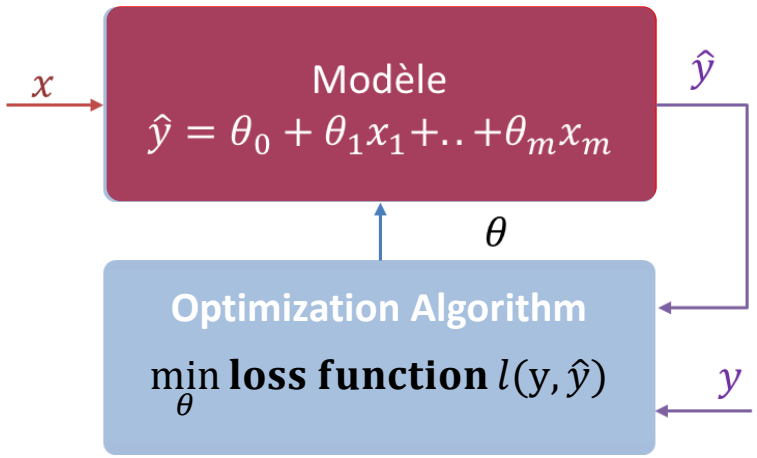
x → Modèle $\hat{y} = f(x, \theta)$ → $\hat{y}$

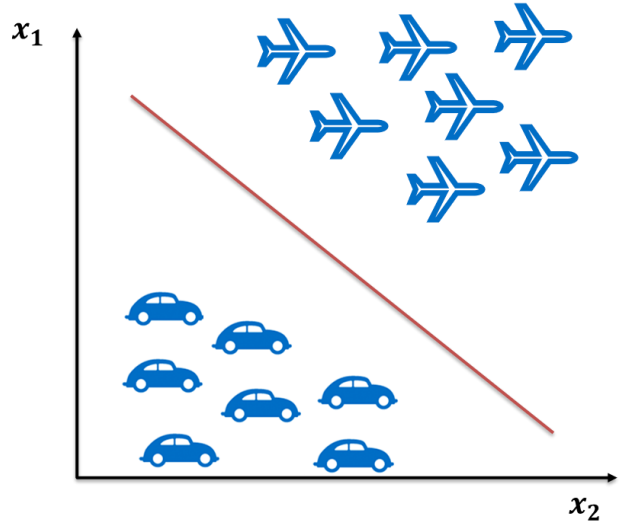




samples	x_1	x_2	y
1	5.3	5.9	0
20	10.0	2.3	0
3	6.2	5.9	0
42	**6.2**	5.9	0
5	10.0	2.3	0
6	6.2	5.9	0
73	6.2	5.9	0
8	50	**5.9**	1
90	60	18	1
10	56	32	1
120	50	20	1
⋮			
n_tr	56	32	1

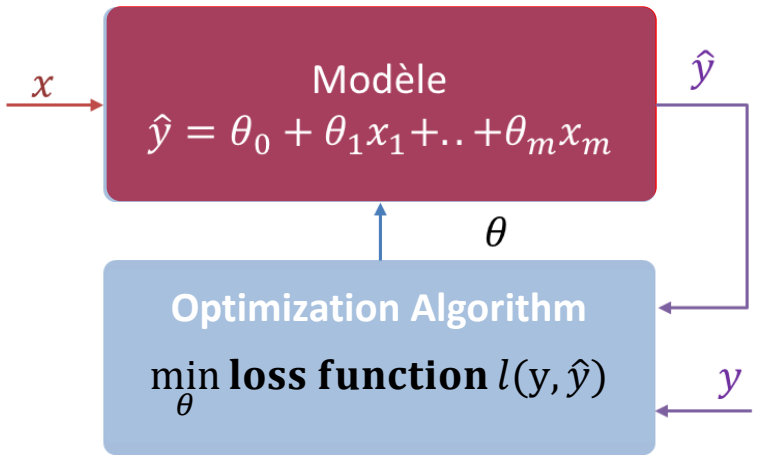
COLLECT DATA | PRE-PROCESSING | **TRAINING** | PREDICTION | TESTING

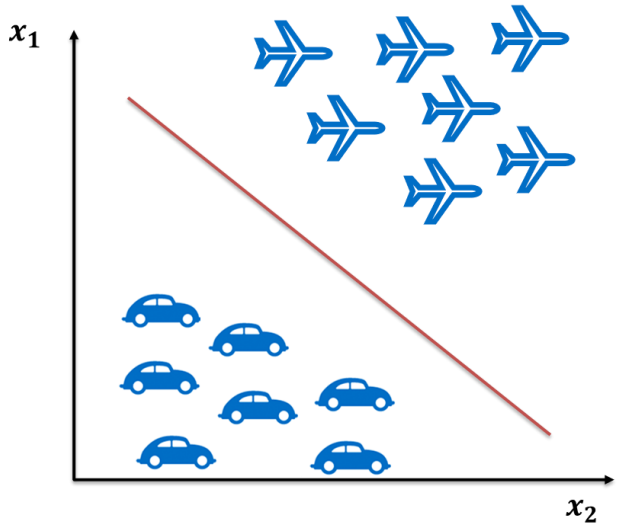




samples	x_1	x_2	y
1	5.3	5.9	0
20	10.0	2.3	0
3	6.2	5.9	0
42	**6.2**	5.9	0
5	10.0	2.3	0
6	6.2	5.9	0
73	6.2	5.9	0
8	50	**5.9**	1
90	60	18	1
10	56	32	1
120	50	20	1
⋮			
n_tr	56	32	1

COLLECT DATA | PRE-PROCESSING | **TRAINING** | PREDICTION | TESTING





samples	x_1	x_2	y
1	5.3	5.9	0
20	10.0	2.3	0
3	6.2	5.9	0
42	**6.2**	5.9	0
5	10.0	2.3	0
6	6.2	5.9	0
73	6.2	5.9	0
8	50	**5.9**	1
90	60	18	1
10	56	32	1
120	50	20	1
⋮			
n_tr	56	32	1

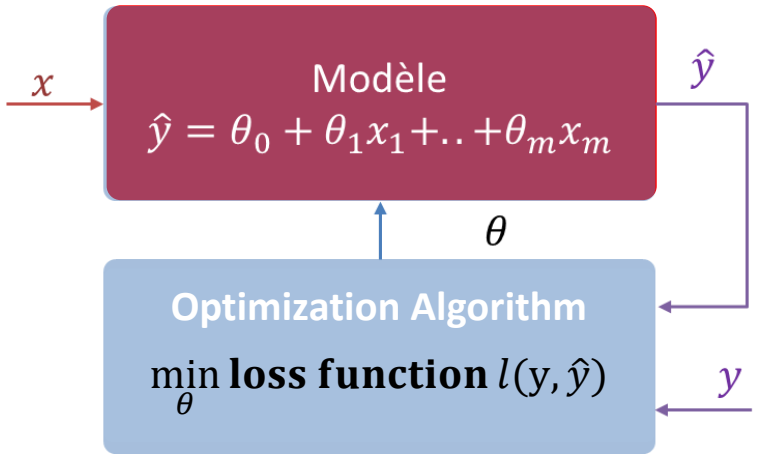
Binary Cross Entropy (Classification problem)

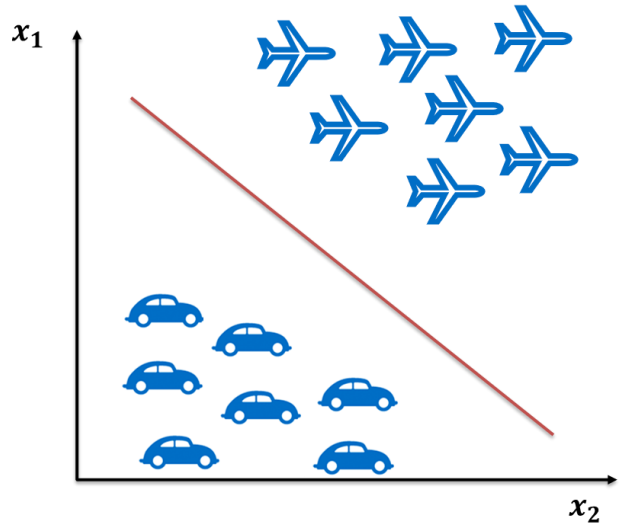
$$l(y, \hat{y}) = -\frac{1}{n\_tr} \sum_{i=1}^{n\_tr} y_i \log(\hat{y}) + (1 - y_i) \log(1 - \hat{y})$$

Mean Squared Error MSE (Regression problem)

$$l(y, \hat{y}) = \frac{1}{n\_tr} \sum_{i=1}^{n\_tr} (y - \hat{y})^2$$

COLLECT DATA | PRE-PROCESSING | **TRAINING** | PREDICTION | TESTING





samples	x_1	x_2	y
1	5.3	5.9	0
20	10.0	2.3	0
3	6.2	5.9	0
42	**6.2**	5.9	0
5	10.0	2.3	0
6	6.2	5.9	0
73	6.2	5.9	0
8	50	**5.9**	1
90	60	18	1
10	56	32	1
120	50	20	1
⋮			
n_tr	56	32	1

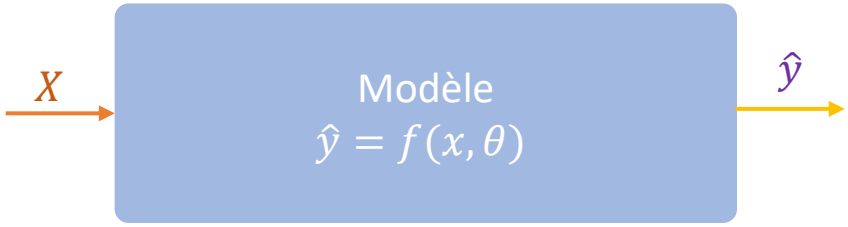
Ex: Gradient Decent Optimization Algorithm

$$\theta_t = \theta_{t-1} - \eta \frac{\partial l(y, \hat{y})}{\partial \theta_{t-1}}$$

η is learning rate $0 < \eta \leq 1$ (ex: $\eta = 0{,}01$)

COLLECT DATA | PRE-PROCESSING | TRAINING | **PREDICTION** | TESTING

x_1	x_2
5.3	5.9
10.0	2.3
6.2	5.9
6.2	5.9
10.0	2.3
6.2	5.9
50	**5.9**
60	18
56	32
50	20



$\hat{y}$
1
0
0
1
0
0
1
0
1
0

COLLECT DATA | PRE-PROCESSING | TRAINING | PREDICTION | **TESTING**

TESTING DATA (INPUT)

x_1	x_2
5.3	5.9
10.0	2.3
6.2	5.9
6.2	5.9
10.0	2.3
6.2	5.9
50	**5.9**
60	18
56	32
50	20

X → Modèle $\hat{y} = f(x, \theta)$ → $\hat{y}$

$\hat{y}$
1
0
0
1
0
0
1
0
1
0

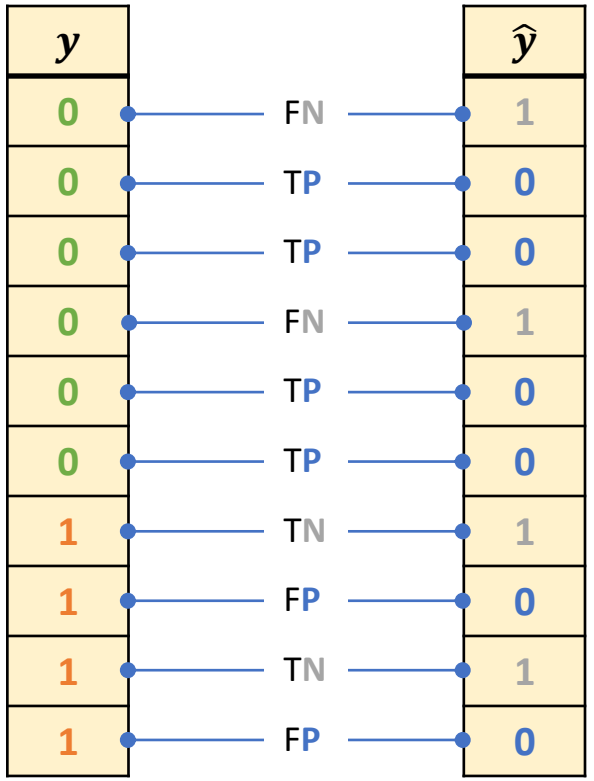
TESTING DATA (OUTPUT)

y
0
0
0
0
0
0
1
1
1
1

Evaluation Metrics of Binary classification model

Confusion Matrix

N = 10	Predicted $\hat{y}$: Positive (Car)	Predicted $\hat{y}$: Negative (No-Car)	
Actual y: Positive (Car)	TP = 4	FN = 2	6
Actual y: Negative (No-Car)	FP = 2	TN = 2	4
	6	4	

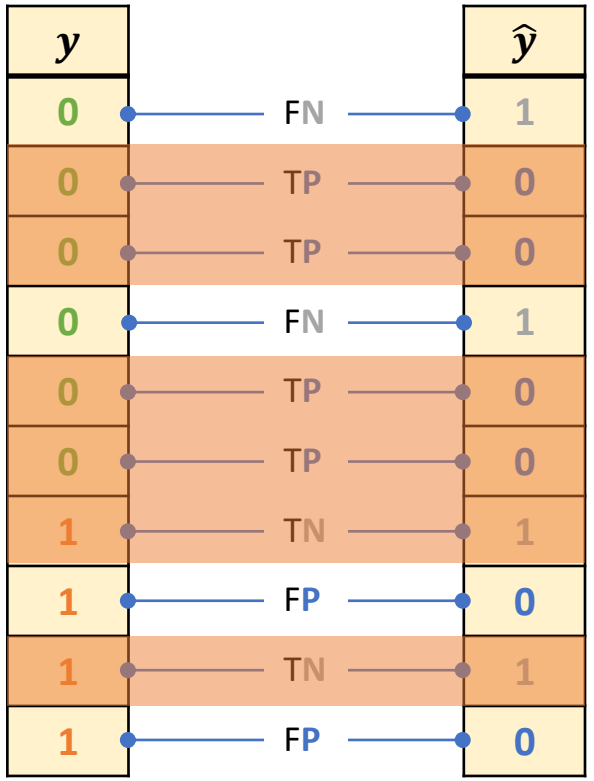


Evaluation Metrics of Binary classification model

Confusion Matrix

N = 10	Predicted $\hat{y}$: Positive (Car)	Predicted $\hat{y}$: Negative (No-Car)	
Actual y: Positive (Car)	TP = 4	FN = 2	6
Actual y: Negative (No-Car)	FP = 2	TN = 2	4
	6	4	

$$\text{Accuracy} = \frac{TN+TP}{Total\ samples}$$



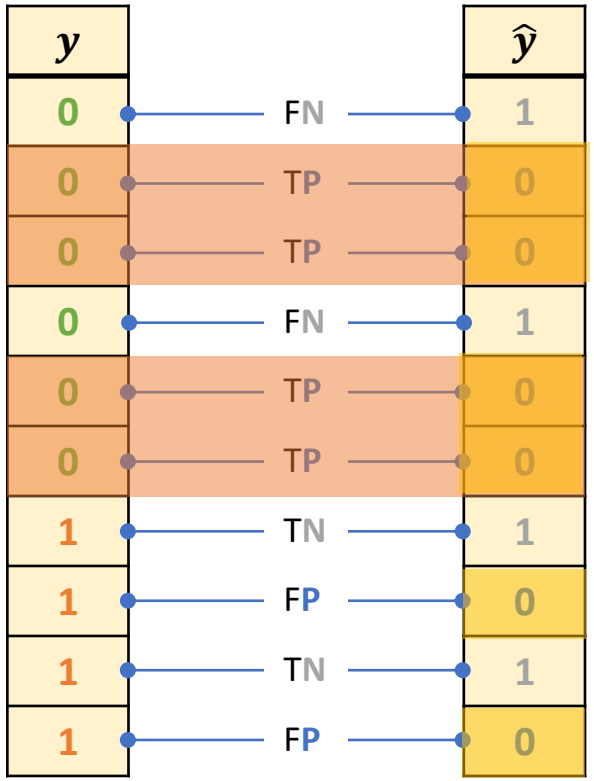
Evaluation Metrics of Binary classification model

Confusion Matrix

N = 10	Predicted $\hat{y}$: Positive (Car)	Predicted $\hat{y}$: Negative (No-Car)	
Actual y: Positive (Car)	TP = 4	FN = 2	6
Actual y: Negative (No-Car)	FP = 2	TN = 2	4
	6	4	

$$\text{Accuracy} = \frac{TN+TP}{Total\ samples}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$



Evaluation Metrics of Binary classification model

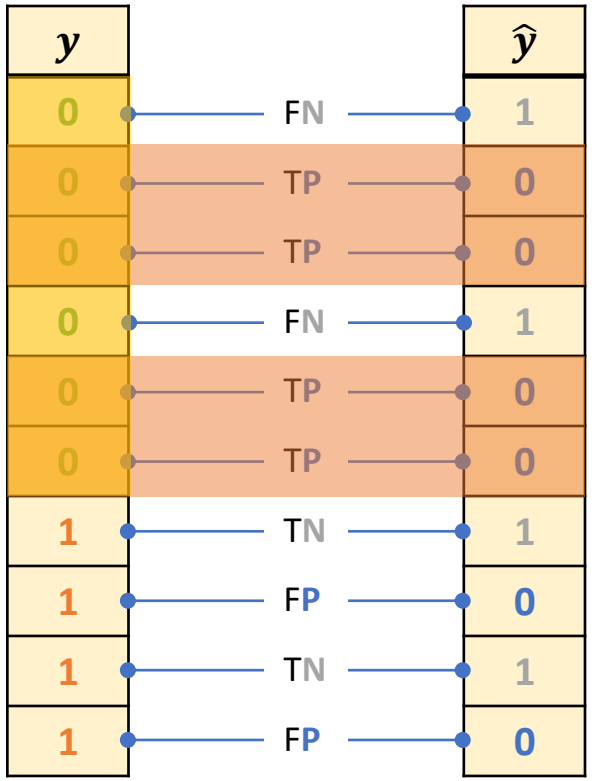
Confusion Matrix

N = 10	Predicted $\hat{y}$: Positive (Car)	Predicted $\hat{y}$: Negative (No-Car)	
Actual y: Positive (Car)	TP = 4	FN = 2	6
Actual y: Negative (No-Car)	FP = 2	TN = 2	4
	6	4	

$$\text{Accuracy} = \frac{TN+TP}{Total\ samples}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$



Evaluation Metrics of Binary classification model

Confusion Matrix

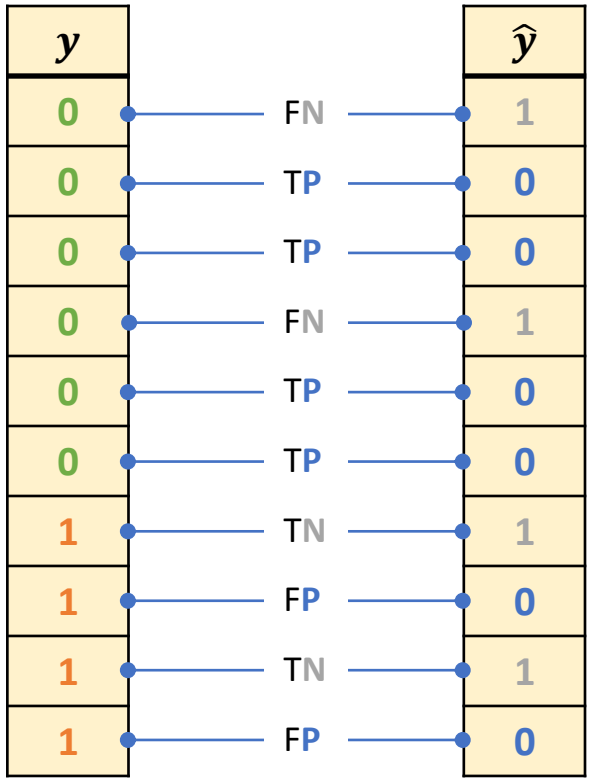
N = 10	Predicted $\hat{y}$: Positive (Car)	Predicted $\hat{y}$: Negative (No-Car)	
Actual y: Positive (Car)	TP = 4	FN = 2	6
Actual y: Negative (No-Car)	FP = 2	TN = 2	4
	6	4	

$$\text{Accuracy} = \frac{TN+TP}{Total\ samples}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{F1 score} = \frac{2 * Recall * Precision}{Recall + Precision}$$



Evaluation Metrics of Multi classification model

Confusion Matrix

Actual y \ Predicted $\hat{y}$	Car	Airplane	Train	Bicycle
Car	9	1	0	0
Airplane	1	15	3	1
Train	5	0	24	1
Bicycle	0	4	1	15

N = 80

$$\text{Accuracy} = \frac{Total\ number\ of\ correct\ predictions}{Total\ samples}$$

$$\text{Accuracy} = \frac{9+15+24+15}{80} = 0{,}78$$

COLLECT DATA | PRE-PROCESSING | TRAINING | PREDICTION | **TESTING**

Evaluation Metrics of Multi classification model

Confusion Matrix of model 1

Actual y \ Predicted $\hat{y}$	Car	Airplane	Train	Bicycle
Car	10	0	0	0
Airplane	0	5	3	2
Train	0	1	8	1
Bicycle	0	1	0	9

$$\text{Accuracy} = \frac{10+5+8+9}{40} = 0,8$$

Confusion Matrix of model 2

Actual y \ Predicted $\hat{y}$	Car	Airplane	Train	Bicycle
Car	8	2	0	0
Airplane	1	7	0	2
Train	0	0	9	1
Bicycle	2	3	0	5

$$\text{Accuracy} = \frac{8+7+9+5}{40} = 0,72$$

COLLECT DATA | PRE-PROCESSING | TRAINING | PREDICTION | **TESTING**

Evaluation Metrics of Multi classification model

Confusion Matrix of model 1

Actual y \ Predicted $\hat{y}$	Car	Airplane	Train	Bicycle
Car	10	0	0	0
Airplane	0	5	3	2
Train	0	1	8	1
Bicycle	0	1	0	9

N = 40

Accuracy = $\frac{10+5+8+9}{40} = 0,8$

Confusion Matrix of model 2

Actual y \ Predicted $\hat{y}$	Car	Airplane	Train	Bicycle
Car	8	2	0	0
Airplane	1	7	0	2
Train	0	0	9	1
Bicycle	2	3	0	5

N = 40

Accuracy = $\frac{8+7+9+5}{40} = 0,72$

Accuracy works well on balanced data

COLLECT DATA | PRE-PROCESSING | TRAINING | PREDICTION | **TESTING**

Evaluation Metrics of Multi classification

Accuracy on imbalanced data misleads performance

Confusion Matrix of model 1

Actual y \ Predicted $\hat{y}$	Car	Airplane	Train	Bicycle	Recall
Car	100	80	10	10	
Airplane	0	9	0	1	
Train	0	1	8	1	
Bicycle	0	1	0	9	
Precision					

N = 230

Accuracy = $\frac{100+9+8+9}{230} = 0,547$

Confusion Matrix of model 2

Actual y \ Predicted $\hat{y}$	Car	Airplane	Train	Bicycle
Car	198	2	0	0
Airplane	7	1	0	2
Train	0	8	1	1
Bicycle	2	3	4	1

N = 230

Accuracy = $\frac{198+1+1+1}{230} = 0,87$

Evaluation Metrics of Multi classification

Accuracy on imbalanced data misleads performance

Confusion Matrix of model 1

Predicted $\hat{y}$

Actual y (N = 230)	Car	Airplane	Train	Bicycle	Recall
Car	100	80	10	10	$\frac{100}{200}$
Airplane	0	9	0	1	$\frac{9}{10}$
Train	0	1	8	1	$\frac{8}{10}$
Bicycle	0	1	0	9	$\frac{9}{10}$
Precision	$\frac{100}{100}$	$\frac{9}{91}$	$\frac{8}{18}$	$\frac{9}{21}$	

Accuracy $= \mathbf{0,547}$

Avrage Precision = 0,492

Avrage Recall = 0,775

F1 score = $\frac{2 * Recall * Precision}{Recall + Precision}$

F1 score = $\frac{2 * 0,775 * 0,492}{0,775 + 0,492} = \mathbf{0,601}$

Confusion Matrix of model 2

Predicted $\hat{y}$

Actual y (N = 230)	Car	Airplane	Train	Bicycle	Recall
Car	198	2	0	0	$\frac{198}{200}$
Airplane	7	1	0	2	$\frac{1}{10}$
Train	0	8	1	1	$\frac{1}{10}$
Bicycle	2	3	4	1	$\frac{1}{10}$
Precision	$\frac{198}{207}$	$\frac{1}{14}$	$\frac{1}{5}$	$\frac{1}{4}$	

Accuracy $= \mathbf{0,87}$

Avrage Precision = 0,369

Avrage Recall = 0,323

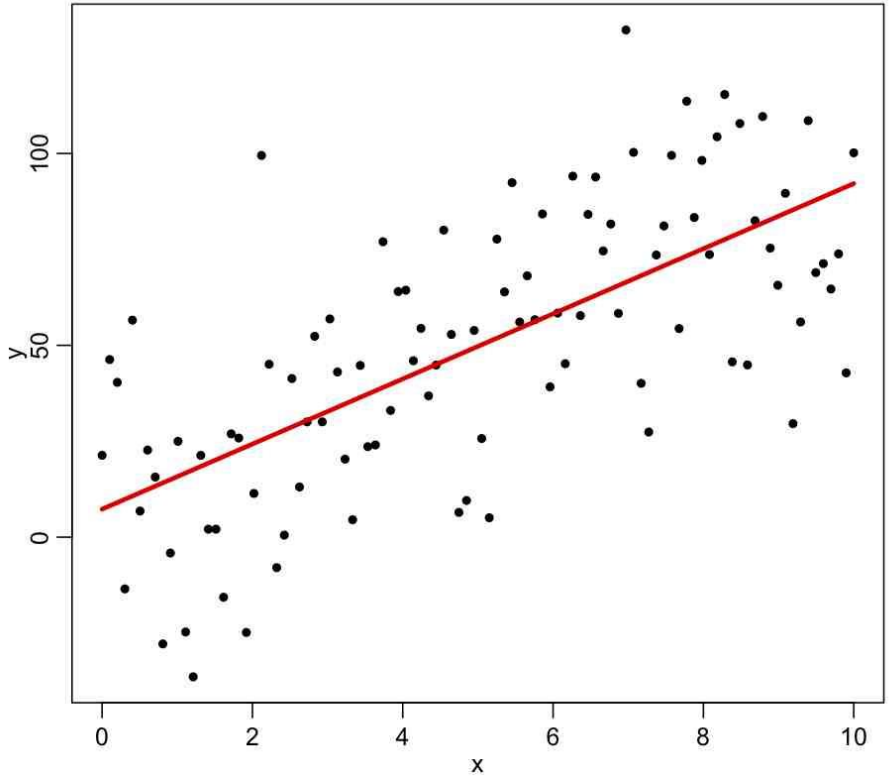
F1 score = $\frac{2 * Recall * Precision}{Recall + Precision}$

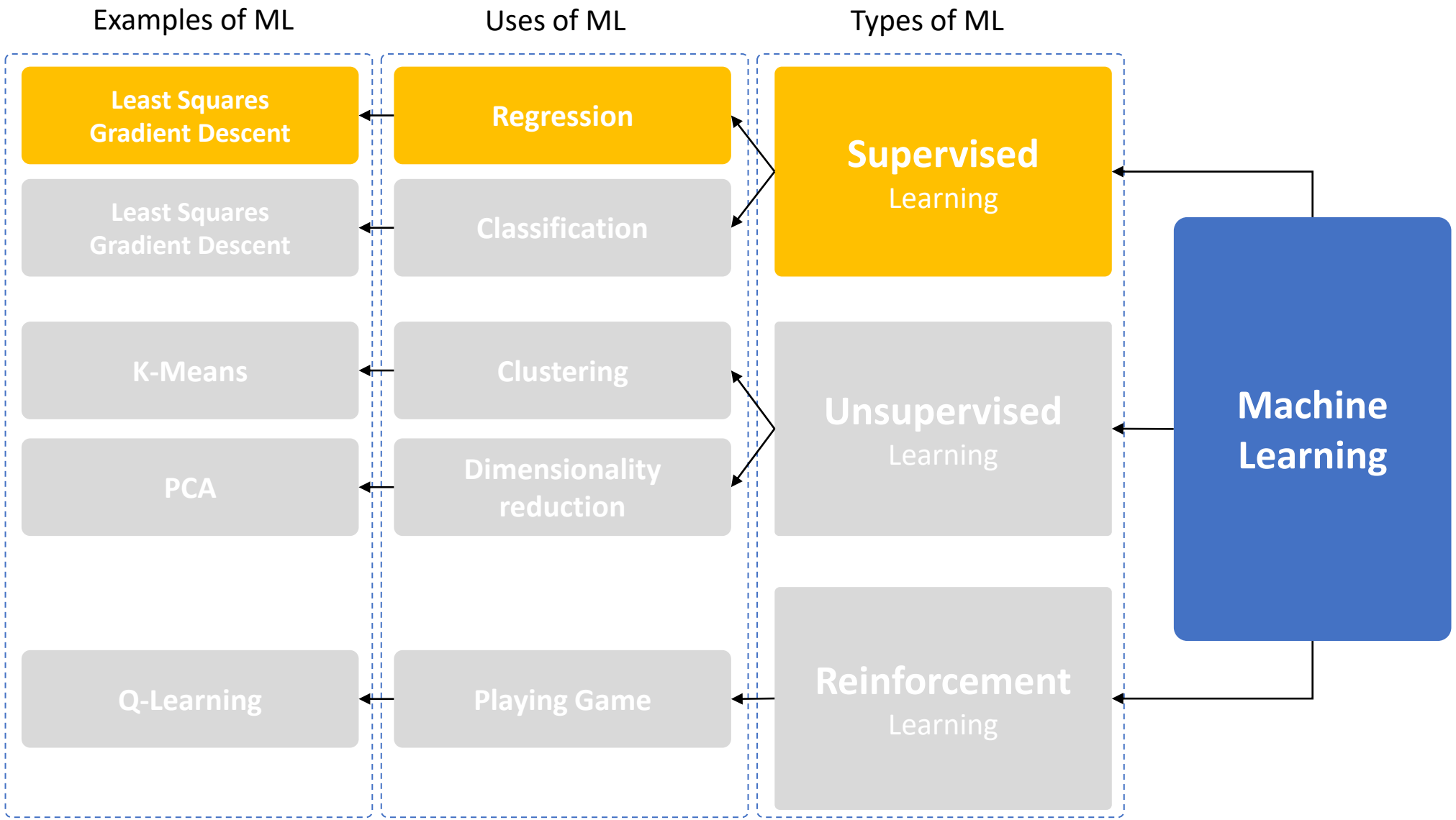
F1 score = $\frac{2 * 0,323 * 0,369}{0,323 + 0,369} = \mathbf{0,344}$

Evaluation Metrics of regression model

R2 score: (R-squared) The best possible score is 1.0, but the score can be negative as well.

$$R2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \quad with \quad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$




Examples of ML
Uses of ML
Types of ML
Least Squares Gradient Descent
Least Squares Gradient Descent
K-Means
PCA
Q-Learning
Regression
Classification
Clustering
Dimensionality reduction
Playing Game
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Machine Learning

linear regression model

Let's consider the following linear regression model:

$$y_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_n x_{in} + e_i \quad \text{with } e_i \text{ is the estimation error}$$

for N samples, the above equation can be written in the following matrix format:

$$Y = X\Theta + \mathrm{E}$$

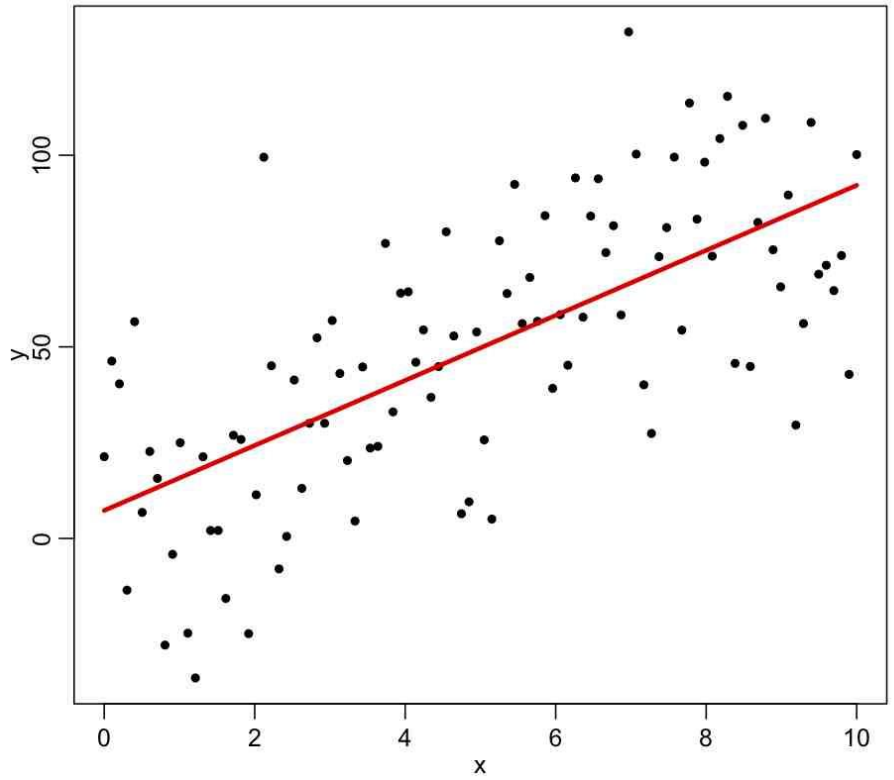
With

$$Y = [y_1, y_2, \ldots, y_N]^T$$

$$\mathrm{E} = [e_1, e_2, \ldots, e_N]^T$$

$$\Theta = [\theta_0, \theta_1, \ldots, \theta_n]^T$$

$$X = \begin{bmatrix} 1 & x_{11} & \ldots & x_{1n} \\ 1 & x_{21} & \ldots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & \ldots & x_{Nn} \end{bmatrix}$$



Linear Least Squares for linear regression problem

From this model $\boldsymbol{Y} = \boldsymbol{X\Theta} + \boldsymbol{E}$, we can write the following error:

$$\mathrm{E} = Y - X\Theta$$

Now to find the optimal parameters Θ, we need to minimize the following loss function $l(\mathrm{Y}, X\Theta)$:

$$l(\mathrm{Y}, X\Theta) = \frac{1}{2N}\,\mathrm{E}^T\mathrm{E} = \frac{1}{2N}\,(Y - X\Theta)^T(Y - X\Theta)$$
$$= \frac{1}{2N}\,(Y^TY - Y^TX\Theta - \Theta^TX^TY + \Theta^TX^TX\Theta)$$

The derivative of the loss is

$$\frac{\partial l(\mathrm{Y}, X\Theta)}{\partial \Theta} = \frac{1}{2N}\,(-2X^TY + 2X^TX\Theta)$$

Setting the derivative of the loss to zero and solving for Θ we get:

$$\frac{\partial l(\mathrm{Y}, X\Theta)}{\partial \Theta} = 0 \quad \Rightarrow \quad -2X^TY + 2X^TX\Theta = 0 \quad \Rightarrow \quad X^TX\Theta = X^TY \quad \Rightarrow \quad \Theta = (X^TX)^{-1}X^TY$$

Click here pdf-file to see the matrix derivation shapes used in this demo

Gradient Descent for linear regression problem

Here we solve the linear regression model $\boldsymbol{Y = X\Theta + E}$ using the gradient descent formula which is given by:

$$\Theta = \Theta \ - \eta \frac{\boldsymbol{\partial l(Y, X\Theta)}}{\boldsymbol{\partial \Theta}} = \Theta - \eta \boldsymbol{\Delta \Theta}$$

Now to find the gradient $\boldsymbol{\Delta\Theta}$, we need to derivative the flowing loss function

$$l(\mathrm{Y}, X\Theta) = \frac{1}{2N} E^T E = \frac{1}{2N} (Y - X\Theta)^T (Y - X\Theta)$$

The derivative of the loss is

$$\Delta\theta = \frac{\partial l(\mathrm{Y}, X\Theta)}{\partial \theta} = \left(\frac{\partial E}{\partial \theta}\right)\left(\frac{\partial l(\mathrm{Y}, X\Theta)}{\partial E}\right)$$

$$= \left(\frac{\partial Y - X\Theta}{\partial \theta}\right)\left(\frac{1}{2N}\frac{\partial E^T E}{\partial E}\right)$$

$$= -\frac{1}{N} X^T E$$

Click here pdf-file to see the matrix derivation shapes used in this demo

Gradient Descent for linear regression problem

For this model $\boldsymbol{Y = X\Theta + E}$, we have the following gradient update:

$$\Delta\boldsymbol{\Theta} = -\frac{\mathbf{1}}{\boldsymbol{N}}\boldsymbol{X^T E}$$

If we want write the model like this $\boldsymbol{Y = X\Theta + I\theta_0 + E}$ (separate the intercept parameter $\boldsymbol{\theta_0}$), we will have the following two gradient updates:

$$\Delta\boldsymbol{\Theta} = \left(\frac{\partial Y - X\Theta - I\theta_0}{\partial\Theta}\right)\left(\frac{1}{2N}\frac{\partial E^T E}{\partial E}\right) = -\frac{\mathbf{1}}{\boldsymbol{N}}\boldsymbol{X^T E}$$

$$\Delta\boldsymbol{\theta_0} = \left(\frac{\partial Y - X\Theta - I\theta_0}{\partial\theta_0}\right)\left(\frac{1}{2N}\frac{\partial E^T E}{\partial E}\right) = -\frac{\mathbf{1}}{\boldsymbol{N}}\boldsymbol{I^T E} = -\frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\boldsymbol{e_i}$$

Where

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots \\ x_{N1} & \cdots & x_{Nn} \end{bmatrix} \; and \; I = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Click here pdf-file to see the matrix derivation shapes used in this demo

Gradient Descent for linear regression problem

For this model $\boldsymbol{Y = X\Theta + E}$, we have the following gradient update:

$$\Delta\boldsymbol{\Theta} = -\frac{\mathbf{1}}{\boldsymbol{N}}\boldsymbol{X^T E}$$

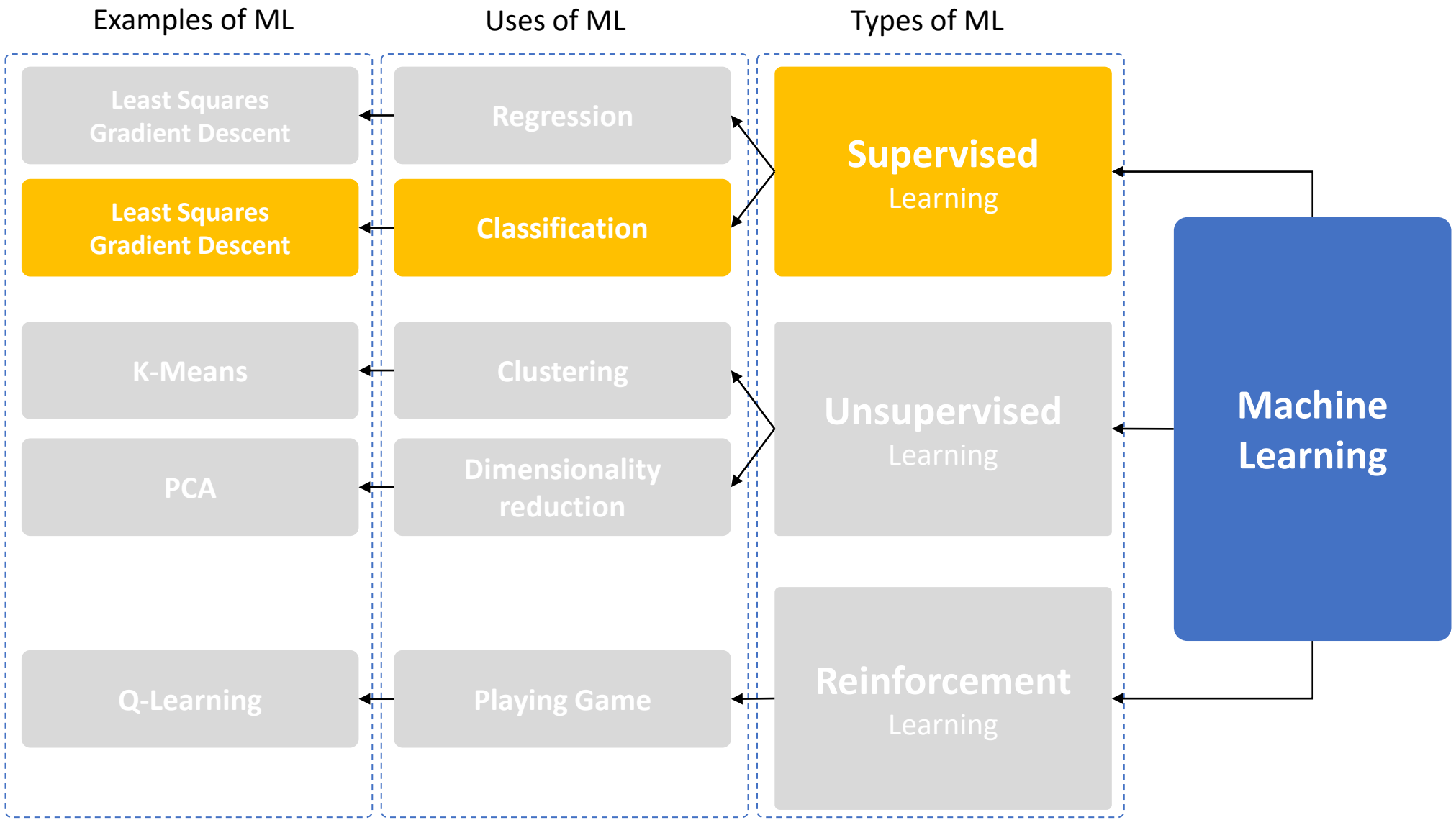
If we want write the model like this $\boldsymbol{Y = X\Theta + I\theta_0 + E}$ (separate the intercept parameter $\boldsymbol{\theta_0}$), we will have the following two gradient updates:

$$\Delta\boldsymbol{\Theta} = \left(\frac{\partial Y - X\Theta - I\theta_0}{\partial\Theta}\right)\left(\frac{1}{2N}\frac{\partial E^T E}{\partial E}\right) = -\frac{\mathbf{1}}{\boldsymbol{N}}\boldsymbol{X^T E}$$

$$\Delta\boldsymbol{\theta_0} = \left(\frac{\partial Y - X\Theta - I\theta_0}{\partial\theta_0}\right)\left(\frac{1}{2N}\frac{\partial E^T E}{\partial E}\right) = -\frac{\mathbf{1}}{\boldsymbol{N}}\boldsymbol{I^T E} = -\frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\boldsymbol{e_i}$$

In the implementation of the Gradient Descent algorithm, we will use this presentation to avoid concatenating the vector of 1 to the input matrix X.

Click here pdf-file to see the matrix derivation shapes used in this demo

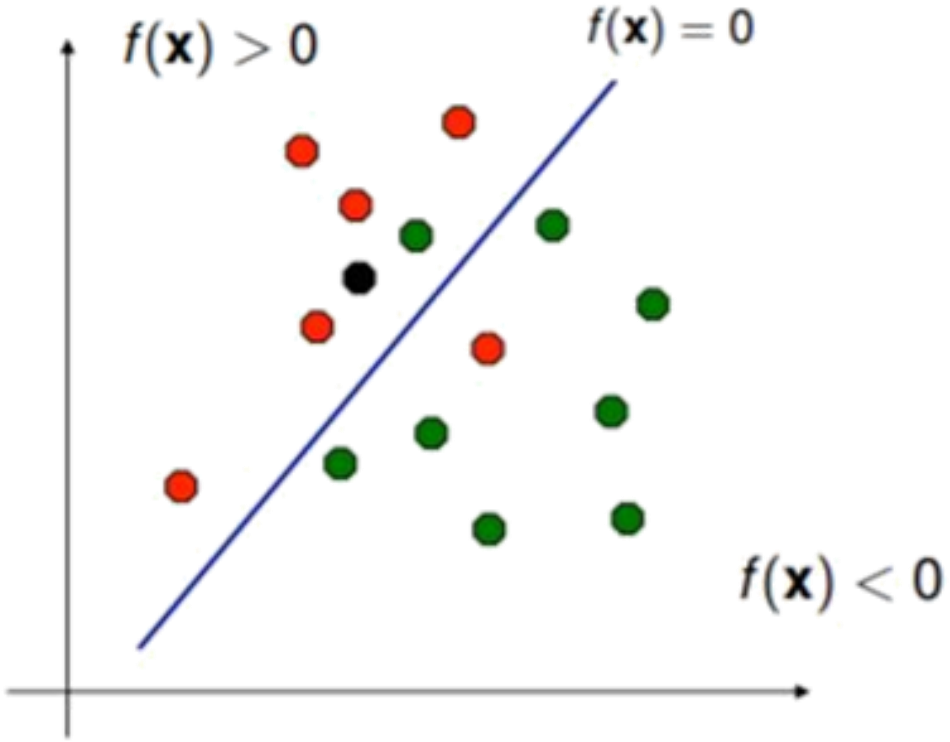

Examples of ML
Uses of ML
Types of ML
Least Squares Gradient Descent
Regression
Supervised Learning
Least Squares Gradient Descent
Classification
Machine Learning
K-Means
Clustering
Unsupervised Learning
PCA
Dimensionality reduction
Q-Learning
Playing Game
Reinforcement Learning

linear classification model for binary classification where labels y =±1

Let's consider the following linear classification model for each sample :

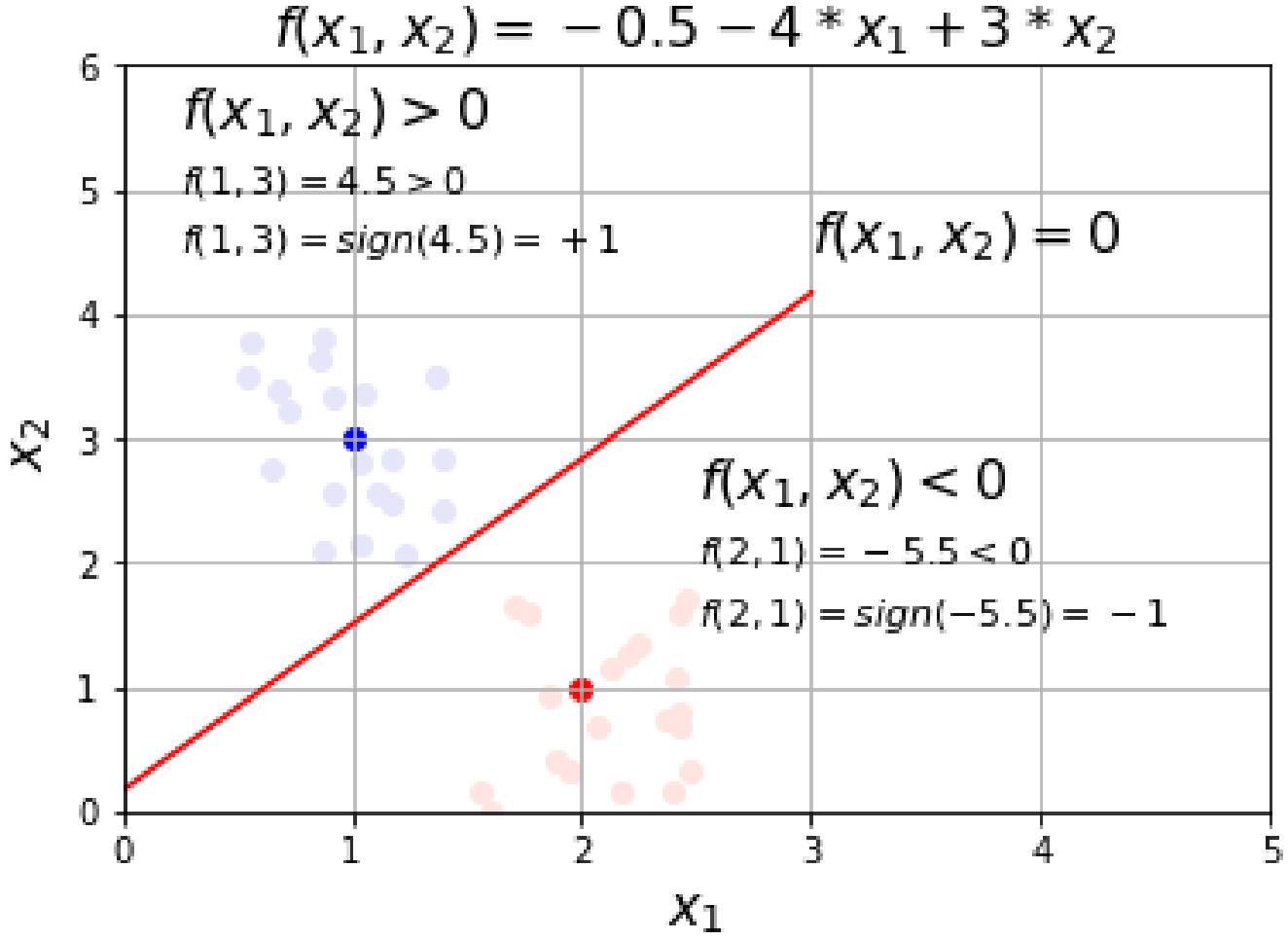
$$f_i(x) = \hat{y}_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_n x_{in} \quad | \, i = 1, \ldots, N \text{ samples}$$

The equation above present linear decision boundary (**decision function**)



linear classification model for binary classification where labels y =±1

Example for 2 features:



linear classification model for binary classification where labels y $=\pm 1$

Now the question is how to find the parameters Θ of the linear model $\widehat{y_i} = \mathrm{X_i}\Theta + \theta_0$?

1- we need first to define the error e_i between true labels y_i and prediction value $\widehat{y_\mathrm{i}}$

2- we need to define the loss function

3- select an algorithm to find the optimal parameters to minimize the loss function

linear classification model for binary classification where labels y =±1

Now the question is how to find the parameters Θ of the linear model $\hat{y}_i = X_i\Theta + \theta_0$?

1- we need first to define the error e_i between true labels y_i and prediction value $\hat{y}_i$

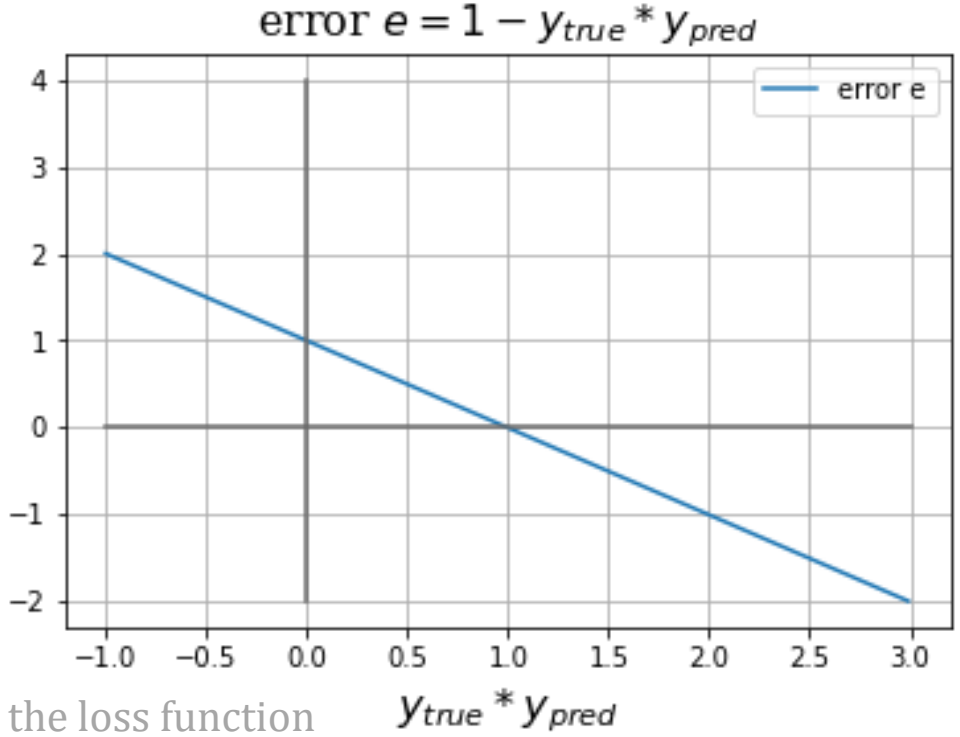
$e_i = 1 - y_i\hat{y}_i$

$e_i = 0$ *si et seulement si* $y_i\hat{y}_i = 1$

$y_i\hat{y}_i > 0$: **prediction correct**

$y_i\hat{y}_i < 0$: **prediction no correct**

y_i \ $\hat{y}_i$	+	-
+	+	-
-	-	+



2- we need to define the loss function

3- select an algorithm to find the optimal parameters to minimize the loss function

linear classification model for binary classification where labels y =±1

Now the question is how to find the parameters Θ of the linear model $\hat{y}_i = X_i\Theta + \theta_0$?

1- we need first to define the error e_i between true labels y_i and prediction value $\hat{y}_i$

$$e_i = 1 - y_i\hat{y}_i$$

2- we need to define the loss function

We can use MSE loss function:

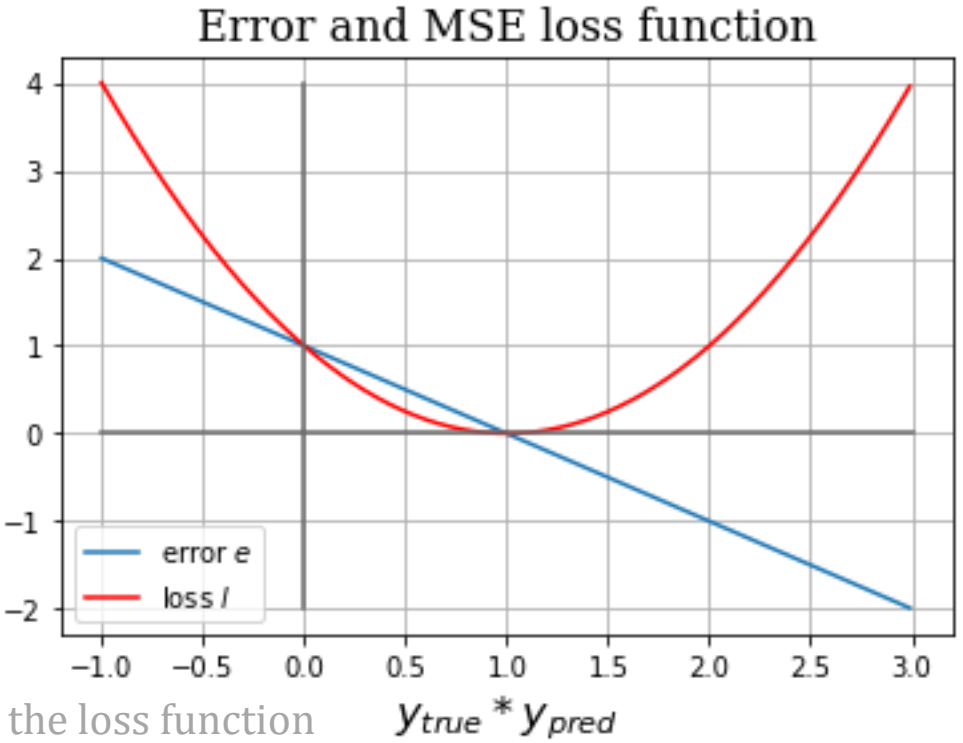
$$l(Y, \hat{Y}) = \frac{1}{2N}\sum_{i=1}^{N} e_i^2$$

Since label $y_i = \pm 1$, we can multiply e_i^2 by y_i^2

$$l(Y, \hat{Y}) = \frac{1}{2N}\sum_{i=1}^{N} y_i^2 e_i^2 = \frac{1}{2N}\sum_{i=1}^{N} y_i^2(1 - y_i\hat{y}_i)^2$$

$$= \frac{1}{2N}\sum_{i=1}^{N}(y_i - y_i^2\hat{y}_i)^2$$

$$= \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad \Leftrightarrow \text{regression loss}$$



3- select an algorithm to find the optimal parameters to minimize the loss function

linear classification model for binary classification where labels y $=\pm 1$

Now the question is how to find the parameters Θ of the linear model $\widehat{y_i} = \mathrm{X_i}\Theta + \theta_0$?

1- we need first to define the error e_i between true labels y_i and prediction value $\widehat{y_i}$

$$e_i = 1 - y_i\widehat{y_i}$$

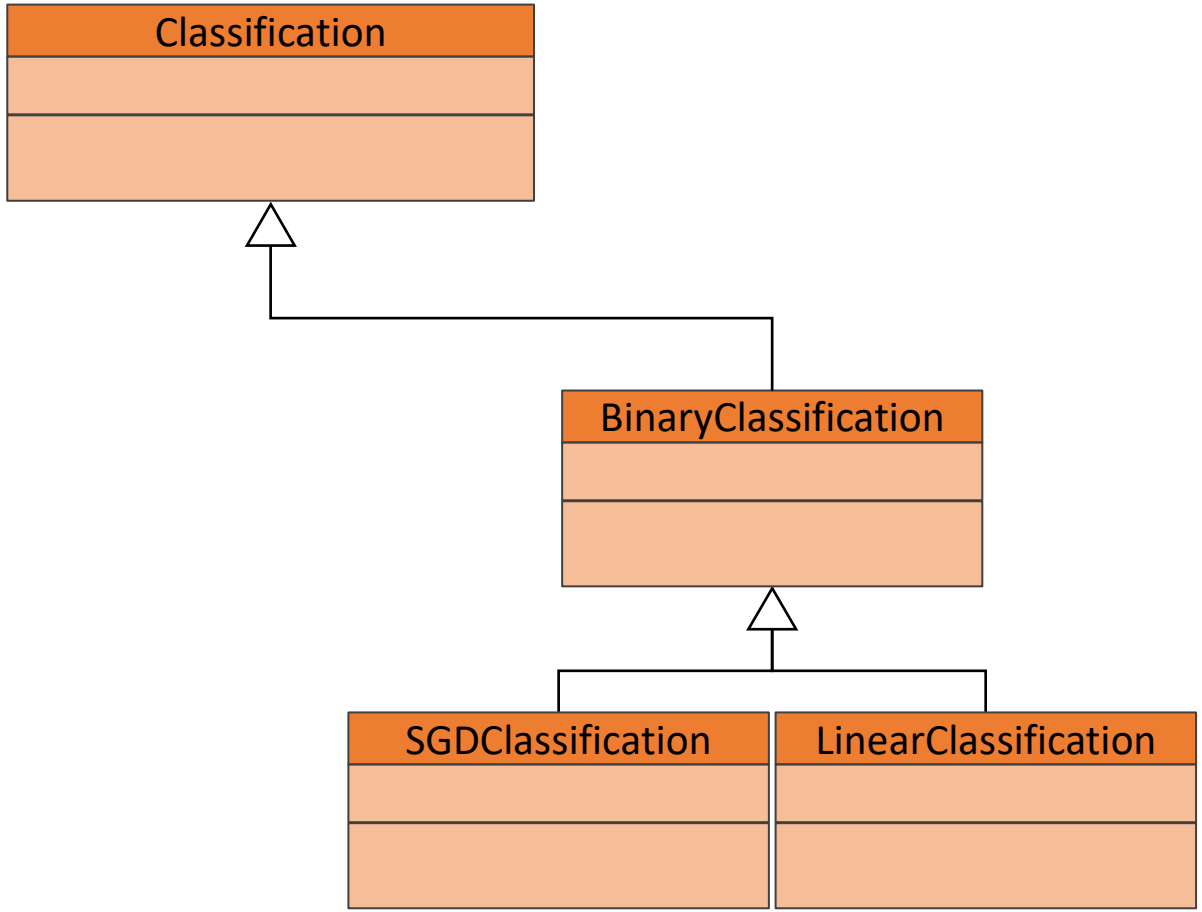
2- we need to define the loss function

$$l(Y, \widehat{Y}) = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \widehat{y_i})^2 \quad \Leftrightarrow \quad \text{regression loss}$$

3- select an algorithm to find the optimal parameters to minimize the loss function

As classification loss is the same of regression loss, we can use linear regression algorithms to solve the linear classification problem where the label y= $\pm$.

linear classification model for binary classification where labels $y = \pm 1$

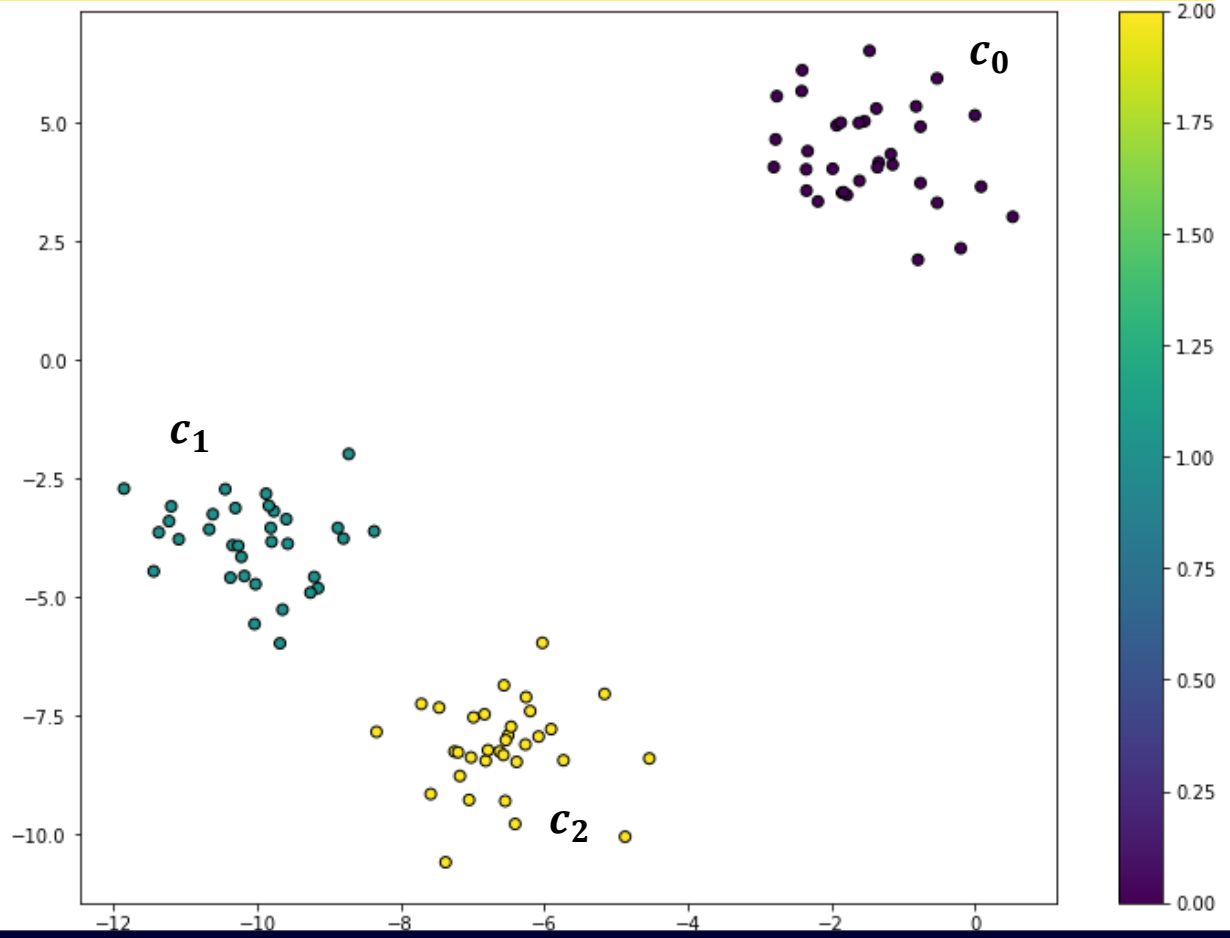


check this link to know more about python inheritance

linear classification model for multiclass classification where labels y $= \pm 1$

Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_0, c_1, c_2\}$



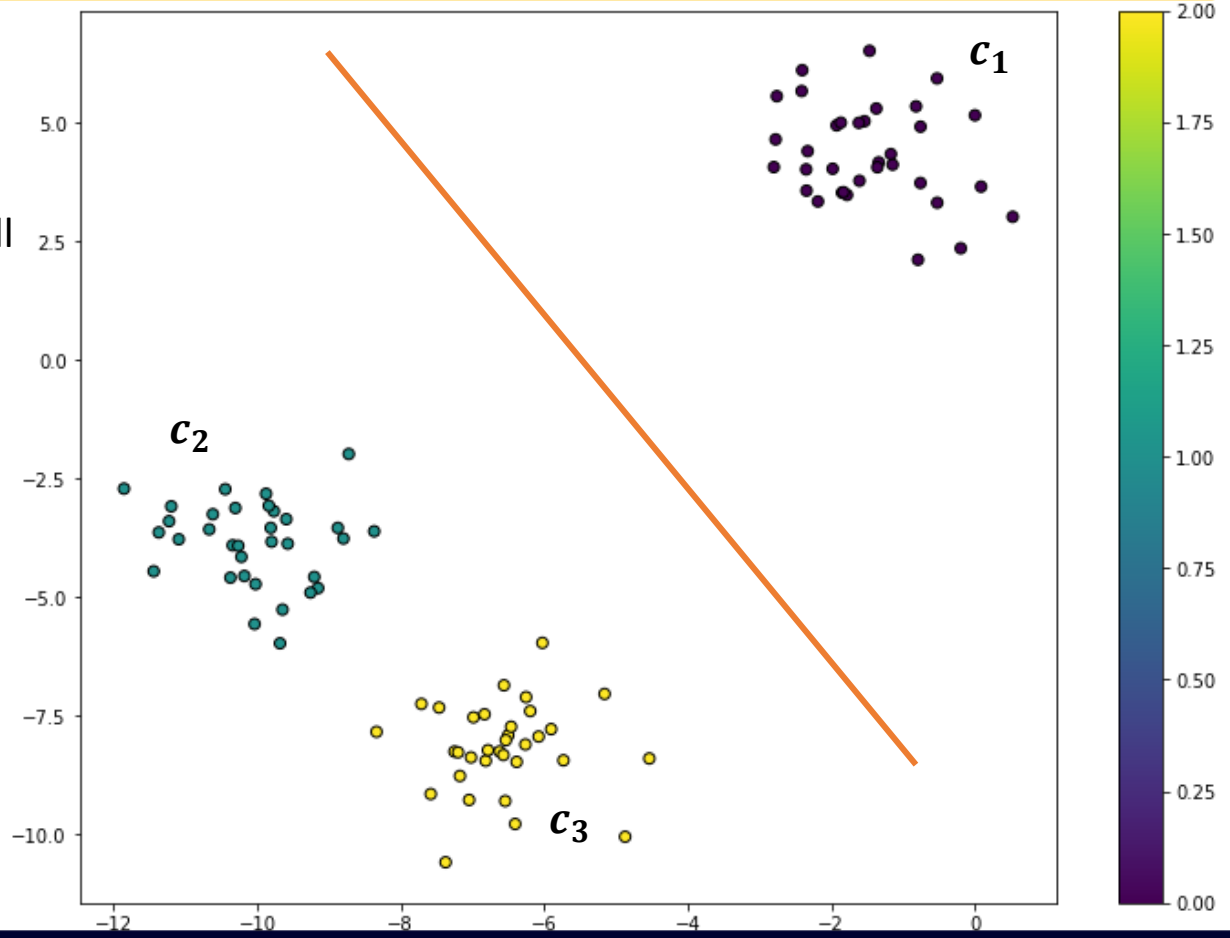
linear classification model for multiclass classification where labels y $=\pm 1$

Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$



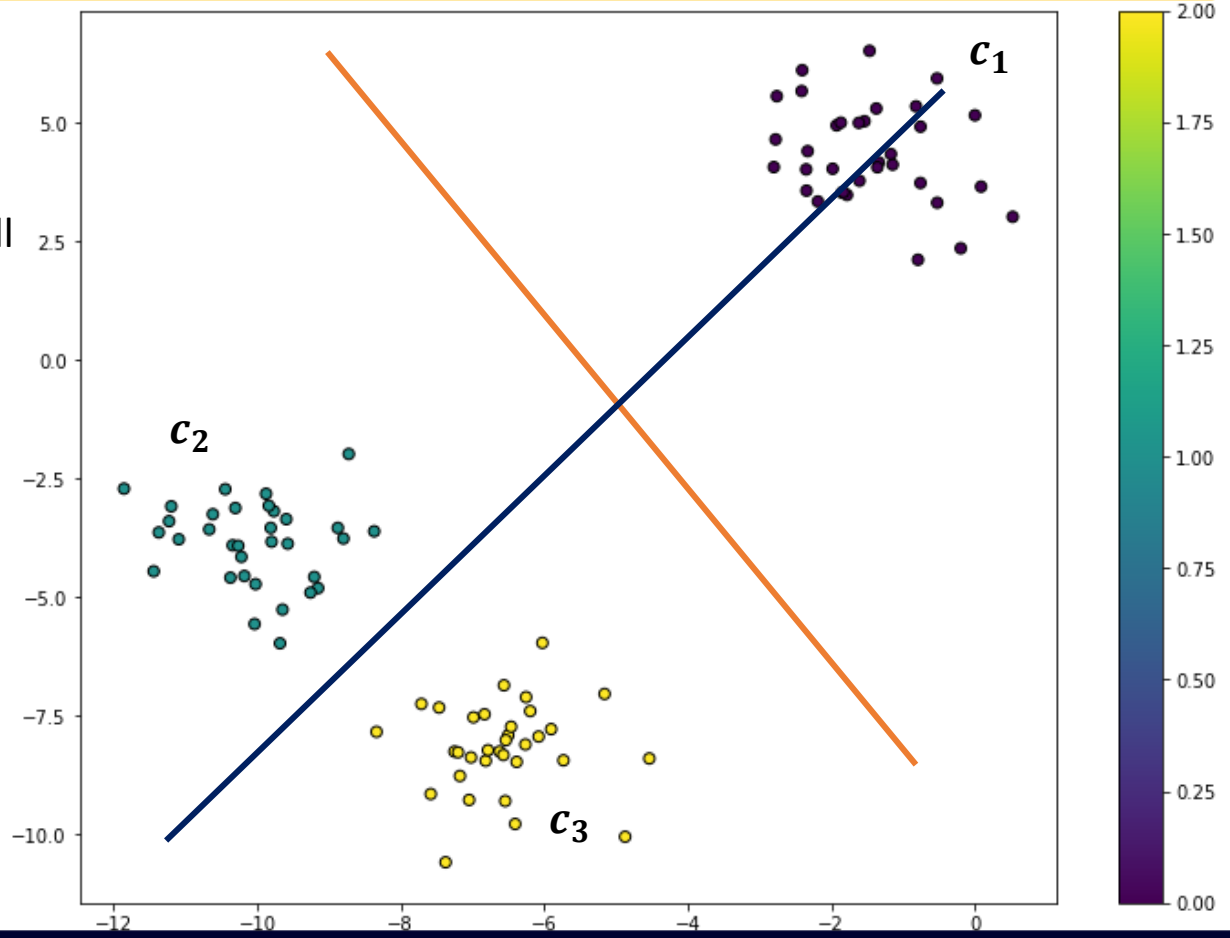
linear classification model for multiclass classification where labels y $=\pm 1$

Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$



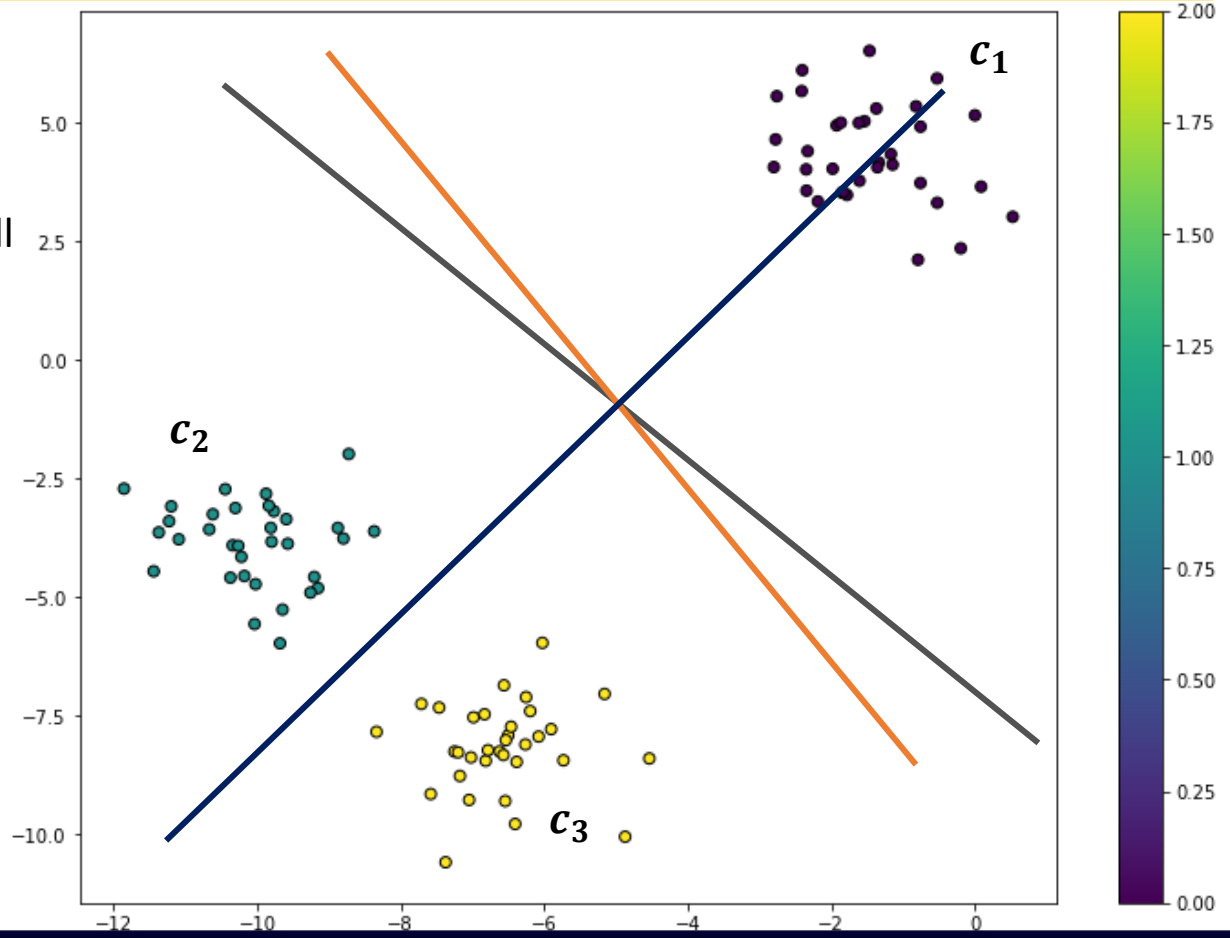
linear classification model for multiclass classification where labels y $=\pm 1$

Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$



linear classification model for multiclass classification where labels y $=\pm 1$

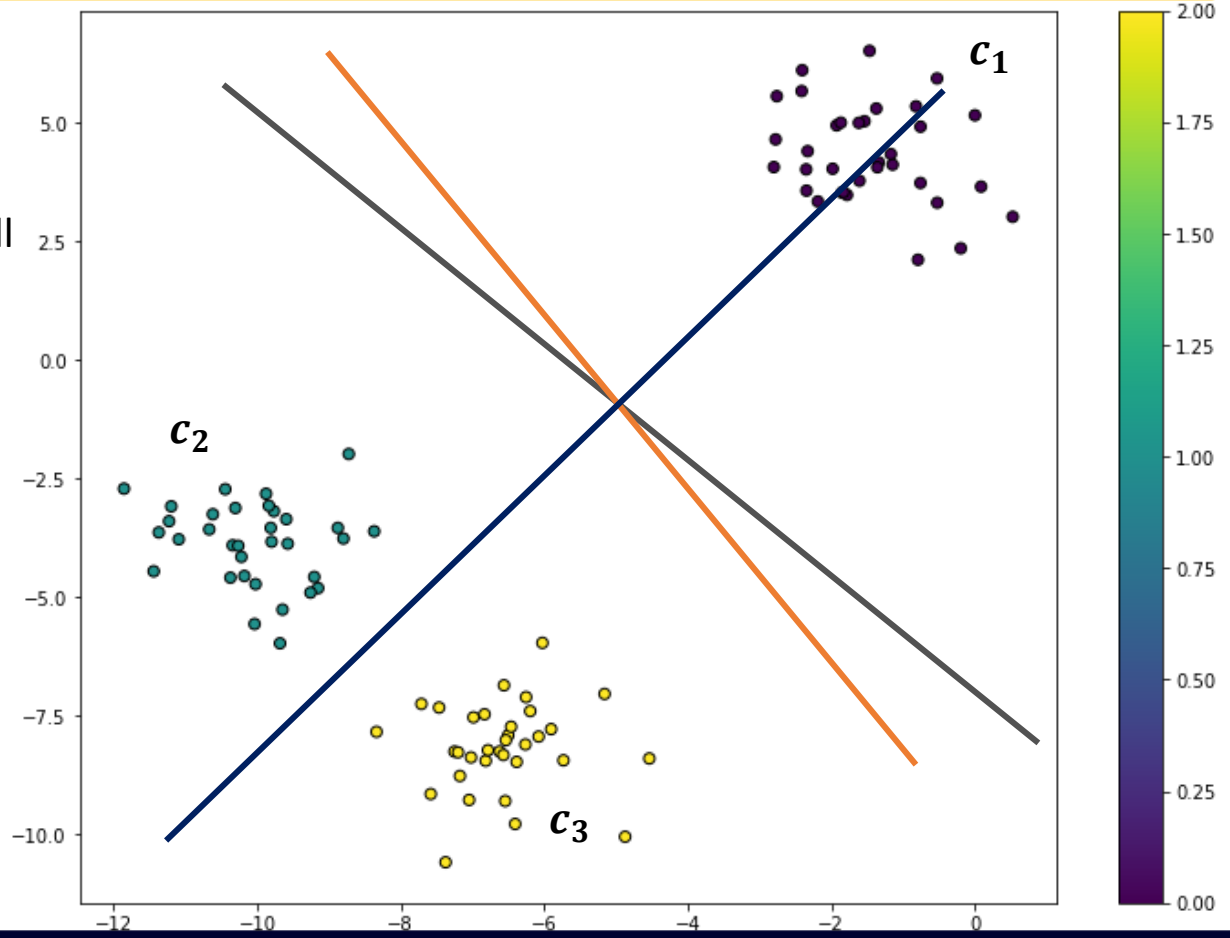
Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$

$p_i \in (y_{12}, y_{23}, y_{31})$



linear classification model for multiclass classification where labels y $=\pm 1$

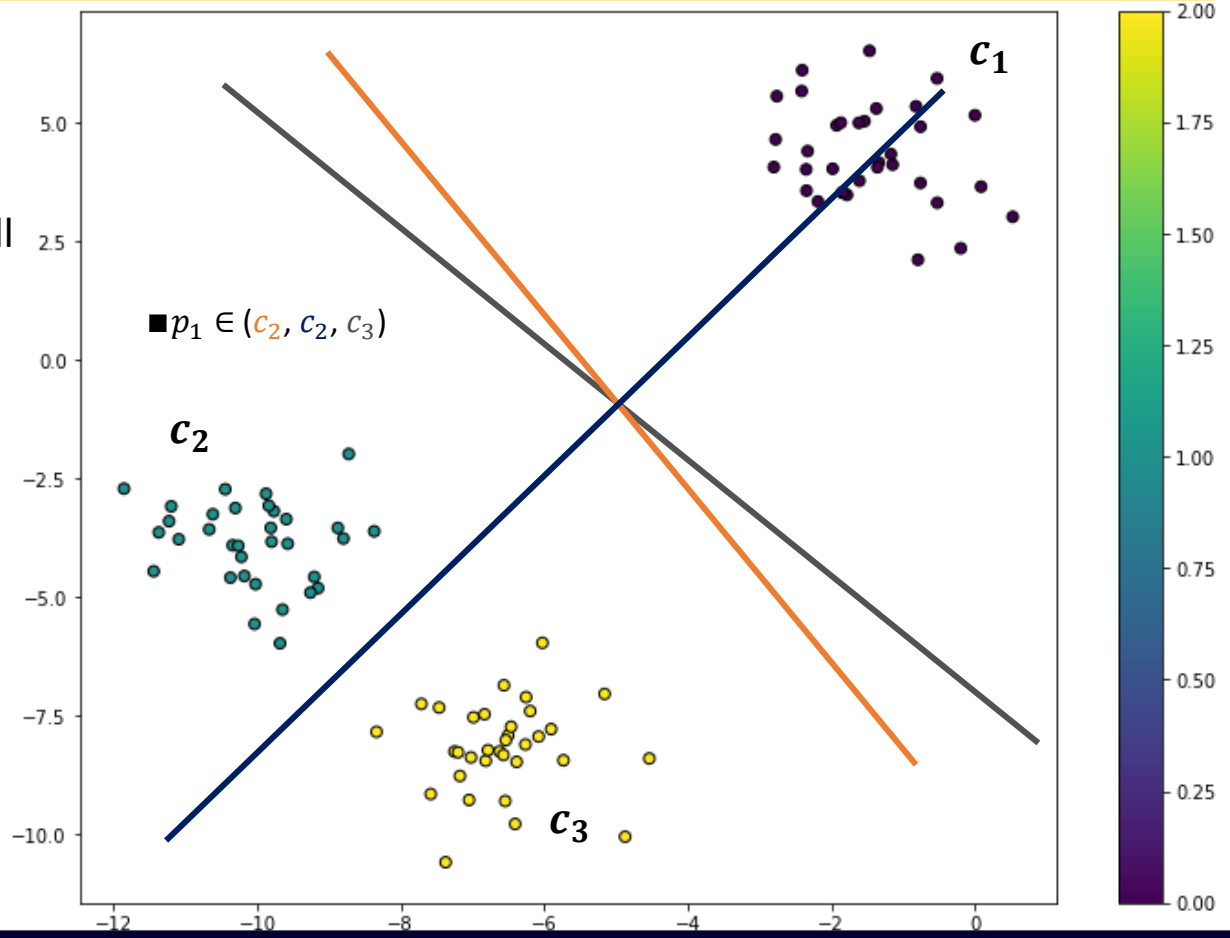
Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$

$p_i \in (y_{12}, y_{23}, y_{31})$



linear classification model for multiclass classification where labels y $=\pm 1$

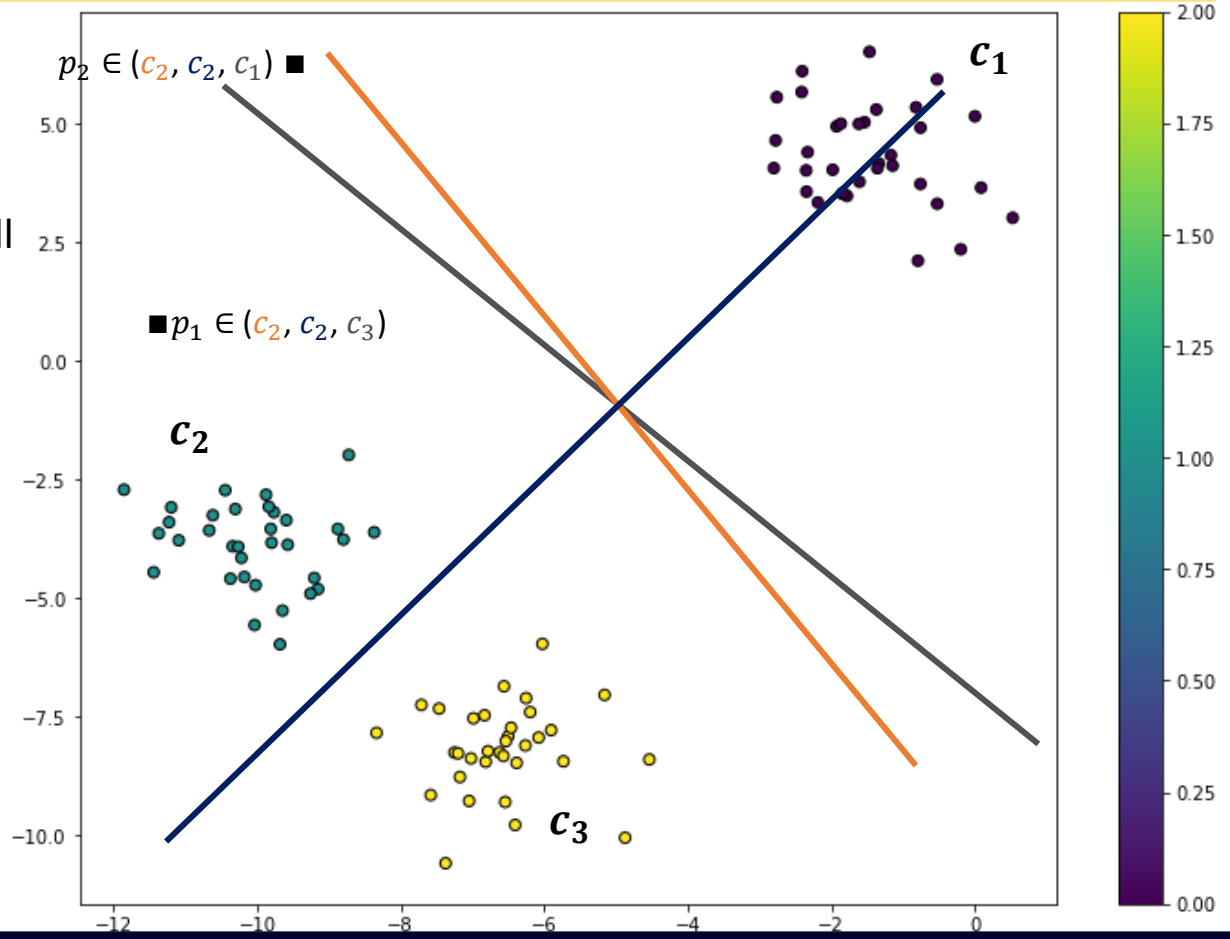
Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$

$p_i \in (y_{12}, y_{23}, y_{31})$



linear classification model for multiclass classification where labels y $=\pm 1$

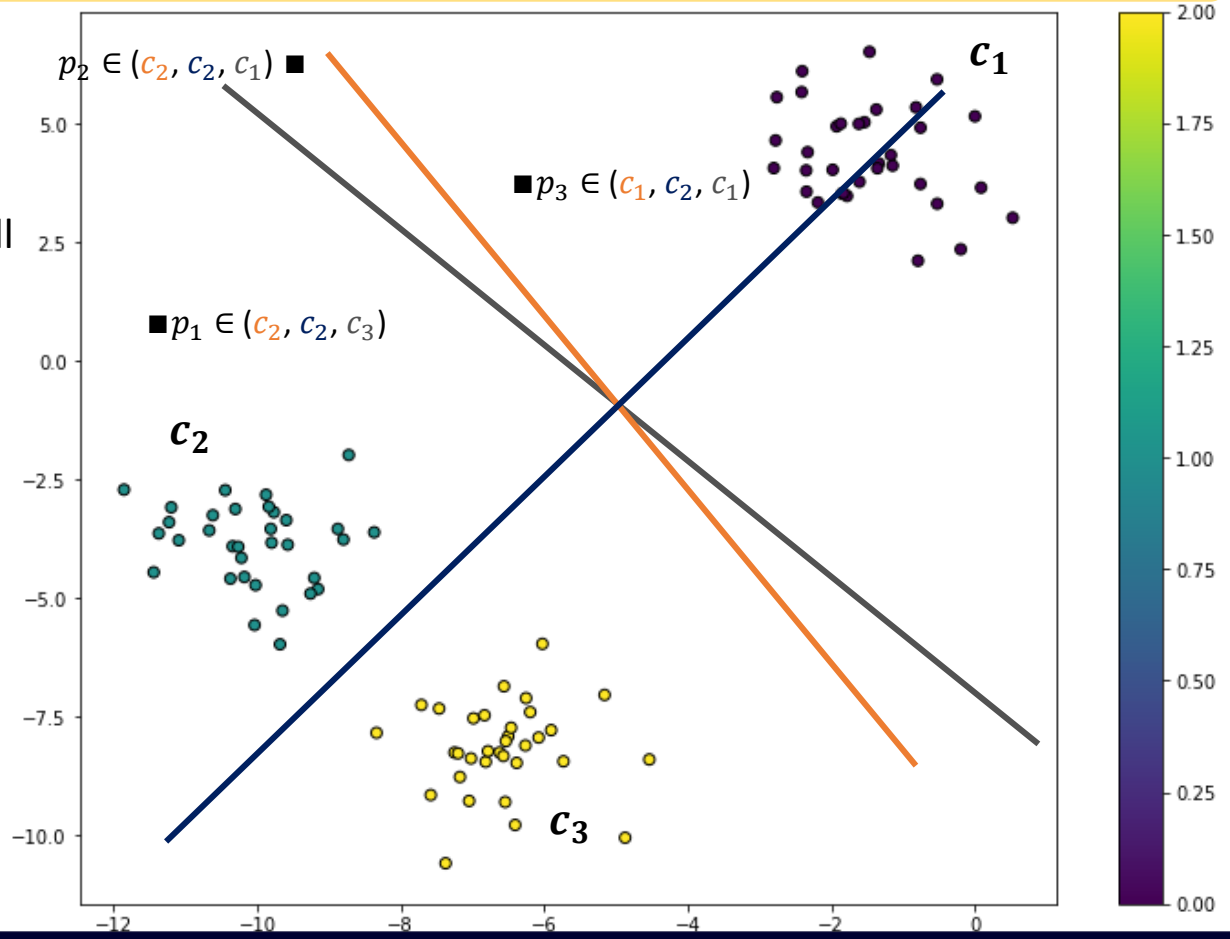
Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$

$p_i \in (y_{12}, y_{23}, y_{31})$



linear classification model for multiclass classification where labels y $=\pm 1$

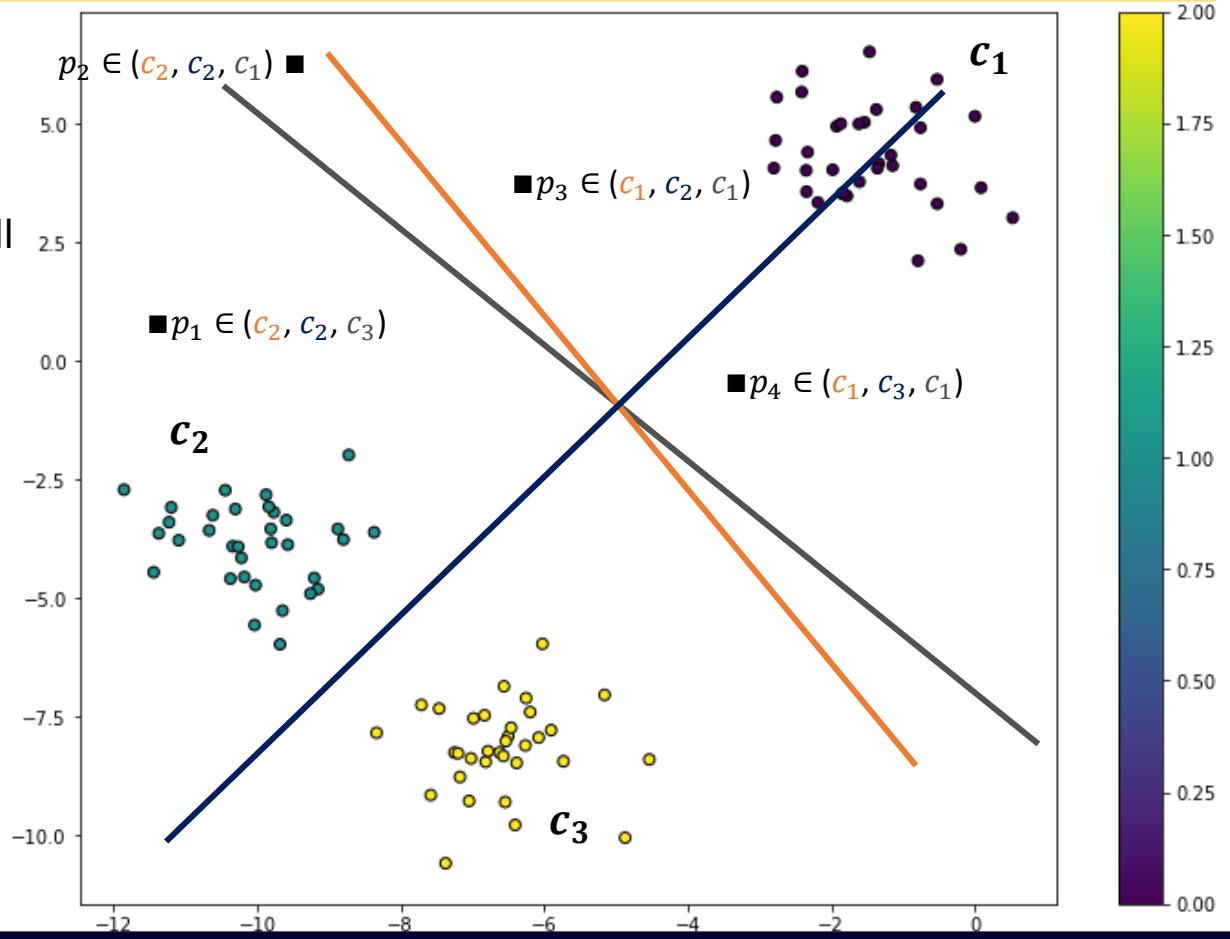
Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$

$p_i \in (y_{12}, y_{23}, y_{31})$



linear classification model for multiclass classification where labels y $=\pm 1$

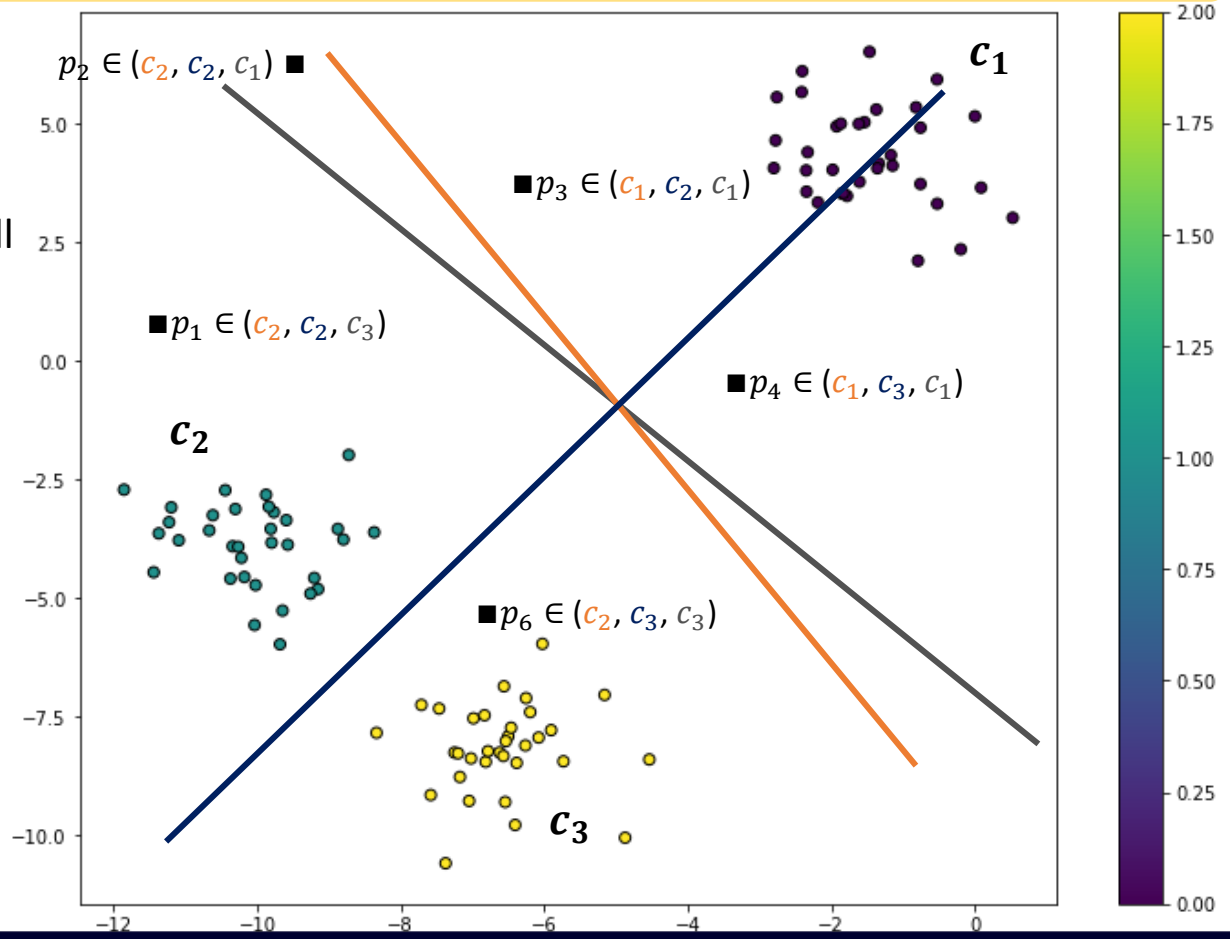
Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$

$p_i \in (y_{12}, y_{23}, y_{31})$



linear classification model for multiclass classification where labels y $= \pm 1$

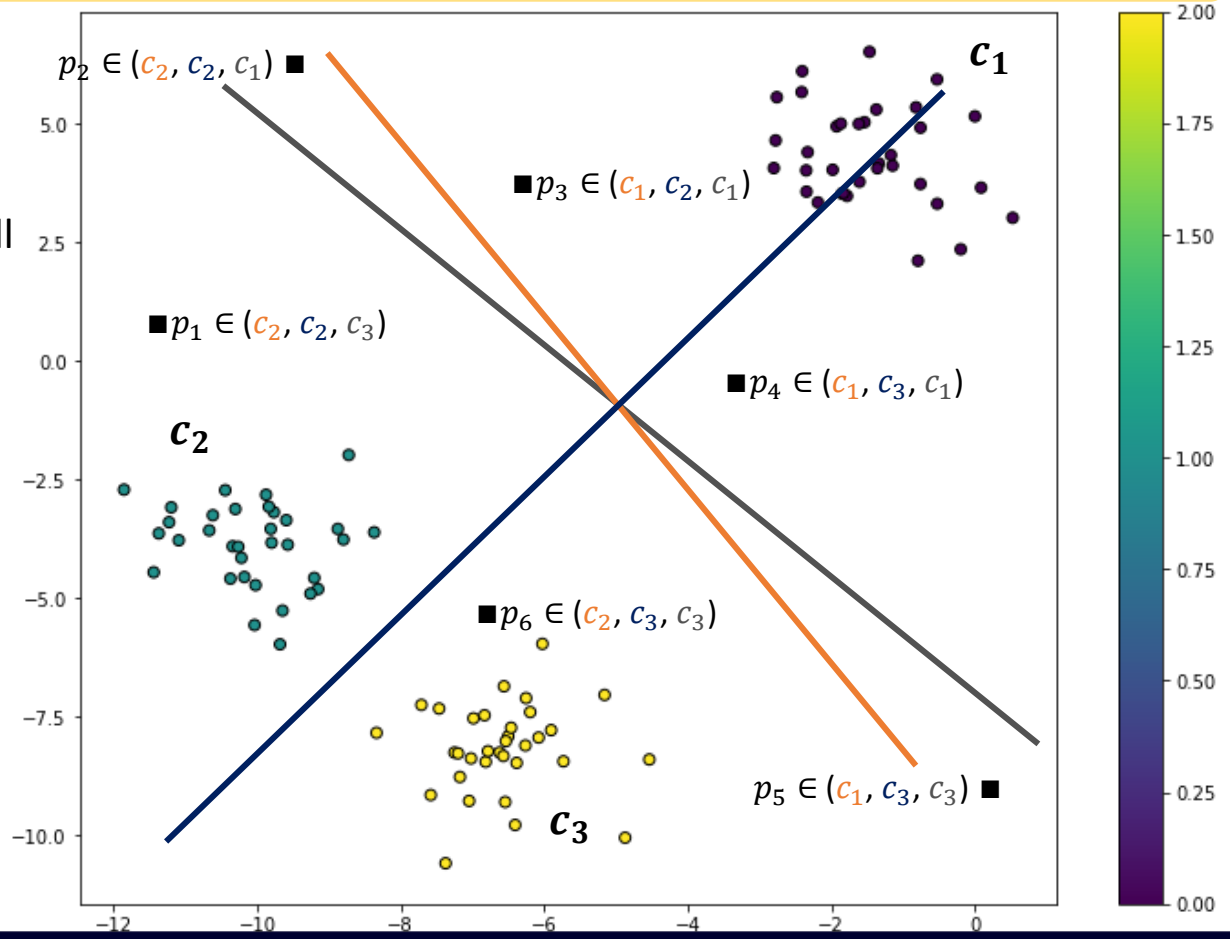
Now, how to solve the multiclass classification problem?

Example: 3 classes $= \{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$

$p_i \in (y_{12}, y_{23}, y_{31})$



linear classification model for multiclass classification where labels y =±1

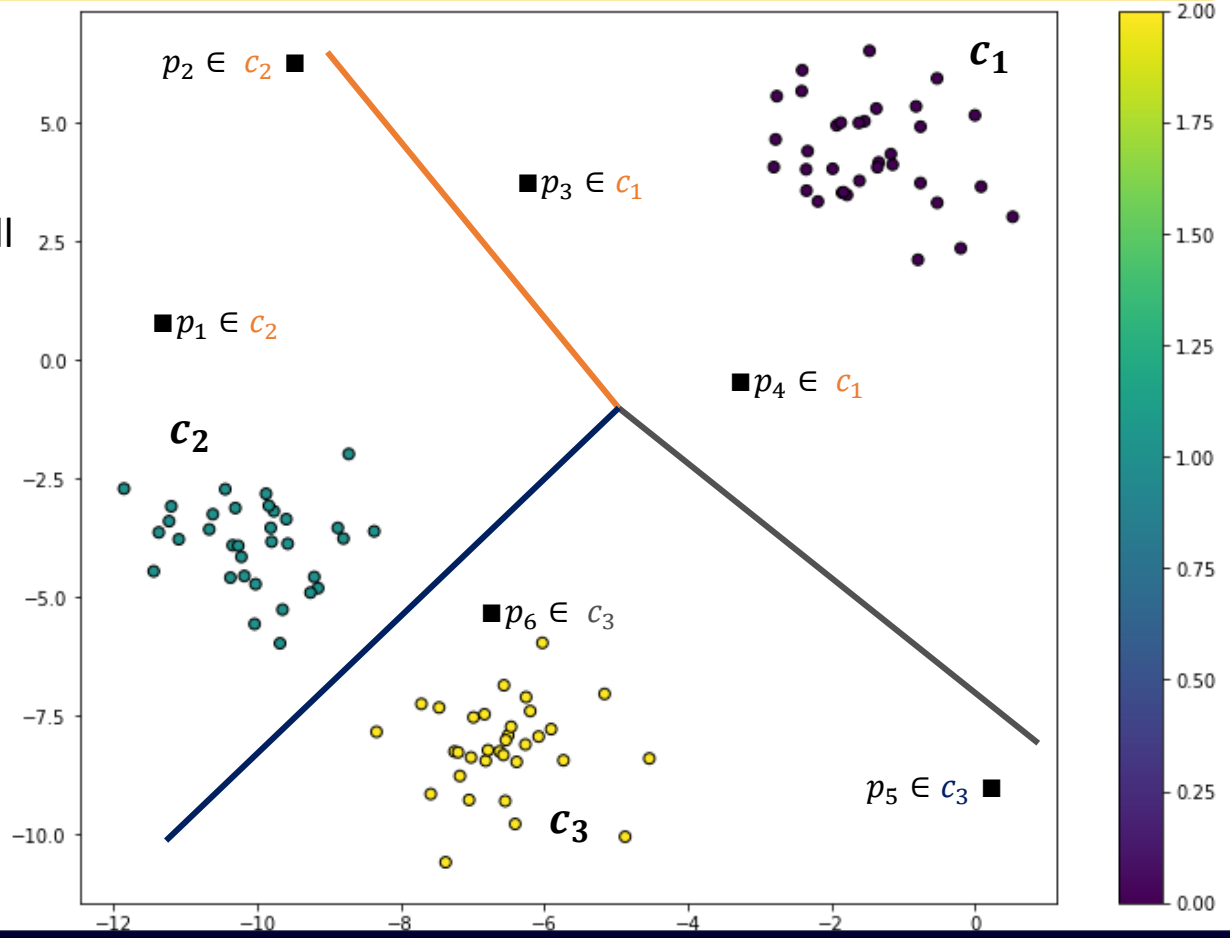
Now, how to solve the multiclass classification problem?

Example: 3 classes = $\{c_1, c_2, c_3\}$

The idea is to use binary classification to classify 2 classes each time. however, we will have 3 **decision functions:**

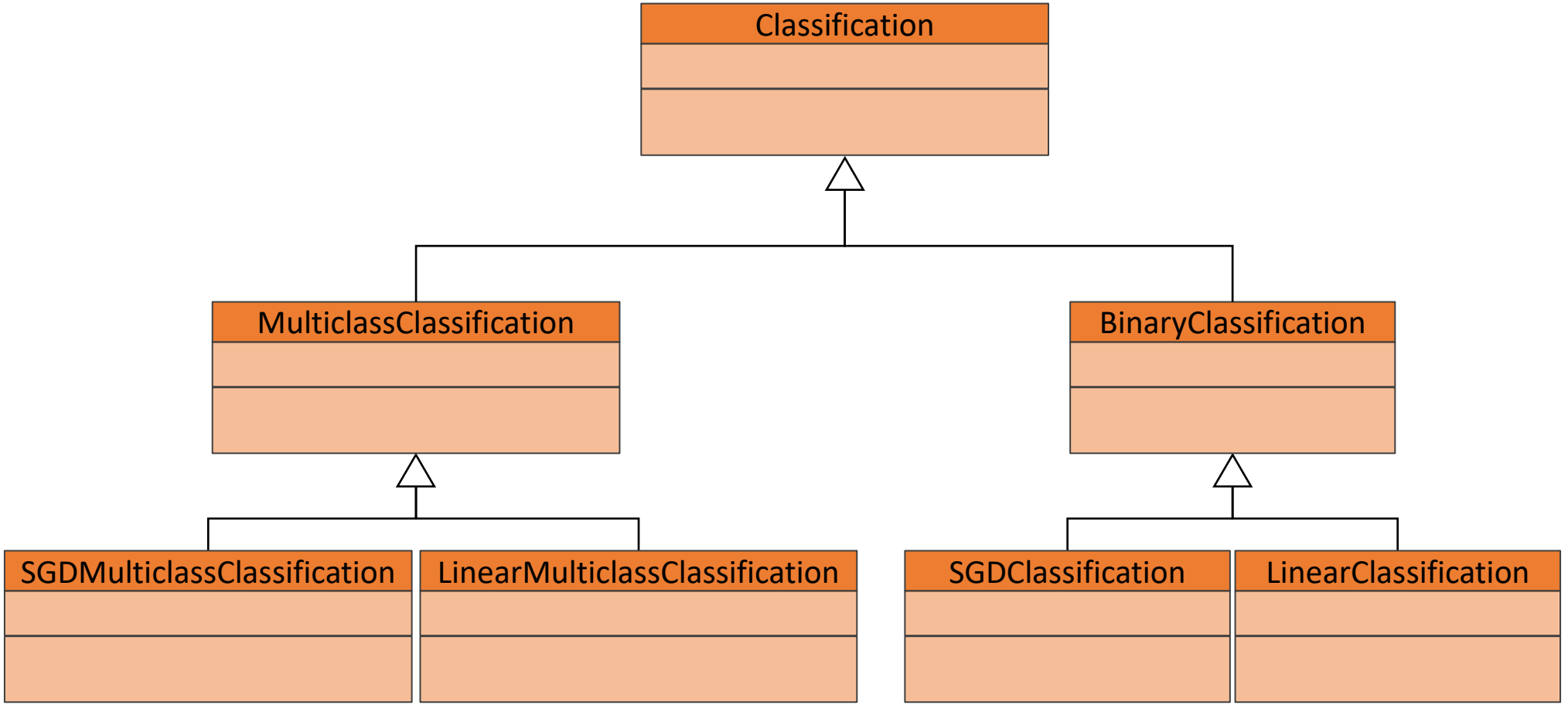
- y_{12} classify $c_1 if\ y_{12} > 0\ else\ c_2$
- y_{23} classify $c_2 if\ y_{23} > 0\ else\ c_3$
- y_{31} classify $c_3 if\ y_{31} > 0\ else\ c_1$

$p_i \in (y_{12}, y_{23}, y_{31})$



linear classification model for multiclass classification where labels $y = \pm 1$

Exercise 2: Complete the code (click colab link ToDo_LinearClassification.ipynb)

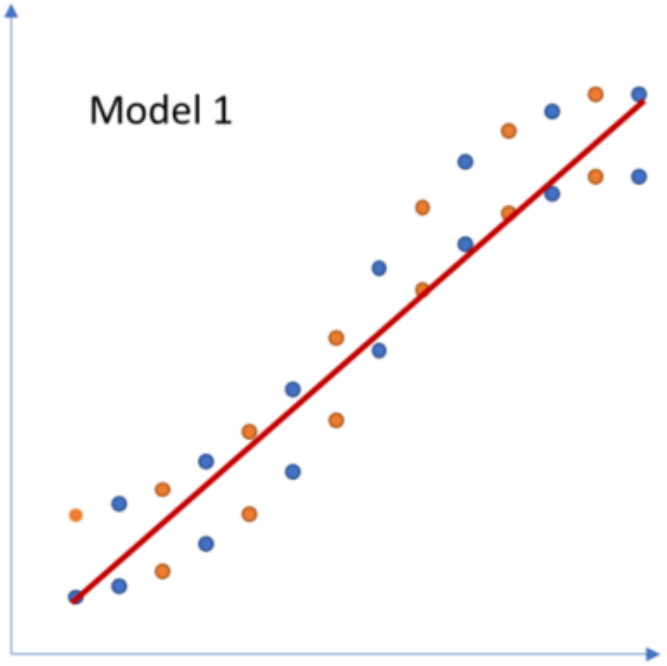


check this link to know more about python inheritance

Underfitting vs Overfitting

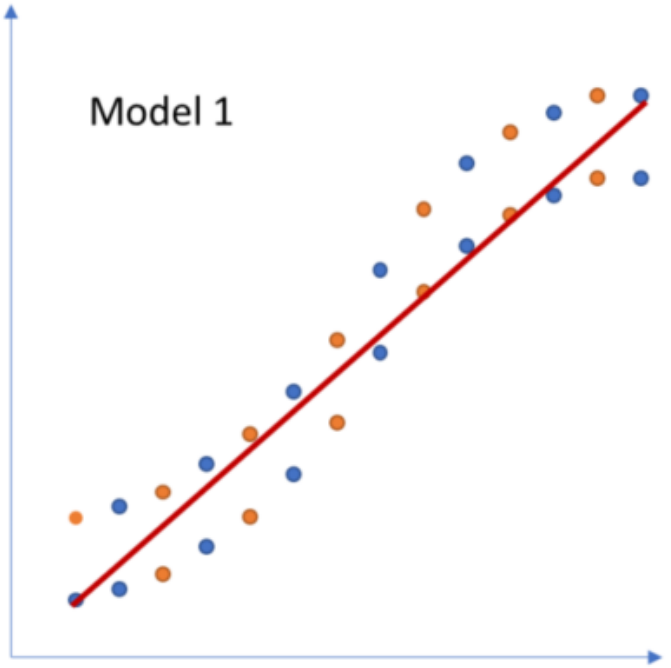
Underfitting vs Overfitting

	Error Model 1: $y = \theta_0 + \theta_1 x$
Training dataset	high
Testing dataset	high

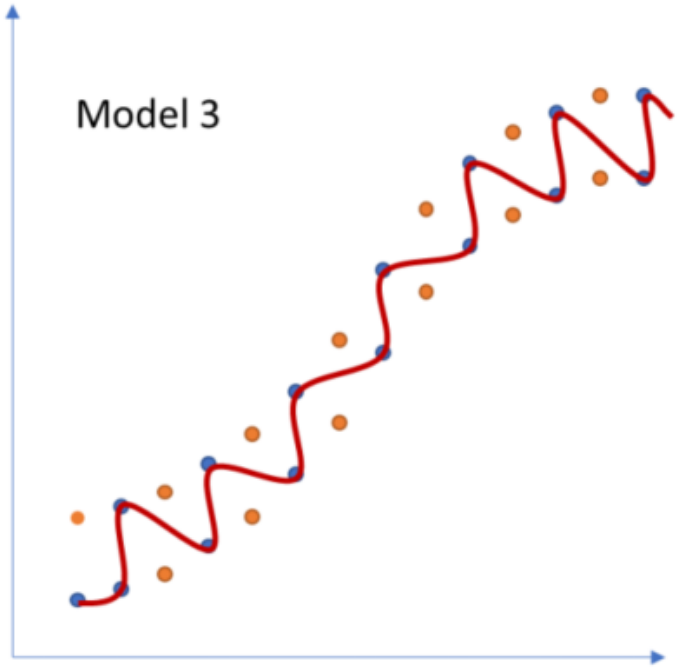


Underfitting vs Overfitting

	Error Model 1: $y = \theta_0 + \theta_1 x$
Training dataset	high
Testing dataset	high

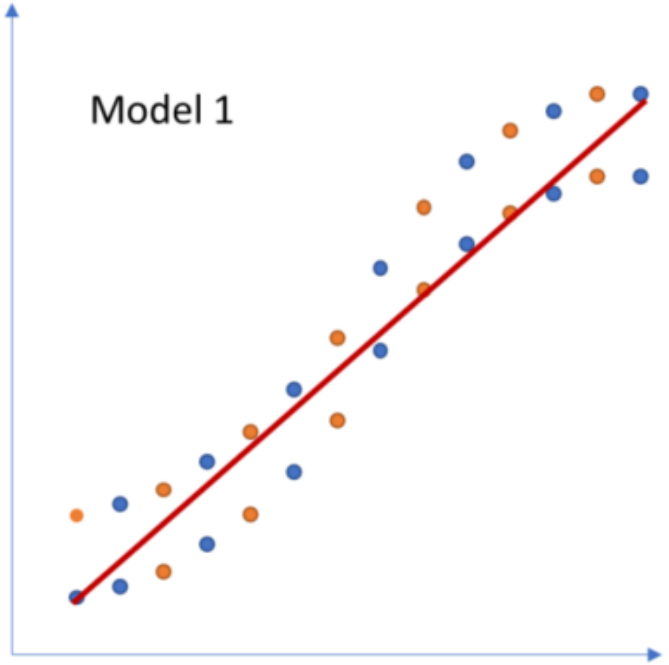


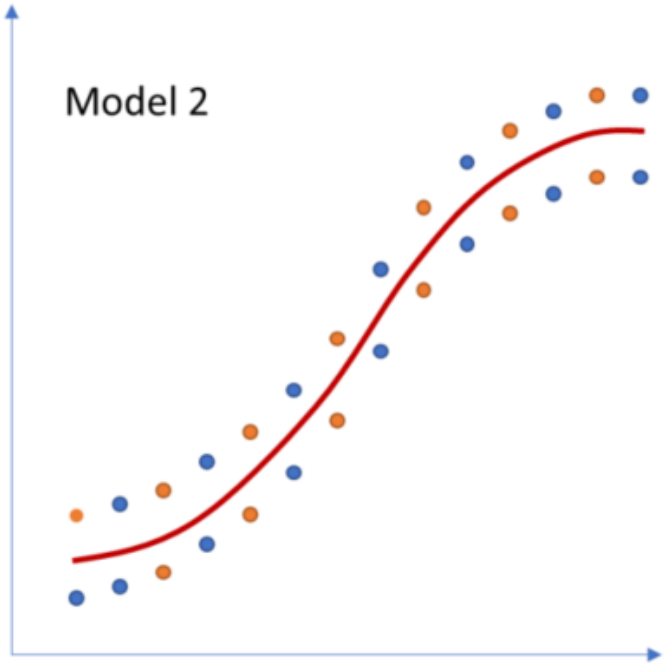
Error Model 3: $y = \theta_0 + \sum \theta_i x^i$
null
medium

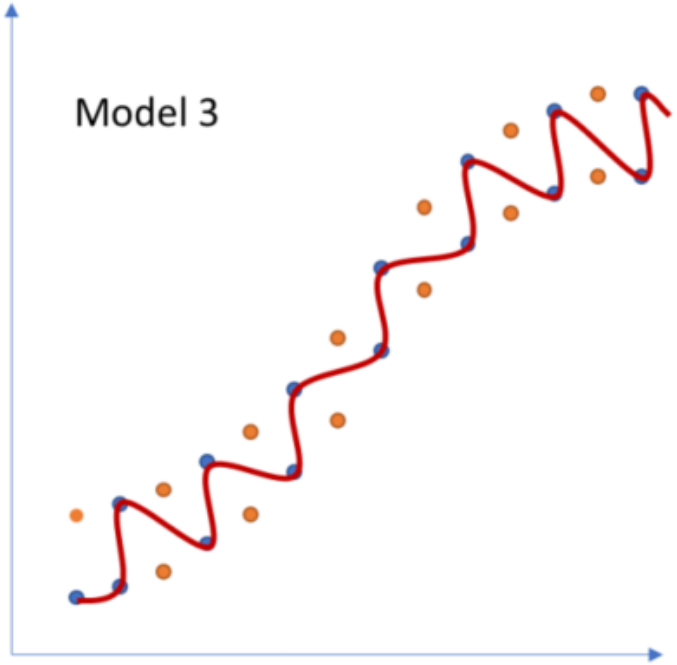


Underfitting vs Overfitting

	Error Model 1: $y = \theta_0 + \theta_1 x$	Error Model 2: $y = \theta_0 + \theta_1 x + \theta_2 x^2$	Error Model 3: $y = \theta_0 + \sum \theta_i x^i$
Training dataset	high	low	null
Testing dataset	high	low	medium

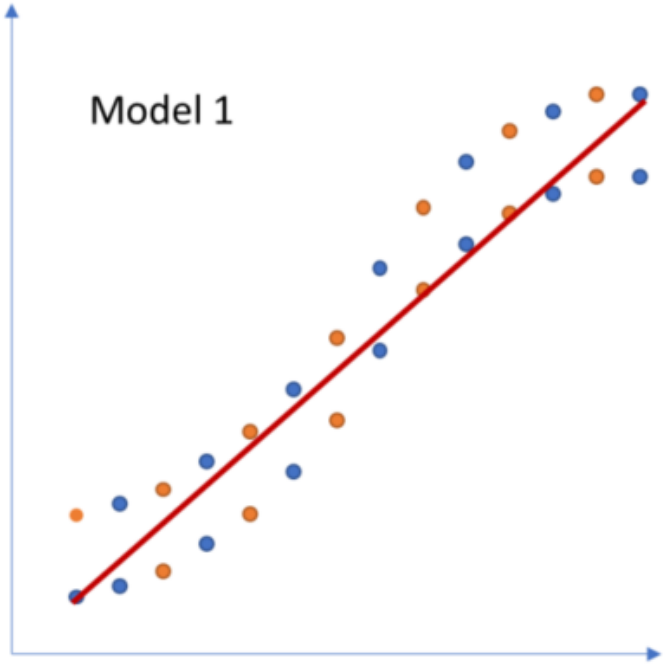




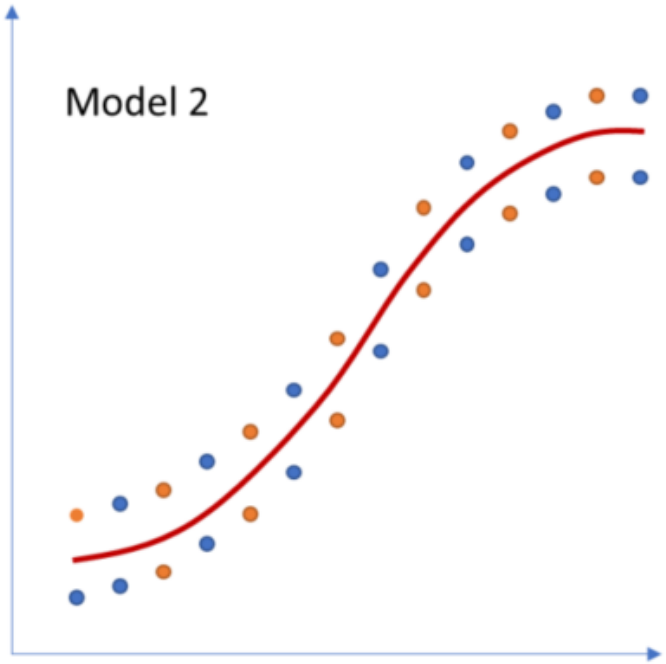


Underfitting vs Overfitting

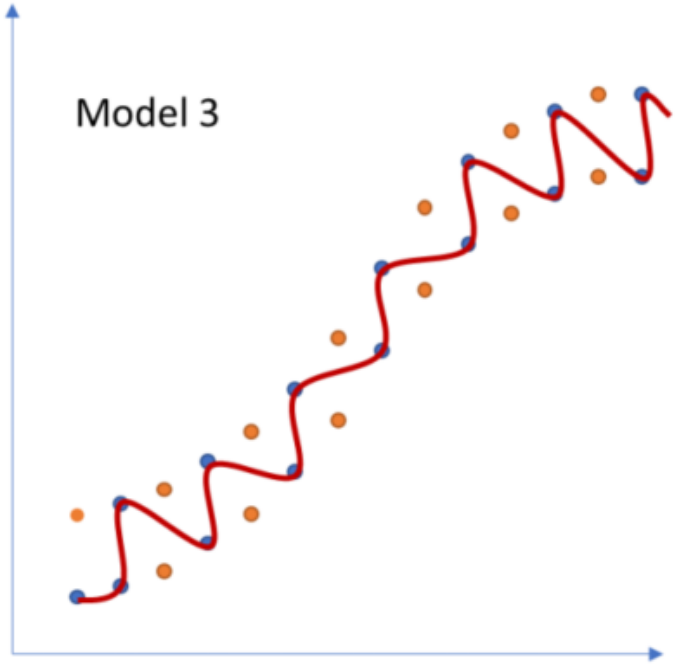
	Error Model 1: $y = \theta_0 + \theta_1 x$	Error Model 2: $y = \theta_0 + \theta_1 x + \theta_2 x^2$	Error Model 3: $y = \theta_0 + \sum \theta_i x^i$
Training dataset	high	low	null
Testing dataset	high	low	medium



Under-fitting



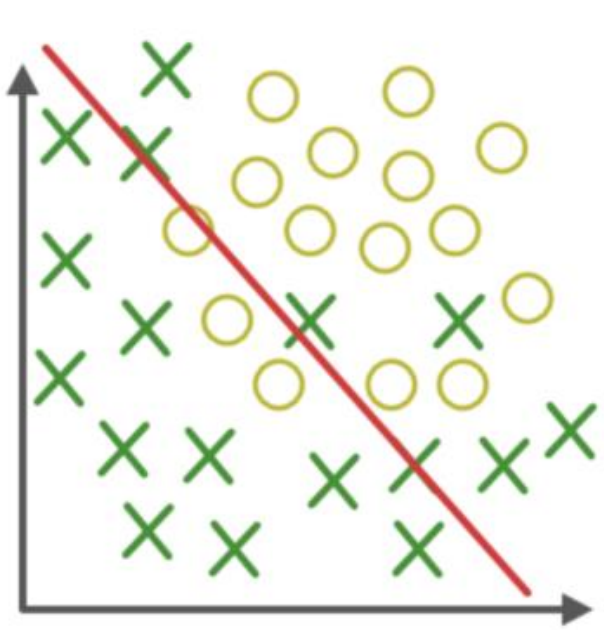
Appropirate-fitting



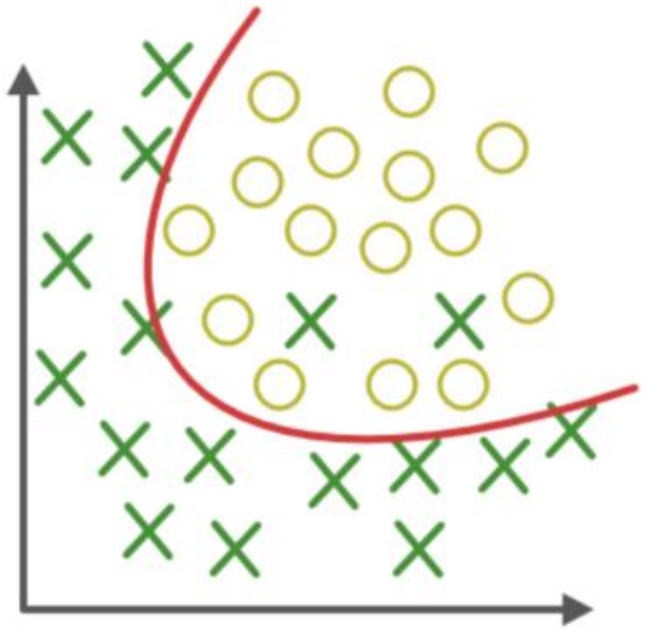
Over-fitting

Underfitting vs Overfitting

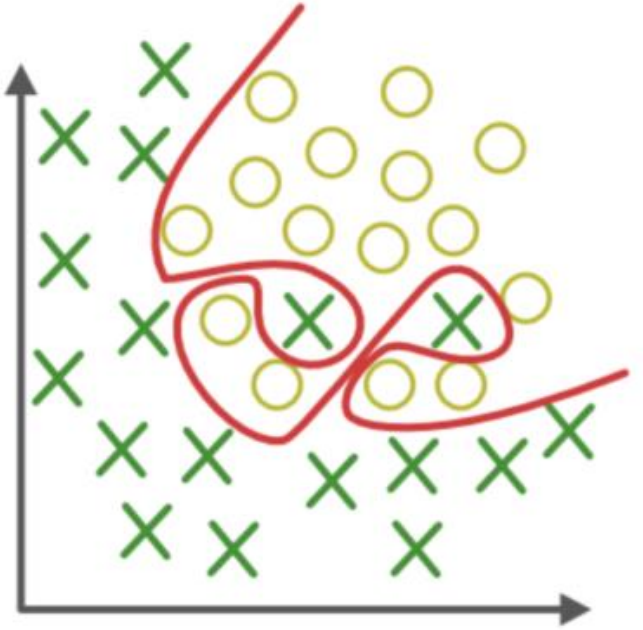
	Error Model 1: $y = \theta_0 + \theta_1 x$	Error Model 2: $y = \theta_0 + \theta_1 x + \theta_2 x^2$	Error Model 3: $y = \theta_0 + \sum \theta_i x^i$
Training dataset	high	low	null
Testing dataset	high	low	medium



Under-fitting

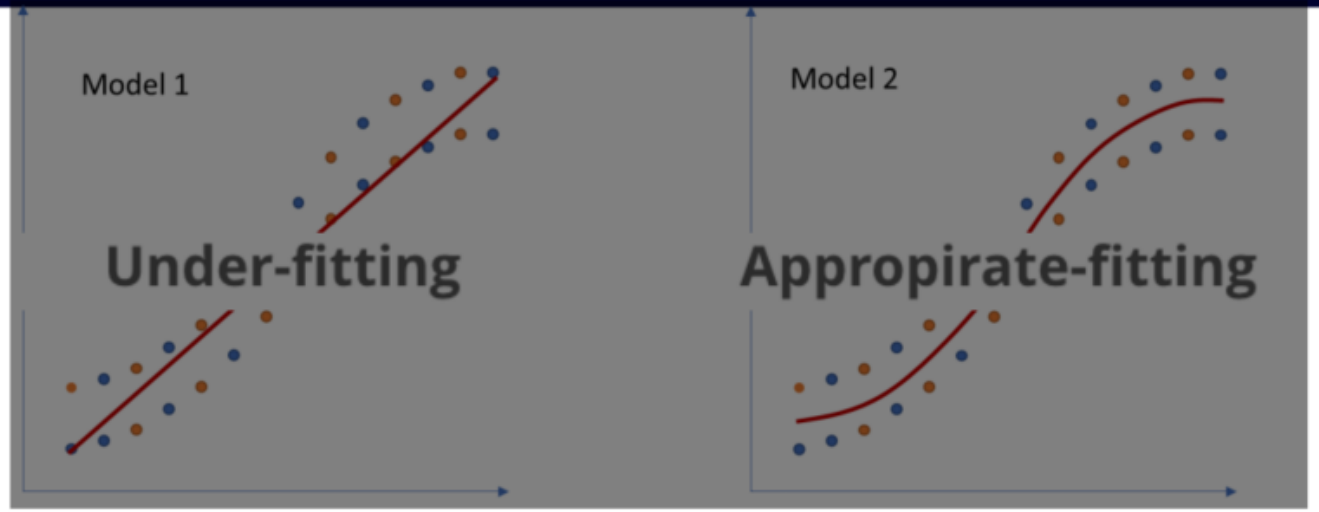


Appropirate-fitting

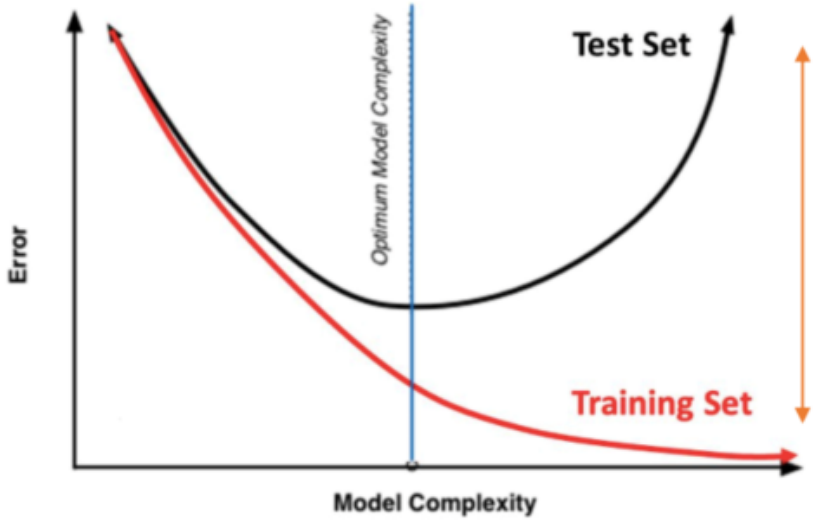


Over-fitting

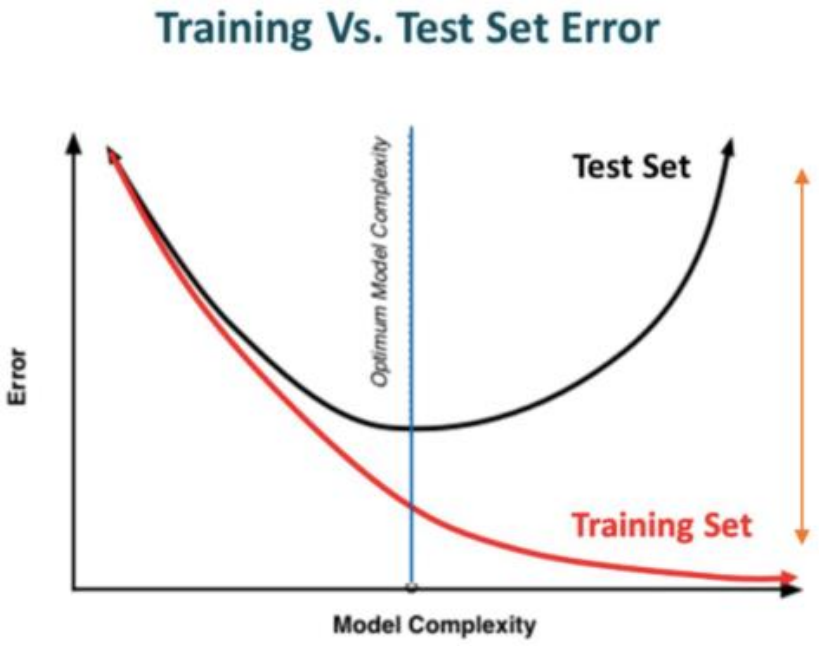
Underfitting vs Overfitting



Training Vs. Test Set Error



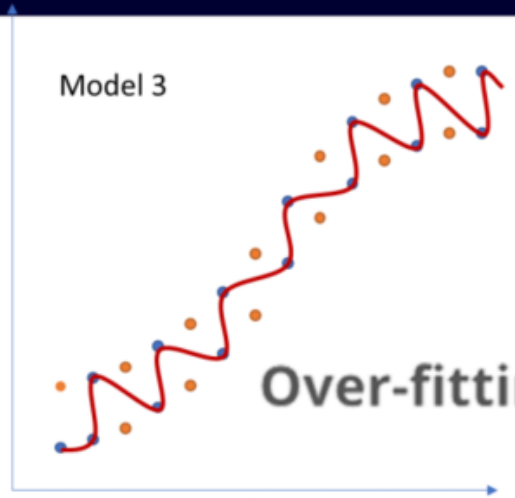
Underfitting vs Overfitting

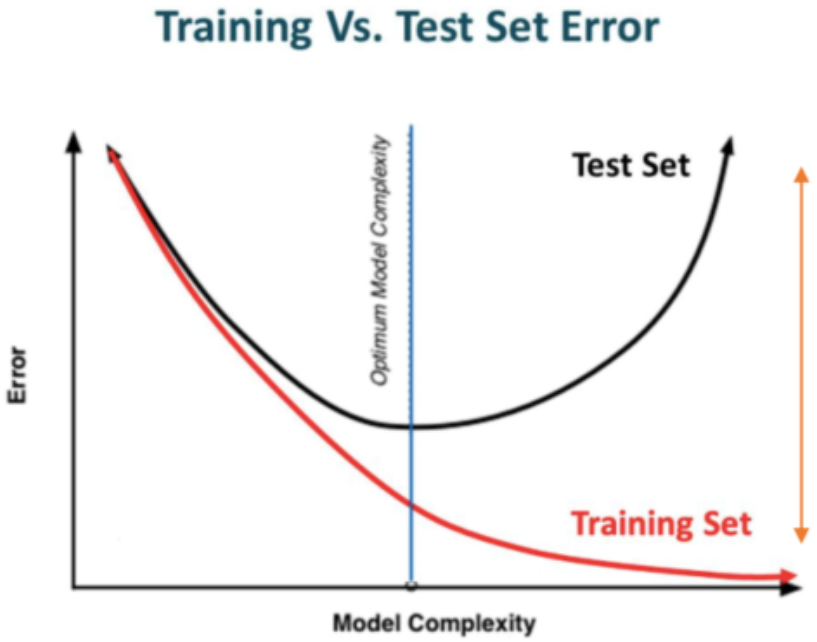


$$y = \theta_0 + \theta_1 \boldsymbol{x} + \theta_2 \boldsymbol{x^2} + \theta_3 \boldsymbol{x^3} + \theta_4 \boldsymbol{x^4} + \theta_5 \boldsymbol{x^5} + \cdots$$

$$\min_{\theta} loss\ function\ l = \frac{1}{2N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Underfitting vs Overfitting

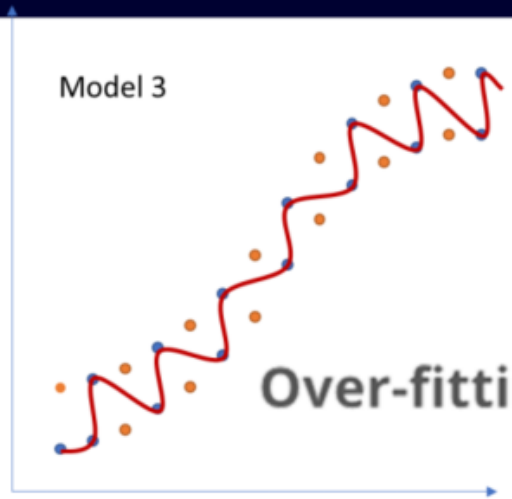


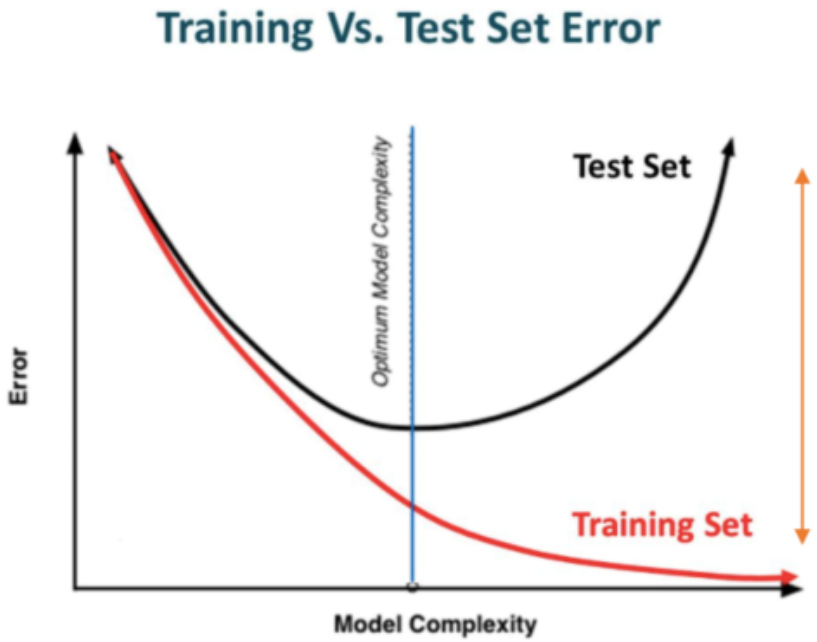


$$y = \theta_0 + \theta_1 \boldsymbol{x} + \theta_2 \boldsymbol{x^2} + \theta_3 \boldsymbol{x^3} + \theta_4 \boldsymbol{x^4} + \theta_5 \boldsymbol{x^5} + \cdots$$

$$\min_{\theta} loss\ function\ l = \frac{1}{2N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \ \boldsymbol{+ 100} * (|\boldsymbol{\theta_3}| + |\boldsymbol{\theta_4}| + |\boldsymbol{\theta_5}| + \cdots)$$

Underfitting vs Overfitting





$$y = \theta_0 + \theta_1 \boldsymbol{x} + \theta_2 \boldsymbol{x^2} + \theta_3 \boldsymbol{x^3} + \theta_4 \boldsymbol{x^4} + \theta_5 \boldsymbol{x^5} + \cdots$$

$$\min_{\theta} loss\ function\ l = \frac{1}{2N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \quad \underbrace{+ \mathbf{100} * (|\boldsymbol{\theta_3}| + |\boldsymbol{\theta_4}| + |\boldsymbol{\theta_5}| + \cdots)}_{\textbf{Penalization}}$$

of the loss function

Regularization in Machine Learning

Ridge Regression (L2 regularization)

$$\min_{\theta} loss\ function\ l = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \ + \boldsymbol{\lambda}\sum_{j=1}^{M}\boldsymbol{\theta_j^2}$$

- Reduce the overfitting
- **Homogeneous all parameters**

Lasso Regression (L1 regularization)

$$\min_{\theta} loss\ function\ l = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \ + \boldsymbol{\lambda}\sum_{j=1}^{M}\boldsymbol{|\theta_j|}$$

- Reduce the overfitting
- **Feature selection**

Regularization in Machine Learning

ElasticNet Regression (L1+L2 regularization)

$$\min_{\theta} loss\ function\ l = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \boldsymbol{\lambda_1}\sum_{j=1}^{M}|\boldsymbol{\theta_j}| + \boldsymbol{\lambda_2}\sum_{j=1}^{M}\boldsymbol{\theta_j^2}$$

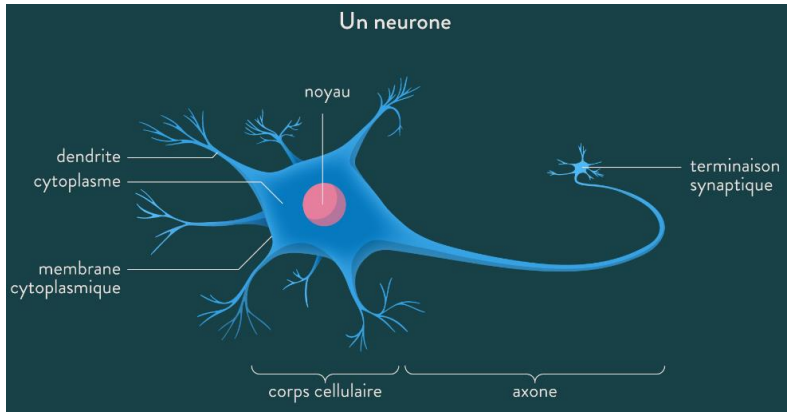
or

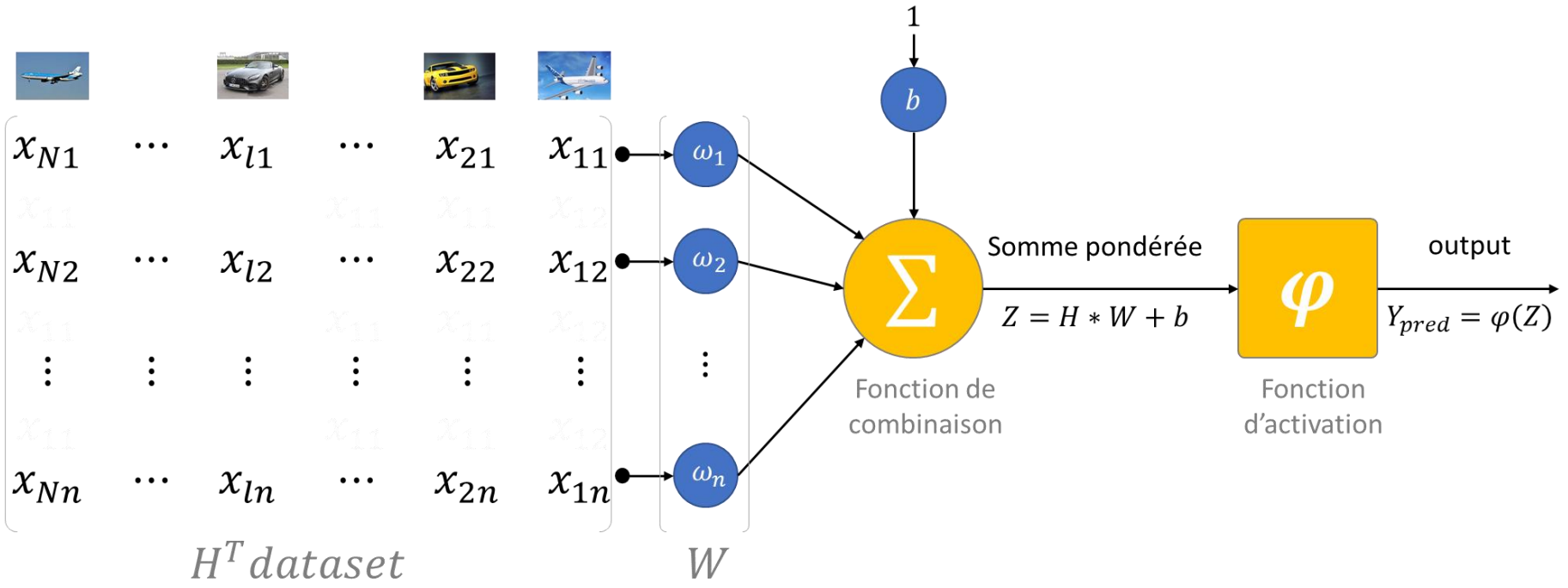
$$\min_{\theta} loss\ function\ l = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \boldsymbol{\lambda}\left(\boldsymbol{\alpha}\sum_{j=1}^{M}|\boldsymbol{\theta_j}| + \frac{\boldsymbol{1-\alpha}}{\boldsymbol{2}}\sum_{j=1}^{M}\boldsymbol{\theta_j^2}\right)$$

- Reduce the overfitting
- **Homogeneous all parameters**
- **Feature selection**

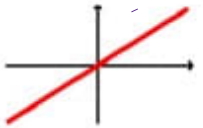
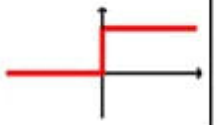
Artificial neural networks (ANNs)

Artificial neural networks (ANNs)

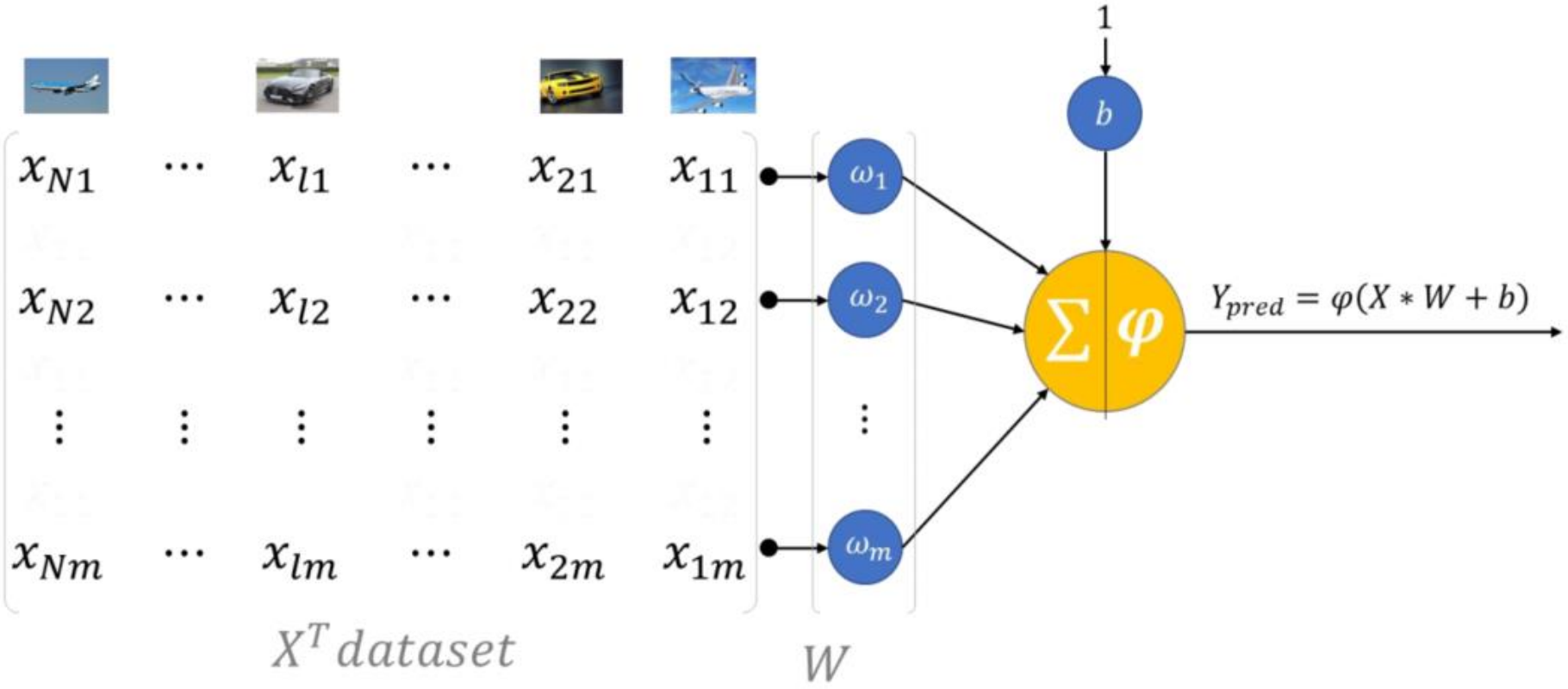




Fonction d'activation

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1 & z \geq \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

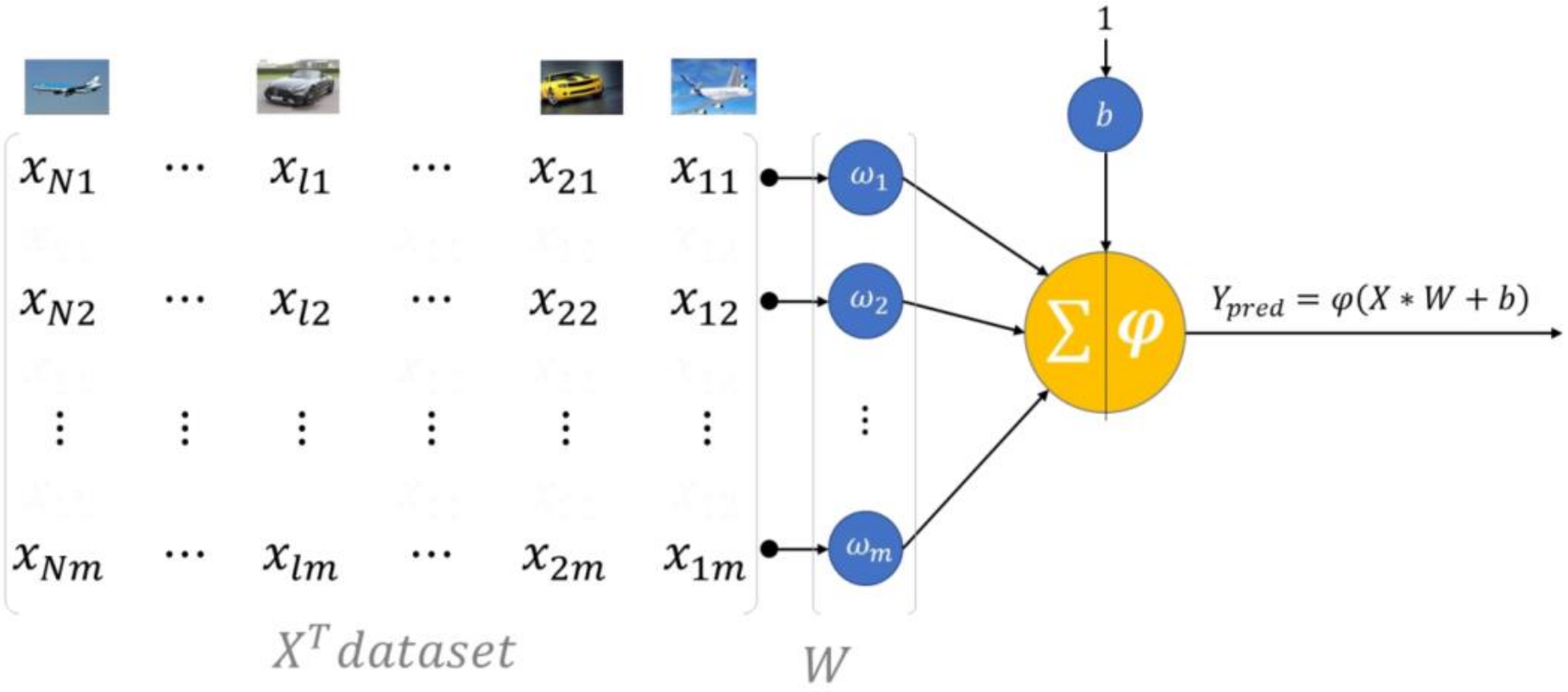
How to adapt the **weights *W*** et **bias b** of this neuron? (how to calculate the gradients of these **weights** and **bias**)



Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textbf{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$



Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$MSE\ loss\ function:\ l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\omega_k = \omega_k - \rho\frac{\partial l}{\partial \omega_k} \qquad \frac{\partial l}{\partial \omega_k} = \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial e_i}\frac{\partial e_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textit{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\omega_k = \omega_k - \rho\frac{\partial l}{\partial \omega_k} \qquad \frac{\partial l}{\partial \omega_k} = \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial e_i}\frac{\partial e_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$$

$$l_i = e_i^2 \longrightarrow \frac{\partial l_i}{\partial e_i} = 2e_i$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textit{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\omega_k = \omega_k - \rho\frac{\partial l}{\partial \omega_k} \qquad \frac{\partial l}{\partial \omega_k} = \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial e_i}\frac{\partial e_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$$

$$l_i = e_i^2 \longrightarrow \frac{\partial l_i}{\partial e_i} = 2e_i$$

$$e_i = y_i - \varphi(z_i) \longrightarrow \frac{\partial e_i}{\partial z_i} = -\dot{\varphi}(z_i)$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textit{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\omega_k = \omega_k - \rho\frac{\partial l}{\partial \omega_k} \qquad \frac{\partial l}{\partial \omega_k} = \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial e_i}\frac{\partial e_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$$

$$l_i = e_i^2 \longrightarrow \frac{\partial l_i}{\partial e_i} = 2e_i$$

$$e_i = y_i - \varphi(z_i) \longrightarrow \frac{\partial e_i}{\partial z_i} = -\dot{\varphi}(z_i)$$

$$z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b \longrightarrow \frac{\partial z_i}{\partial \omega_k} = x_{ik}$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textit{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\omega_k = \omega_k - \rho\frac{\partial l}{\partial \omega_k} \qquad \frac{\partial l}{\partial \omega_k} = \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial e_i}\frac{\partial e_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$$

$$l_i = e_i^2 \longrightarrow \frac{\partial l_i}{\partial e_i} = 2e_i$$

$$e_i = y_i - \varphi(z_i) \longrightarrow \frac{\partial e_i}{\partial z_i} = -\dot{\varphi}(z_i)$$

$$z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b \longrightarrow \frac{\partial z_i}{\partial \omega_k} = x_{ik}$$

$$\longrightarrow \quad \frac{\partial l}{\partial \omega_k} = -\frac{1}{N}\sum_{i=1}^{N} e_i \dot{\varphi}(z_i) x_{ik}$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textit{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\omega_k = \omega_k - \rho\frac{\partial l}{\partial \omega_k} \qquad \frac{\partial l}{\partial \omega_k} = \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial e_i}\frac{\partial e_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$$

$$l_i = e_i^2 \longrightarrow \frac{\partial l_i}{\partial e_i} = 2e_i$$

$$e_i = y_i - \varphi(z_i) \longrightarrow \frac{\partial e_i}{\partial z_i} = -\dot{\varphi}(z_i)$$

$$z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b \longrightarrow \frac{\partial z_i}{\partial \omega_k} = x_{ik}$$

$$\frac{\partial l}{\partial \omega_k} = -\frac{1}{N}\sum_{i=1}^{N} e_i \dot{\varphi}(z_i) x_{ik}$$

$$\frac{\partial l}{\partial b} = -\frac{1}{N}\sum_{i=1}^{N} e_i \dot{\varphi}(z_i)$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textbf{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\frac{\partial l}{\partial \omega_k} = \Delta\omega_k = -\frac{1}{N}\sum_{i=1}^{N} e_i \dot{\varphi}(z_i) x_{ik} = -\frac{1}{N}\left(e_1\dot{\varphi}(z_1)x_{1k} + e_2\dot{\varphi}(z_2)x_{2k} + \cdots + e_N\dot{\varphi}(z_N)x_{Nk}\right)$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textbf{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\frac{\partial l}{\partial \omega_k} = \Delta\omega_k = -\frac{1}{N}\sum_{i=1}^{N} e_i \dot{\varphi}(z_i) x_{ik} = -\frac{1}{N}\left(e_1\dot{\varphi}(z_1)x_{1k} + e_2\dot{\varphi}(z_2)x_{2k} + \cdots + e_N\dot{\varphi}(z_N)x_{Nk}\right)$$

$$\begin{pmatrix} \Delta\omega_1 \\ \Delta\omega_2 \\ \vdots \\ \Delta\omega_m \end{pmatrix} = -\frac{1}{N}\begin{pmatrix} e_1\dot{\varphi}(z_1)x_{11} + e_2\dot{\varphi}(z_2)x_{21} + \cdots + e_N\dot{\varphi}(z_N)x_{N1} \\ e_1\dot{\varphi}(z_1)x_{12} + e_2\dot{\varphi}(z_2)x_{22} + \cdots + e_N\dot{\varphi}(z_N)x_{N2} \\ \vdots \\ e_1\dot{\varphi}(z_1)x_{1m} + e_2\dot{\varphi}(z_2)x_{2m} + \cdots + e_N\dot{\varphi}(z_N)x_{Nm} \end{pmatrix}$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textbf{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\frac{\partial l}{\partial \omega_k} = \Delta\omega_k = -\frac{1}{N}\sum_{i=1}^{N} e_i \dot{\varphi}(z_i) x_{ik} = -\frac{1}{N}\left(e_1\dot{\varphi}(z_1)x_{1k} + e_2\dot{\varphi}(z_2)x_{2k} + \cdots + e_N\dot{\varphi}(z_N)x_{Nk}\right)$$

$$\begin{pmatrix} \Delta\omega_1 \\ \Delta\omega_2 \\ \vdots \\ \Delta\omega_m \end{pmatrix} = -\frac{1}{N}\begin{pmatrix} e_1\dot{\varphi}(z_1)x_{11} + e_2\dot{\varphi}(z_2)x_{21} + \cdots + e_N\dot{\varphi}(z_N)x_{N1} \\ e_1\dot{\varphi}(z_1)x_{12} + e_2\dot{\varphi}(z_2)x_{22} + \cdots + e_N\dot{\varphi}(z_N)x_{N2} \\ \vdots \\ e_1\dot{\varphi}(z_1)x_{1m} + e_2\dot{\varphi}(z_2)x_{2m} + \cdots + e_N\dot{\varphi}(z_N)x_{Nm} \end{pmatrix}$$

$$\begin{pmatrix} \Delta\omega_1 \\ \Delta\omega_2 \\ \vdots \\ \Delta\omega_m \end{pmatrix} = -\frac{1}{N}\begin{pmatrix} x_{11} & x_{21} & \cdots & x_{N1} \\ x_{12} & x_{22} & \cdots & x_{N2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1m} & x_{2m} & \cdots & x_{Nm} \end{pmatrix}\begin{pmatrix} e_1\dot{\varphi}(z_1) \\ e_2\dot{\varphi}(z_2) \\ \vdots \\ e_N\dot{\varphi}(z_N) \end{pmatrix}$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

$$\textbf{MSE loss function: } l = \frac{1}{2N}\sum_{i=1}^{N} l_i = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi(z_i)\right)^2 \quad | \quad z_i = \sum_{k=1}^{m} \omega_k x_{ik} + b$$

$$\frac{\partial l}{\partial \omega_k} = \Delta\omega_k = -\frac{1}{N}\sum_{i=1}^{N} e_i \dot{\varphi}(z_i) x_{ik} = -\frac{1}{N}\left(e_1\dot{\varphi}(z_1)x_{1k} + e_2\dot{\varphi}(z_2)x_{2k} + \cdots + e_N\dot{\varphi}(z_N)x_{Nk}\right)$$

$$\begin{pmatrix} \Delta\omega_1 \\ \Delta\omega_2 \\ \vdots \\ \Delta\omega_m \end{pmatrix} = -\frac{1}{N}\begin{pmatrix} e_1\dot{\varphi}(z_1)x_{11} + e_2\dot{\varphi}(z_2)x_{21} + \cdots + e_N\dot{\varphi}(z_N)x_{N1} \\ e_1\dot{\varphi}(z_1)x_{12} + e_2\dot{\varphi}(z_2)x_{22} + \cdots + e_N\dot{\varphi}(z_N)x_{N2} \\ \vdots \\ e_1\dot{\varphi}(z_1)x_{1m} + e_2\dot{\varphi}(z_2)x_{2m} + \cdots + e_N\dot{\varphi}(z_N)x_{Nm} \end{pmatrix}$$

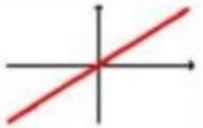
$$\begin{pmatrix} \Delta\omega_1 \\ \Delta\omega_2 \\ \vdots \\ \Delta\omega_m \end{pmatrix} = -\frac{1}{N}\begin{pmatrix} x_{11} & x_{21} & \dots & x_{N1} \\ x_{12} & x_{22} & \dots & x_{N2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1m} & x_{2m} & \cdots & x_{Nm} \end{pmatrix}\begin{pmatrix} e_1\dot{\varphi}(z_1) \\ e_2\dot{\varphi}(z_2) \\ \vdots \\ e_N\dot{\varphi}(z_N) \end{pmatrix}$$

$$\Delta W = -\frac{1}{N}X^T\bar{E}$$

$$\Delta b = -\frac{1}{N}I^T\bar{E} \quad with\ I\ is\ Matrix\ of\ ones$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with MSE loss function

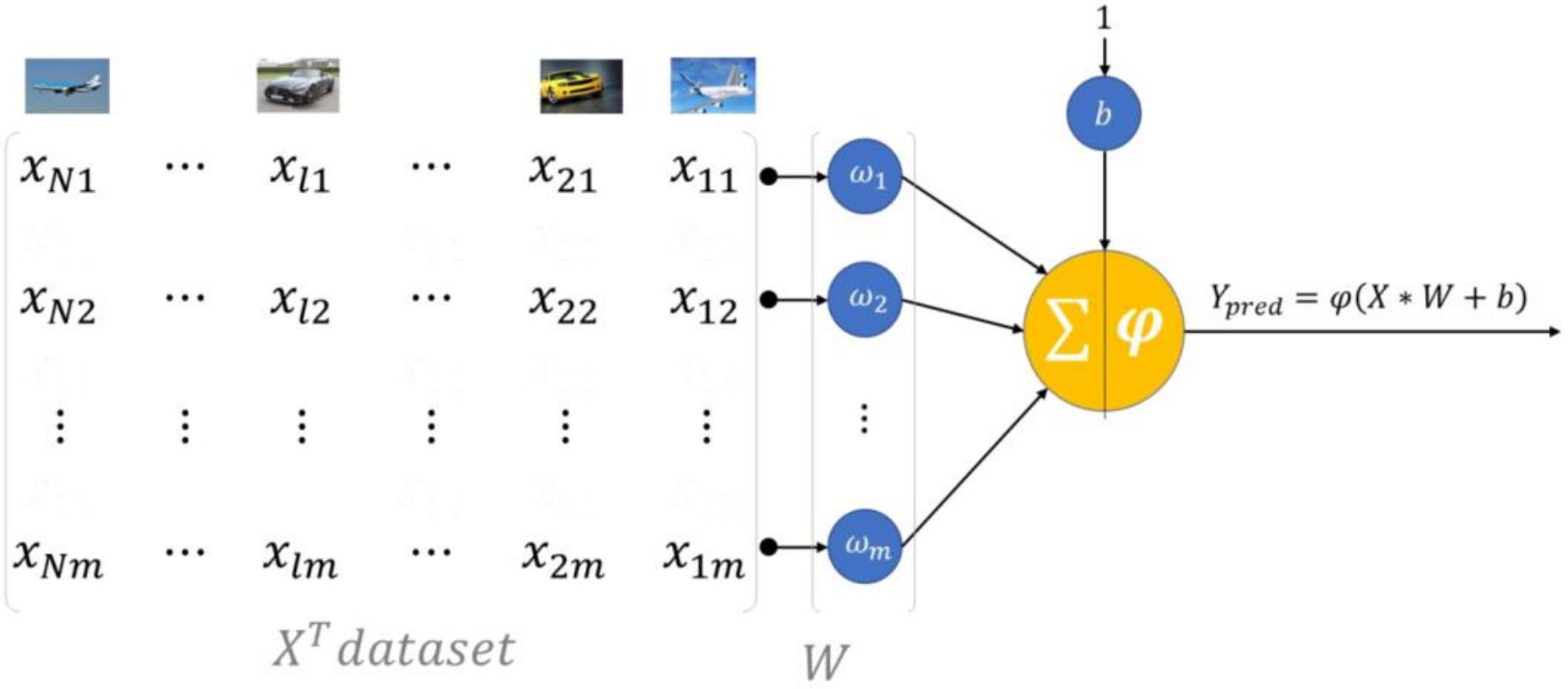
Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise Linear	$\phi(z)=\begin{cases} 0 & z \le -½ \\ z + ½ & -½ \le z \le ½ \\ 1 & z \ge ½ \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z)=\frac{1}{1+e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z)=\begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Derivative	Gradient with $\hat{y}_i = \varphi(z_i)$
$\dot{\varphi} = 1$	$\frac{\partial l}{\partial \theta_k} = -\frac{1}{N}\sum_{i=1}^{N} e_i x_{ik}$
$\dot{\varphi} = \begin{cases} 1 & \lvert z\rvert \le 0{,}5 \\ 0 & else \end{cases}$	$\frac{\partial l}{\partial \theta_k} = -\frac{1}{N}\sum_{i=1}^{N} e_i x_{ik}$
$\dot{\varphi} = \varphi(1-\varphi)$	$\frac{\partial l}{\partial \theta_k} = -\frac{1}{N}\sum_{i=1}^{N} e_i \left(\hat{y}_i(1-\hat{y}_i)\right) x_{ik}$
$\dot{\varphi} = 1-\varphi^2$	$\frac{\partial l}{\partial \theta_k} = -\frac{1}{N}\sum_{i=1}^{N} e_i \left(\hat{y}_i(1-\hat{y}_i)\right) x_{ik}$
$\dot{\varphi} = \begin{cases} 0 & z \le 0 \\ 1 & z > 0 \end{cases}$	$\frac{\partial l}{\partial \theta_k} = -\frac{1}{N}\sum_{i=1}^{N} e_i x_{ik}$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

Cross-entropy loss, also known as negative log-likelihood loss, measures the dissimilarity between the predicted probability distribution and the true distribution by computing the negative log-likelihood of the true distribution.
Cross-entropy can be used only with an activation function which represent a probability distribution like sigmoid, softmax.

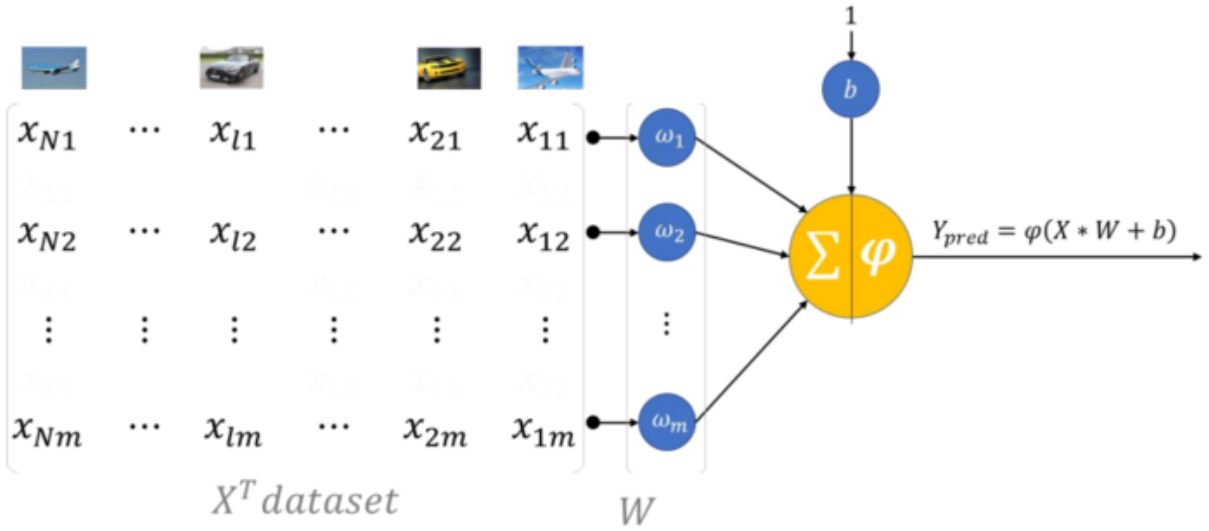


Artificial neural networks (ANNs)

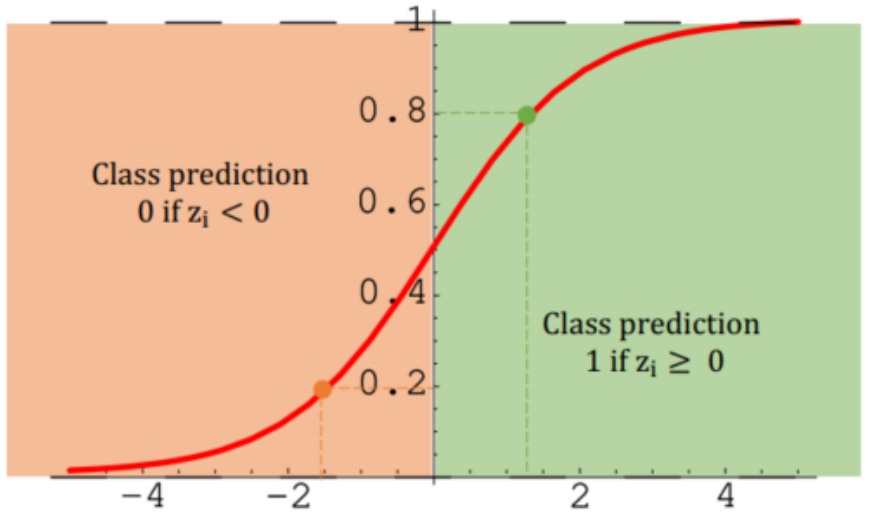
PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

Cross-entropy loss, also known as negative log-likelihood loss, measures the dissimilarity between the predicted probability distribution and the true distribution by computing the negative log-likelihood of the true distribution.
Cross-entropy can be used only with an activation function which represent a probability distribution like sigmoid, softmax.

$$\boldsymbol{Cross-entropy\ loss:}\ l = \frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i\, log(\hat{y}_i) - (1-y_i) log(1-\hat{y}_i)$$



$$\hat{y}_i = sigm(z_i) = 1/(1+e^{-z_i})$$



$$z_i = X_i * W + b$$

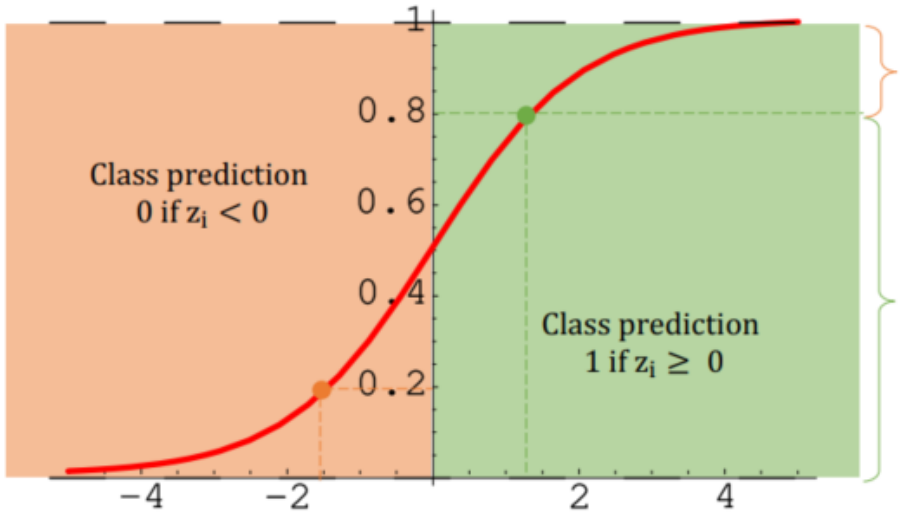
Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

Cross-entropy loss, also known as negative log-likelihood loss, measures the dissimilarity between the predicted probability distribution and the true distribution by computing the negative log-likelihood of the true distribution.
Cross-entropy can be used only with an activation function which represent a probability distribution like sigmoid, softmax.

$$\textbf{Cross-entropy loss}: \; l = \frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i \, log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i)$$

$$\hat{y}_i = s(z_i) = 1/(1 + e^{-z_i})$$



$$P(Y = 0) = 1 - s(z_i)$$

$$P(Y = 1) = s(z_i)$$

Bernoulli distribution:

$$P(Y = y_i) = s(z_i)^{y_i} * (1 - s(z_i))^{1 - y_i}$$

$$z_i = X_i * W + b$$

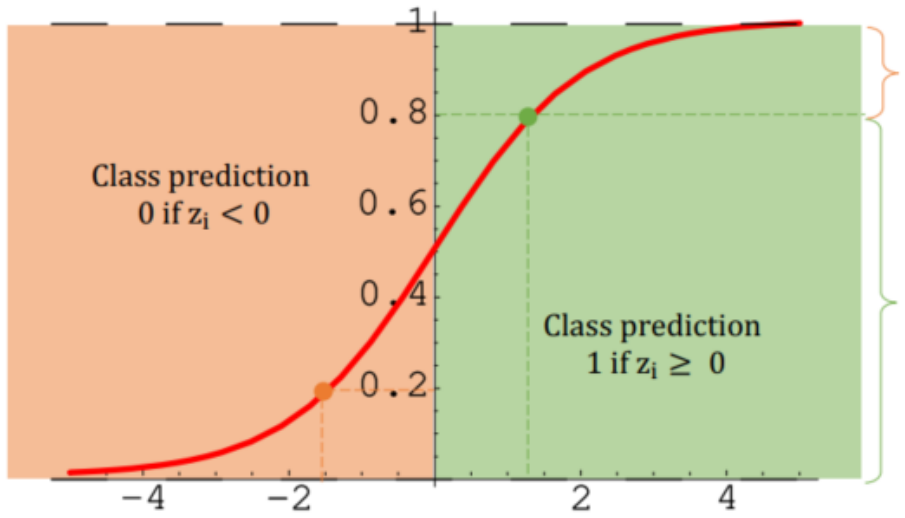
Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

Cross-entropy loss, also known as negative log-likelihood loss, measures the dissimilarity between the predicted probability distribution and the true distribution by computing the negative log-likelihood of the true distribution.
Cross-entropy can be used only with an activation function which represent a probability distribution like sigmoid, softmax.

$$\textbf{Cross-entropy loss}: \; l = \frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i \, log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i)$$

$$\hat{y}_i = s(z_i) = 1/(1 + e^{-z_i})$$



$$P(Y = 0) = 1 - s(z_i)$$

$$P(Y = 1) = s(z_i)$$

Bernoulli distribution:

$$P(Y = y_i) = s(z_i)^{y_i} * (1 - s(z_i)^{1-y_i})$$

max Likelihood h:

$$h = \prod_{i=1}^{N} P(Y = y_i)$$

$$h = \prod_{i=1}^{N} s(z_i)^{y_i} * (1 - s(z_i)^{1-y_i})$$

$$z_i = X_i * W + b$$

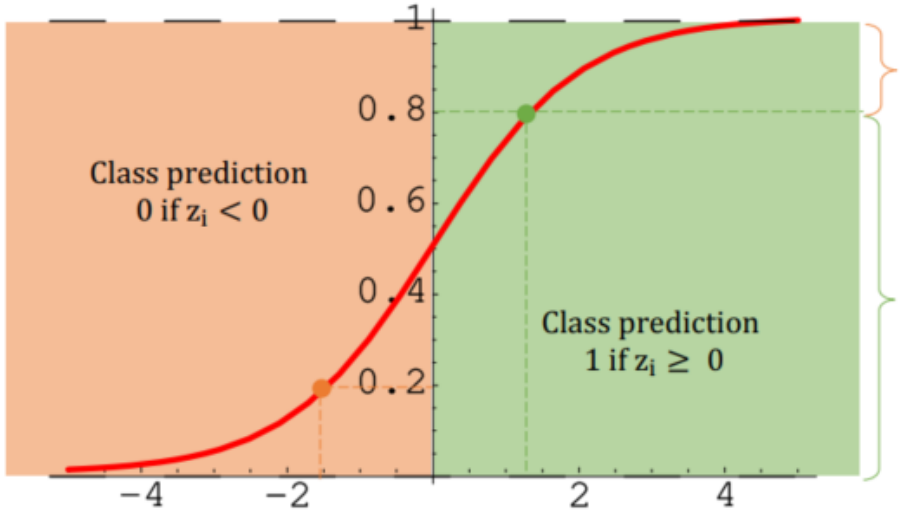
Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

Cross-entropy loss, also known as negative log-likelihood loss, measures the dissimilarity between the predicted probability distribution and the true distribution by computing the negative log-likelihood of the true distribution.
Cross-entropy can be used only with an activation function which represent a probability distribution like sigmoid, softmax.

$$\boldsymbol{Cross-entropy\ loss:}\ \boldsymbol{l = \frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i\, log(\hat{y}_i) - (1-y_i) log(1-\hat{y}_i)}$$

$$\hat{y}_i = s(z_i) = 1/(1+e^{-z_i})$$



$$P(Y=0) = 1 - s(z_i)$$

$$P(Y=1) = s(z_i)$$

$$z_i = X_i * W + b$$

Bernoulli distribution:

$$P(Y=y_i) = s(z_i)^{y_i} * (1 - s(z_i)^{1-y_i})$$

max Log-Likelihood $lh = \log(h)$:

$$lh = \log\left(\prod_{i=1}^{N} s(z_i)^{y_i} * (1 - s(z_i)^{1-y_i})\right)$$

$$lh = \sum_{i=1}^{N} \log\left(s(z_i)^{y_i} * (1 - s(z_i)^{1-y_i})\right)$$

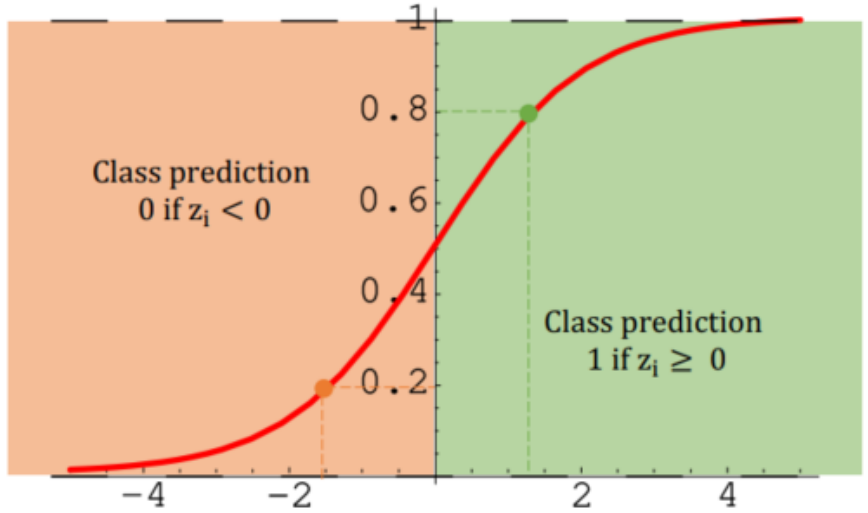
$$lh = \sum_{i=1}^{N} \boldsymbol{y_i\, log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)}$$

Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

$$Cross-entropy\ loss:\ l=\frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i\, log(\hat{y}_i) - (1-y_i) log(1-\hat{y}_i)$$

$$\hat{y}_i = s(z_i) = 1/(1+e^{-z_i})$$



$$z_i = X_i * W + b$$

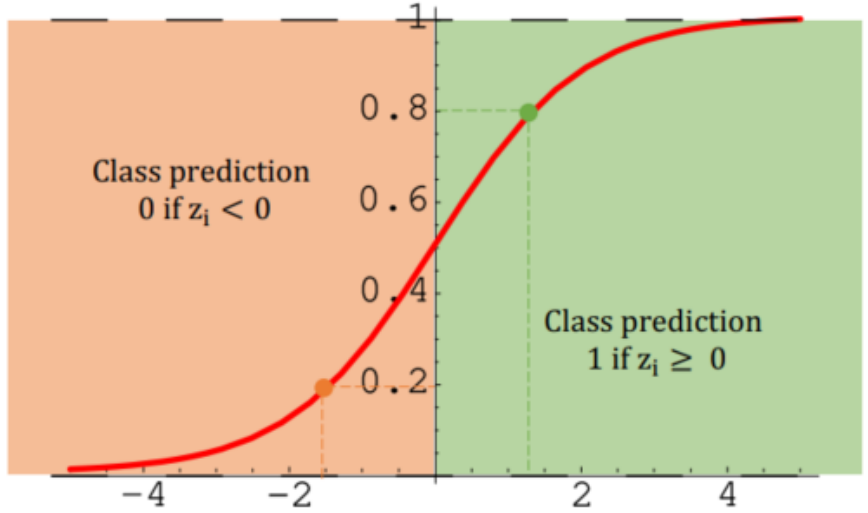
Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

$$Cross-entropy\ loss:\ l=\frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i\,log(\hat{y}_i)-(1-y_i)log(1-\hat{y}_i)$$

$$\hat{y}_i = s(z_i) = 1/(1+e^{-z_i})$$

$\frac{\partial l}{\partial \omega_k}$	$\frac{1}{N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$	
For	$y_i = 1$	
l_i	$-log(\hat{y}_i)$	
$\frac{\partial l_i}{\partial \hat{y}_i}$	$-\frac{1}{\hat{y}_i}$	
$\frac{\partial \hat{y}_i}{\partial z_i}$	$\hat{y}_i(1-\hat{y}_i)$	
$\frac{\partial z_i}{\partial \omega_k}$	x_{ik}	
$\frac{\partial l}{\partial \omega_k}$	$-\frac{1}{N}\sum_{i=1}^{N}(1-\hat{y}_i)x_{ik}$	
$\frac{\partial l}{\partial \omega_k}$		



$$z_i = X_i * W + b$$

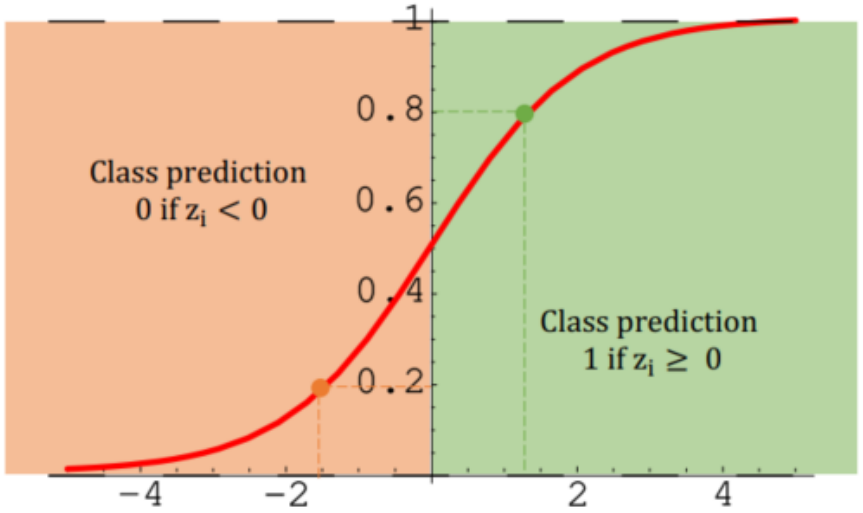
Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

$$Cross-entropy\ loss:\ l=\frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i\, log(\hat{y}_i) - (1-y_i) log(1-\hat{y}_i)$$

$$\hat{y}_i = s(z_i) = 1/(1+e^{-z_i})$$

$\frac{\partial l}{\partial \omega_k}$	$\frac{1}{N}\sum_{i=1}^{N} \frac{\partial l_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$	
For	$y_i = 1$	$y_i = 0$
l_i	$-log(\hat{y}_i)$	$-log(1-\hat{y}_i)$
$\frac{\partial l_i}{\partial \hat{y}_i}$	$-\frac{1}{\hat{y}_i}$	$\frac{1}{1-\hat{y}_i}$
$\frac{\partial \hat{y}_i}{\partial z_i}$	$\hat{y}_i(1-\hat{y}_i)$	
$\frac{\partial z_i}{\partial \omega_k}$	x_{ik}	
$\frac{\partial l}{\partial \omega_k}$	$-\frac{1}{N}\sum_{i=1}^{N}(1-\hat{y}_i)x_{ik}$	$-\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i)x_{ik}$
$\frac{\partial l}{\partial \omega_k}$	$-\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)x_{ik}$	



$$z_i = X_i * W + b$$

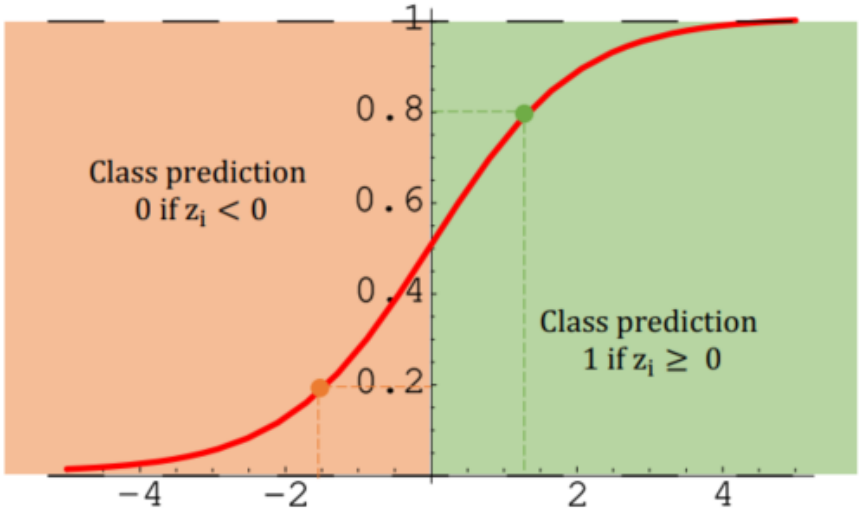
Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

$$Cross-entropy\ loss:\ l=\frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i\, log(\hat{y}_i) - (1-y_i) log(1-\hat{y}_i)$$

$$\hat{y}_i = s(z_i) = 1/(1+e^{-z_i})$$

$\frac{\partial l}{\partial \omega_k}$	$\frac{1}{N}\sum_{i=1}^{N} \frac{\partial l_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$	
For	$y_i = 1$	$y_i = 0$
l_i	$-log(\hat{y}_i)$	$-log(1-\hat{y}_i)$
$\frac{\partial l_i}{\partial \hat{y}_i}$	$-\frac{1}{\hat{y}_i}$	$\frac{1}{1-\hat{y}_i}$
$\frac{\partial \hat{y}_i}{\partial z_i}$	$\hat{y}_i(1-\hat{y}_i)$	
$\frac{\partial z_i}{\partial \omega_k}$	x_{ik}	
$\frac{\partial l}{\partial \omega_k}$	$-\frac{1}{N}\sum_{i=1}^{N}(1-\hat{y}_i)x_{ik}$	$-\frac{1}{N}\sum_{i=1}^{N}(0-\hat{y}_i)x_{ik}$
$\frac{\partial l}{\partial \omega_k}$	$-\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)x_{ik}$	



$$z_i = X_i * W + b$$

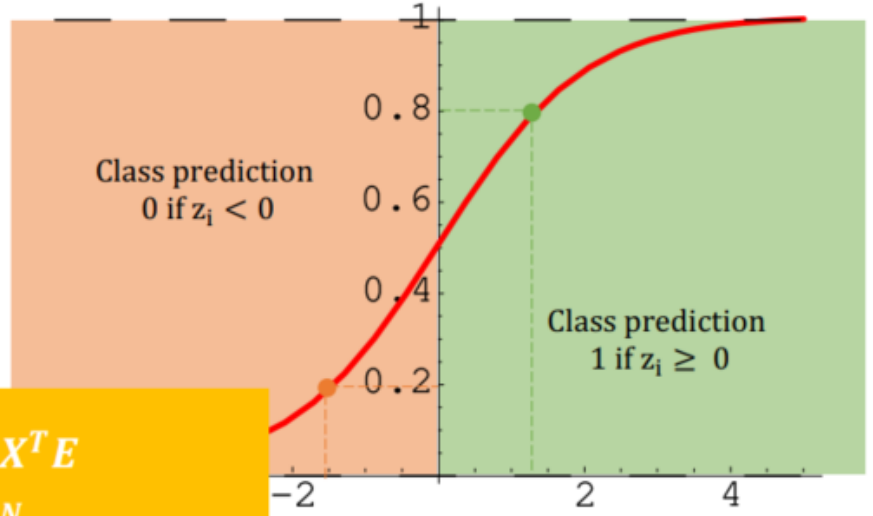
Artificial neural networks (ANNs)

PERCEPTRON >> Gradient descent with Cross entropy loss function classification (0,1)

$$Cross-entropy\ loss:\ l=\frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} -y_i\,log(\hat{y}_i)-(1-y_i)log(1-\hat{y}_i)$$

$$\hat{y}_i = s(z_i) = 1/(1+e^{-z_i})$$

$\frac{\partial l}{\partial \omega_k}$	$\frac{1}{N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i}\frac{\partial z_i}{\partial \omega_k}$	
For	$y_i = 1$	$y_i = 0$
l_i	$-log(\hat{y}_i)$	$-log(1-\hat{y}_i)$
$\frac{\partial l_i}{\partial \hat{y}_i}$	$-\frac{1}{\hat{y}_i}$	$\frac{1}{1-\hat{y}_i}$
$\frac{\partial \hat{y}_i}{\partial z_i}$	$\hat{y}_i(1-\hat{y}_i)$	
$\frac{\partial z_i}{\partial \omega_k}$	x_{ik}	
$\frac{\partial l}{\partial \omega_k}$	$-\frac{1}{N}\sum_{i=1}^{N}(1-\hat{y}_i)x_{ik}$	$-\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i)x_{ik}$
$\frac{\partial l}{\partial \omega_k}$	$-\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)x_{ik}$	



$$z_i = X_i * W + b$$

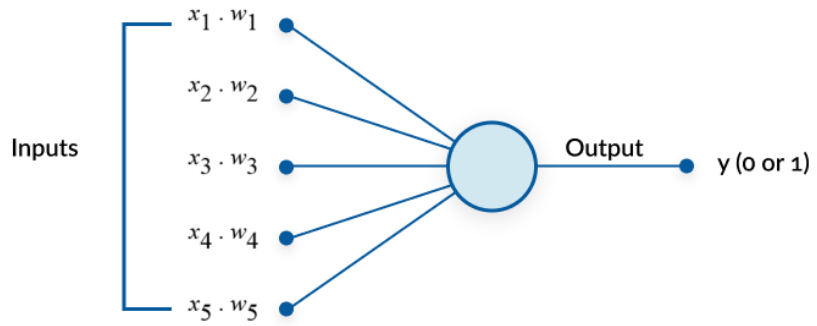
$$\Delta W = -\frac{1}{N}X^T E$$

$$\Delta b = -\frac{1}{N}\sum_{i=1}^{N} e_i$$

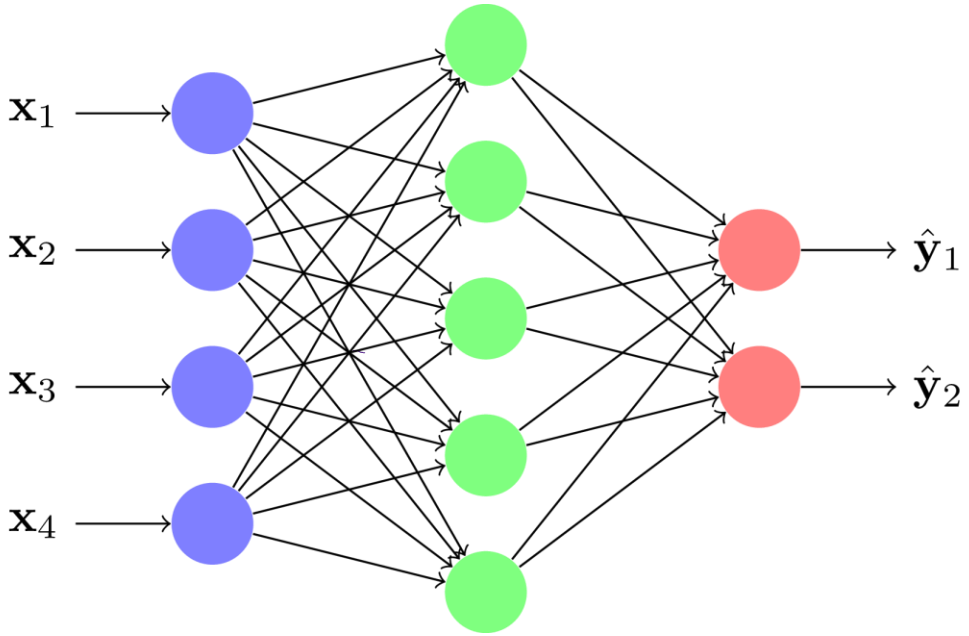
$$with$$

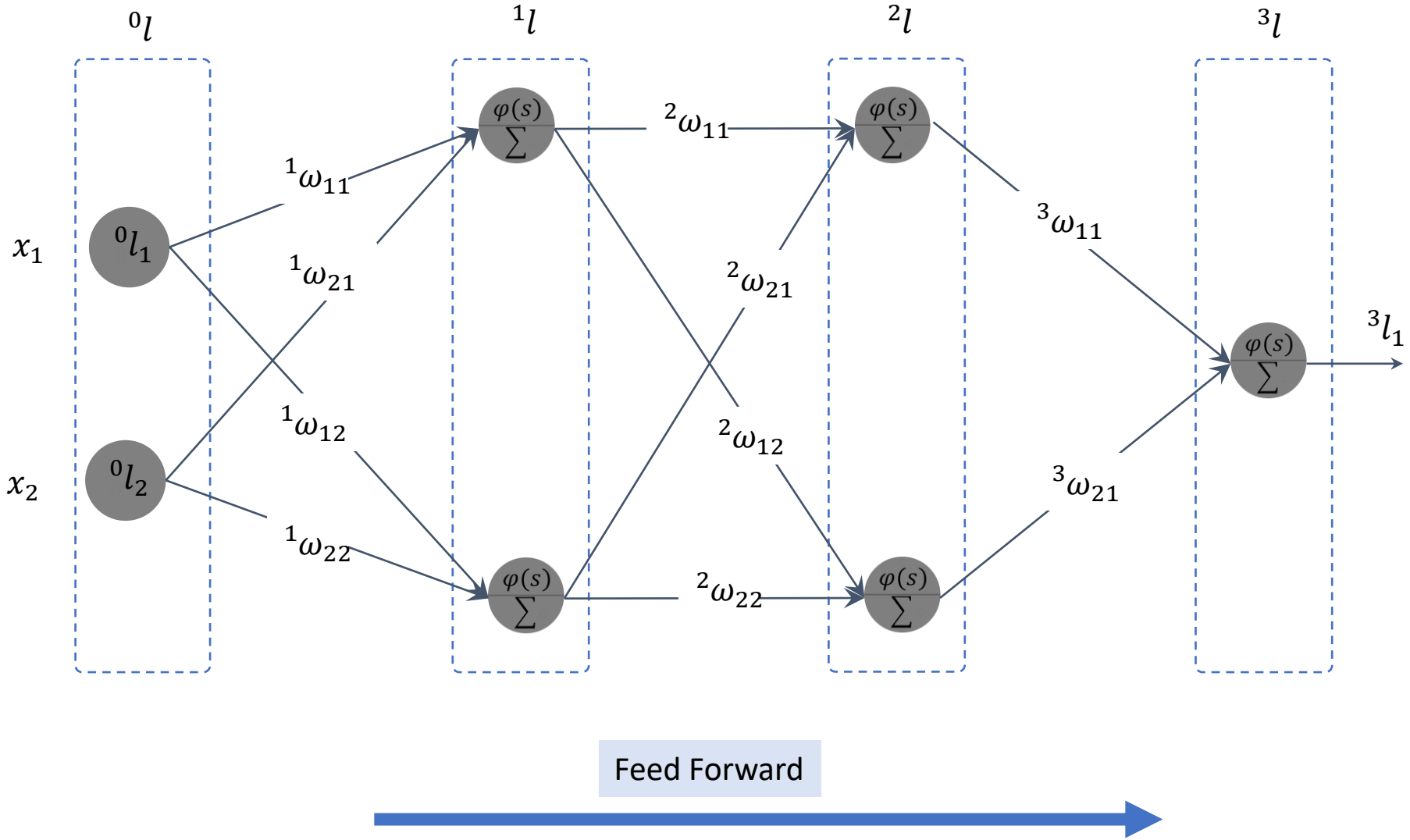
$$e_i = y_i - \hat{y}_i$$

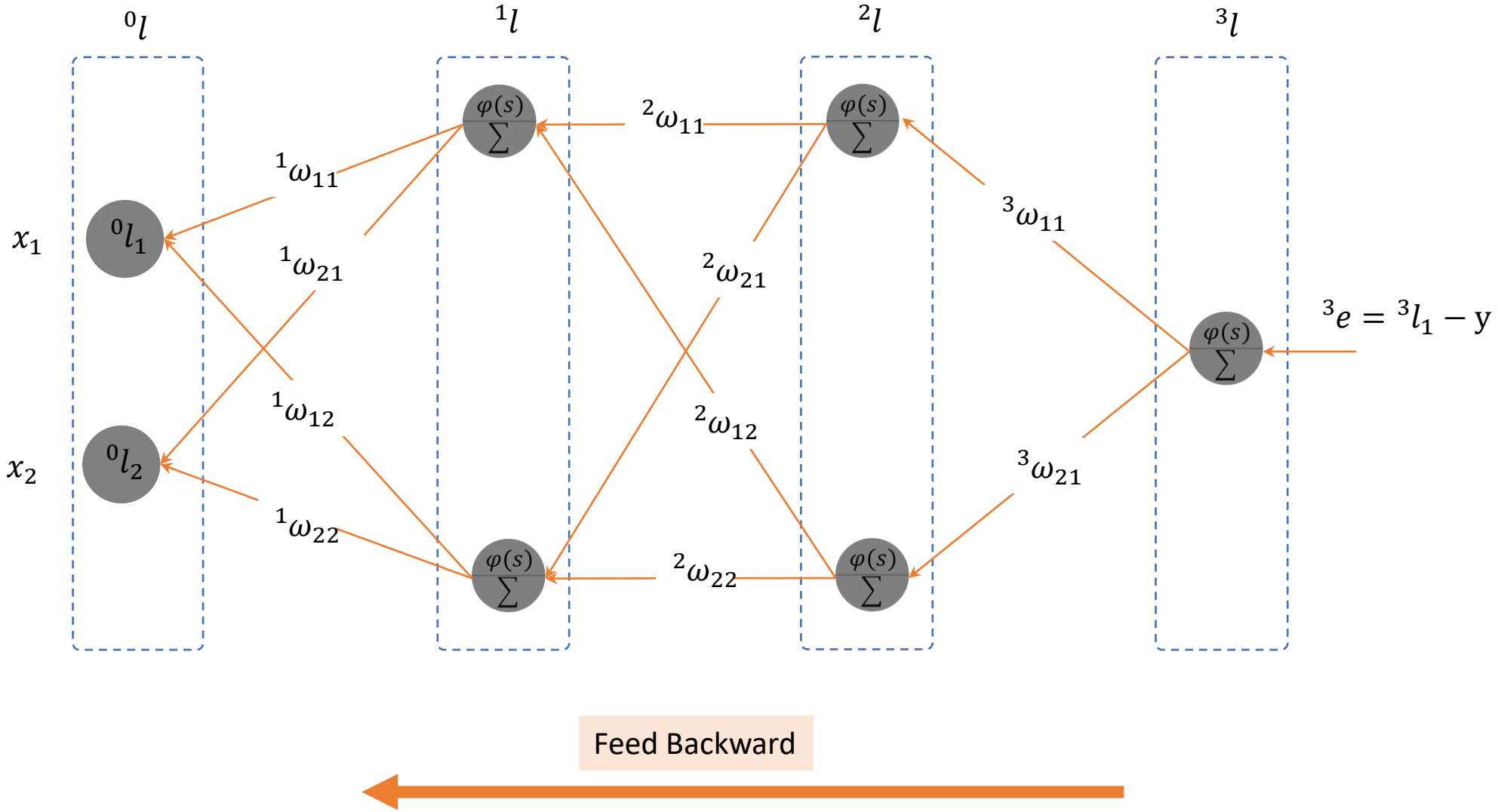
perceptron

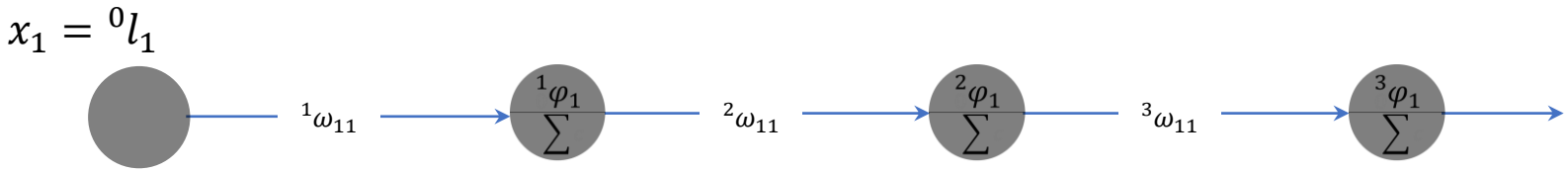


perceptron multicouche
MultiLayer Perceptron (MLP)

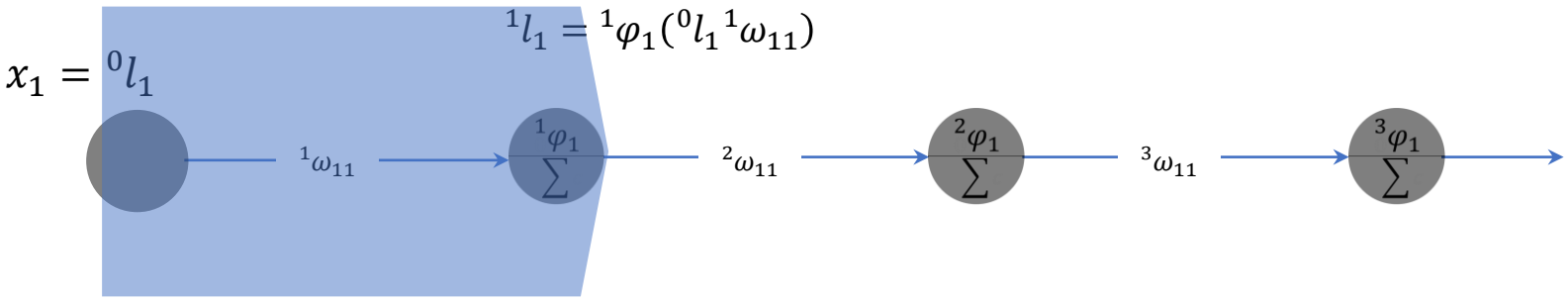



${}^0l$
${}^1l$
${}^2l$
${}^3l$
x_1
x_2
${}^0l_1$
${}^0l_2$
$\varphi(s)$
$\sum$
${}^1\omega_{11}$
${}^1\omega_{21}$
${}^1\omega_{12}$
${}^1\omega_{22}$
${}^2\omega_{11}$
${}^2\omega_{21}$
${}^2\omega_{12}$
${}^2\omega_{22}$
${}^3\omega_{11}$
${}^3\omega_{21}$
${}^3l_1$
Feed Forward

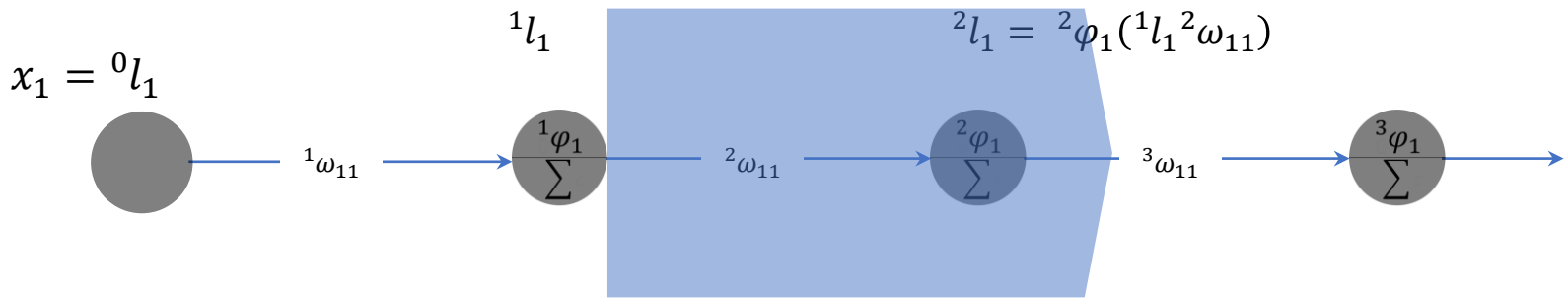

${}^0l$
${}^1l$
${}^2l$
${}^3l$
x_1
x_2
${}^0l_1$
${}^0l_2$
$\varphi(s)$
$\sum$
${}^1\omega_{11}$
${}^1\omega_{21}$
${}^1\omega_{12}$
${}^1\omega_{22}$
${}^2\omega_{11}$
${}^2\omega_{21}$
${}^2\omega_{12}$
${}^2\omega_{22}$
${}^3\omega_{11}$
${}^3\omega_{21}$
${}^3e = {}^3l_1 - \mathrm{y}$
Feed Backward



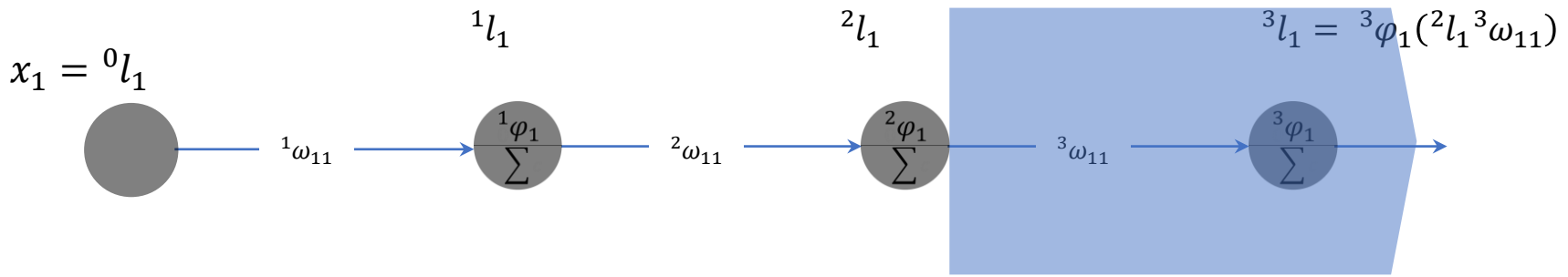
Feed Forward



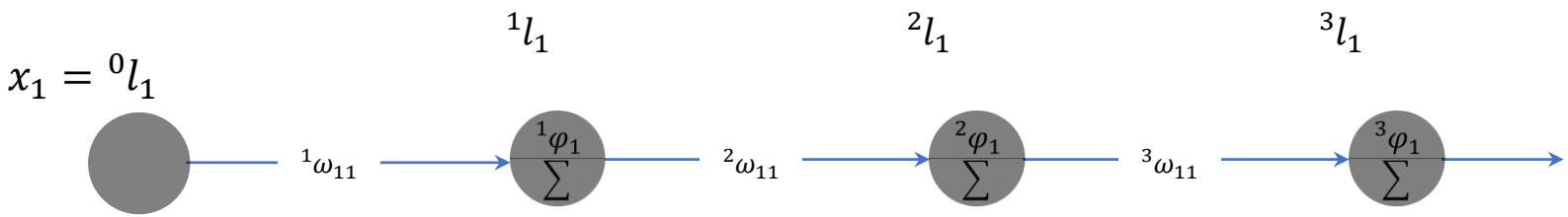
Feed Forward



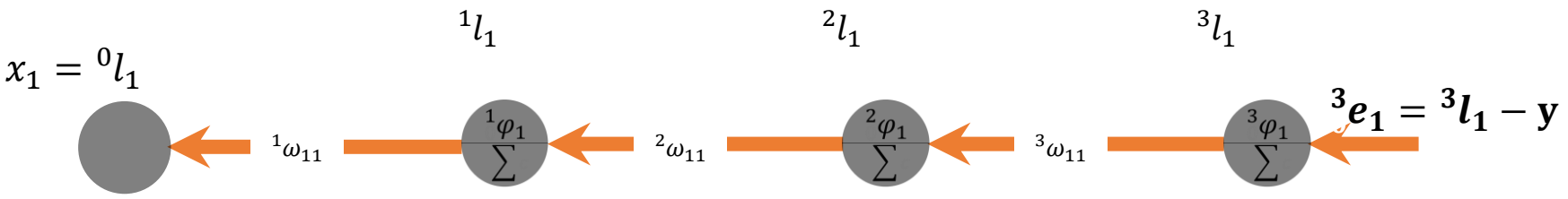
Feed Forward



Feed Forward

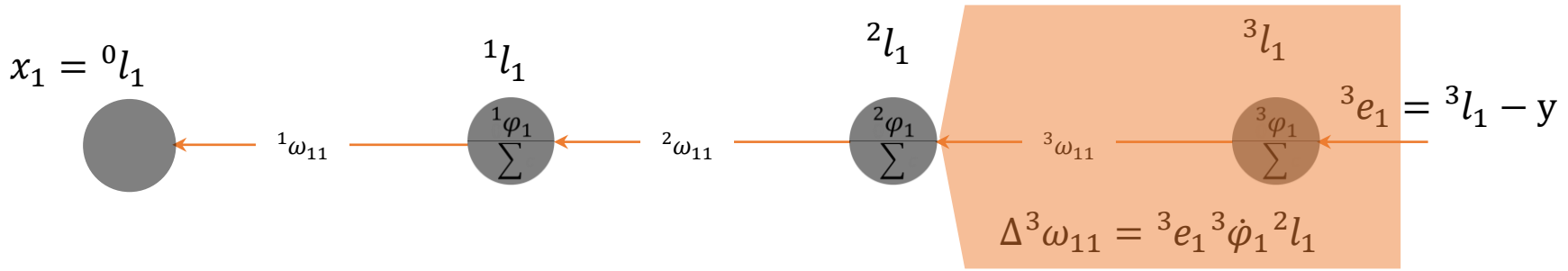


Feed Forward



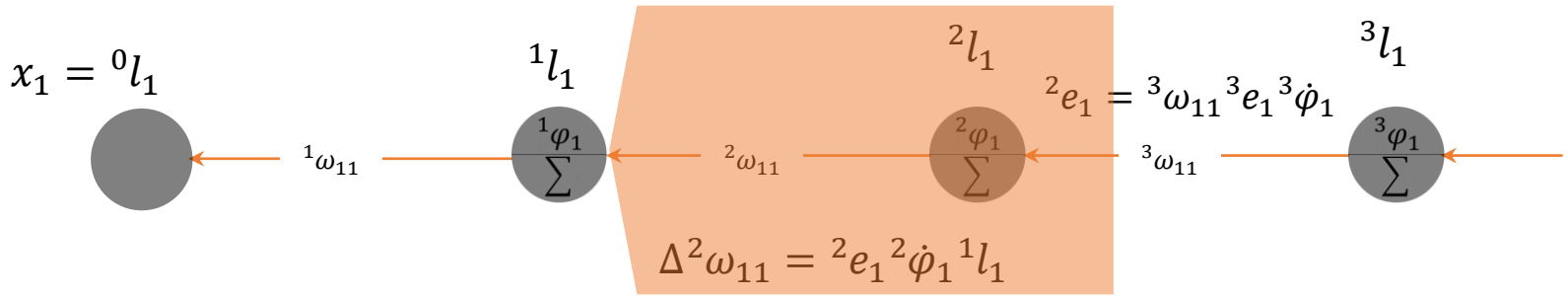
Feed Backward

Gradient Descent algorithm



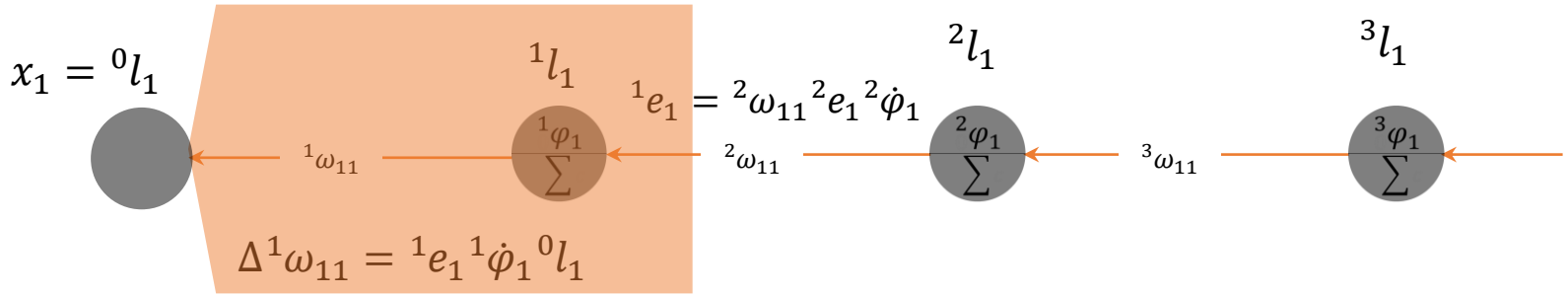
Feed Backward

Gradient Descent algorithm



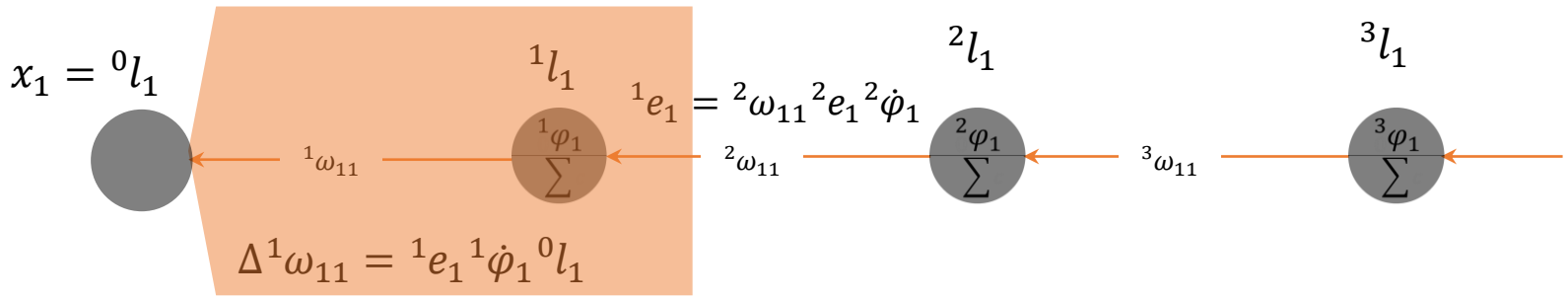
Feed Backward

Gradient Descent algorithm



Feed Backward

Gradient Descent algorithm



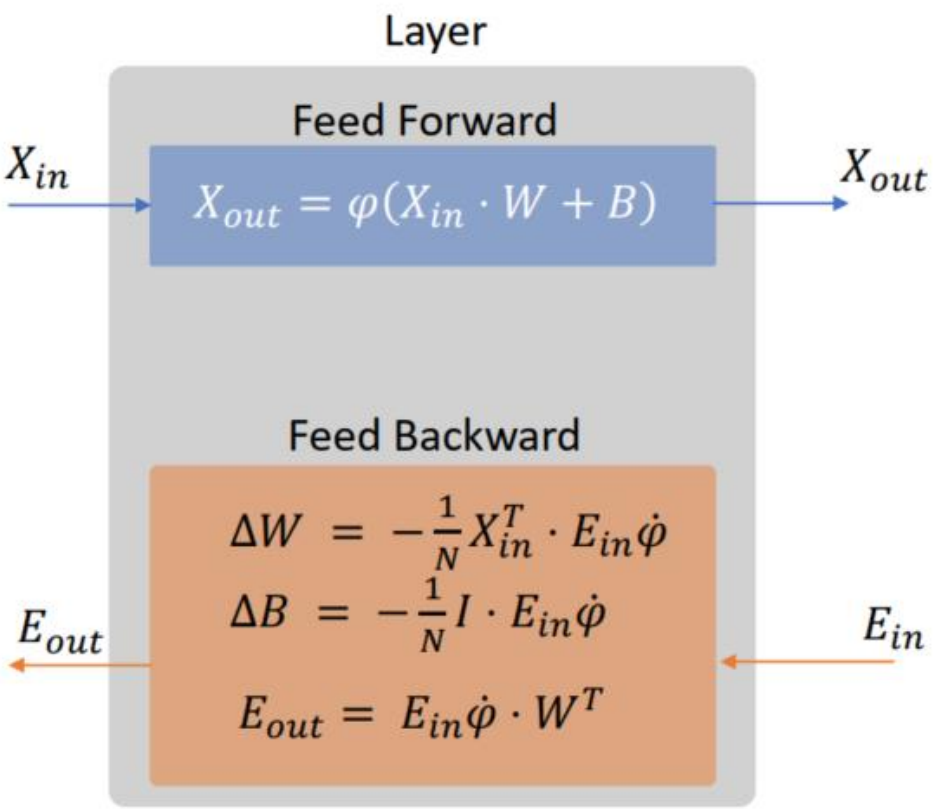
Feed Backward

Gradient Descent algorithm

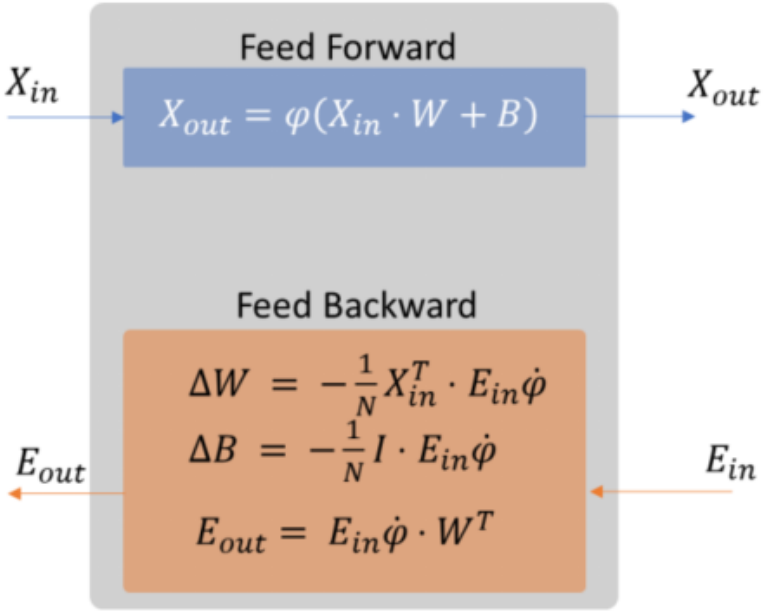
N.B. « · » (\cdot): produit matriciel

0	Initialize all weights with random values between [-1,1]		
1	${}^0L = X$	(n_s, n_0)	**Propagation** of the input data.
2	${}^1L = {}^1\phi({}^0L \cdot {}^1W)$	$(n_s, n_1) = (n_s, n_0) \cdot (n_0, n_1)$	
3	${}^2L = {}^2\phi({}^1L \cdot {}^2W)$	$(n_s, n_2) = (n_s, n_1) \cdot (n_1, n_2)$	
4	${}^2E = {}^2L - Y$	$(n_s, n_2) = (n_s, n_2) - (n_s, n_2)$	**Backpropagation** of the output error. Layer 2 weight adjustment
5	$\Delta^2W = {}^1L^T \cdot {}^2E\,{}^2\dot{\phi}$	$(n_1, n_2) = (n_1, n_s) \cdot (n_s, n_2)$	
6	${}^2W = {}^2W - \frac{\eta}{n_s}\Delta^2W$	(n_1, n_2)	
7	${}^1E = {}^2E\,{}^2\dot{\phi} \cdot {}^2W^T$	$(n_s, n_1) = (n_s, n_2) \cdot (n_2, n_1)$	**Backpropagation** of the output error. Layer 1 weight adjustment
8	$\Delta^1W = {}^0L^T \cdot {}^1E\,{}^1\dot{\phi}$	$(n_0, n_1) = (n_0, n_s) \cdot (n_s, n_1)$	
9	${}^1W = {}^1W - \frac{\eta}{n_s}\Delta^1W$	(n_0, n_1)	
10	repeat lines 1 through 9 until the stop condition is true.		

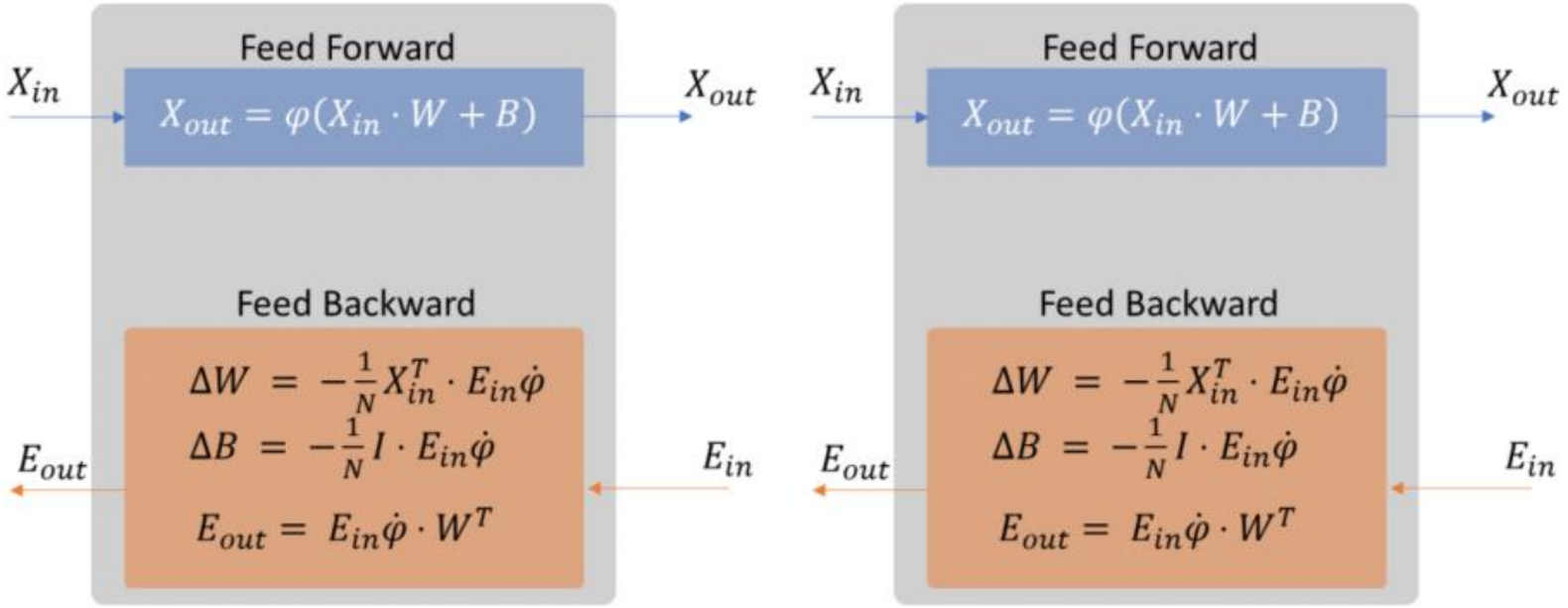
Artificial neural networks (ANNs)



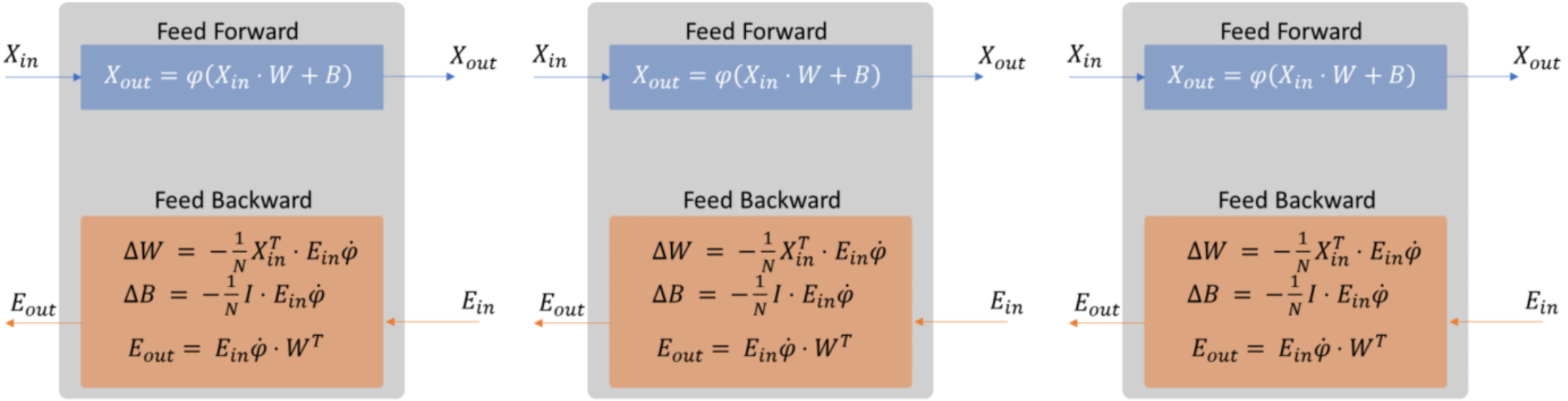
Artificial neural networks (ANNs)



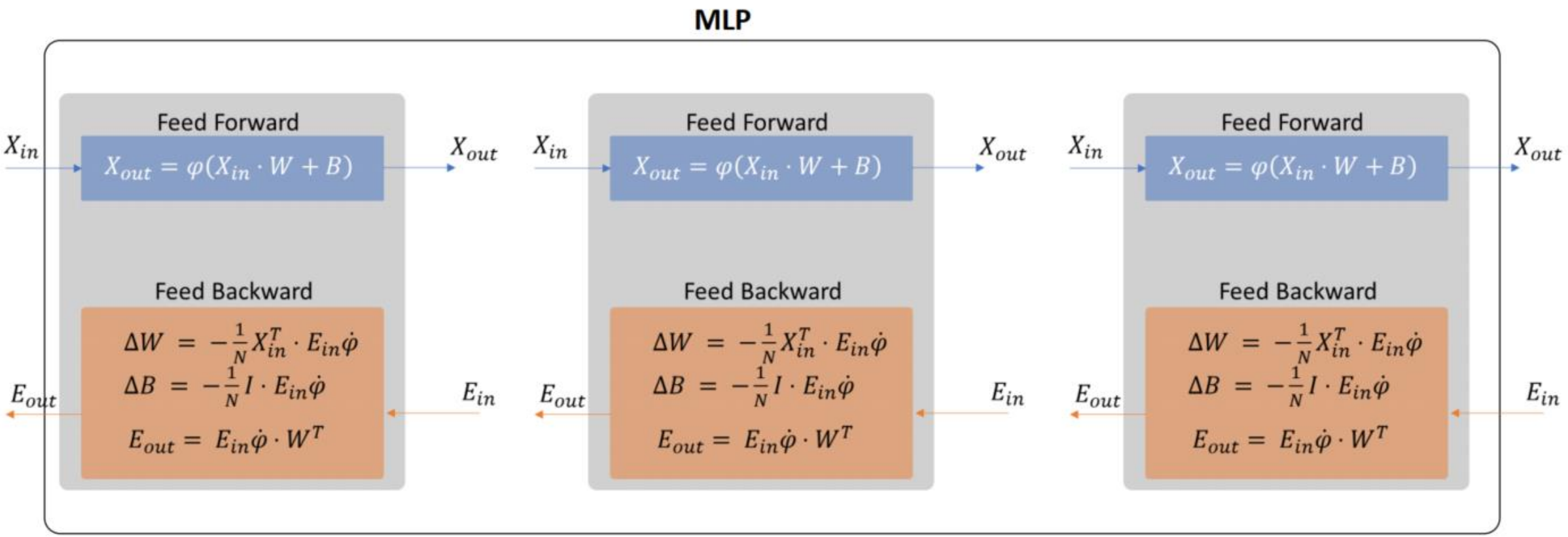
Artificial neural networks (ANNs)



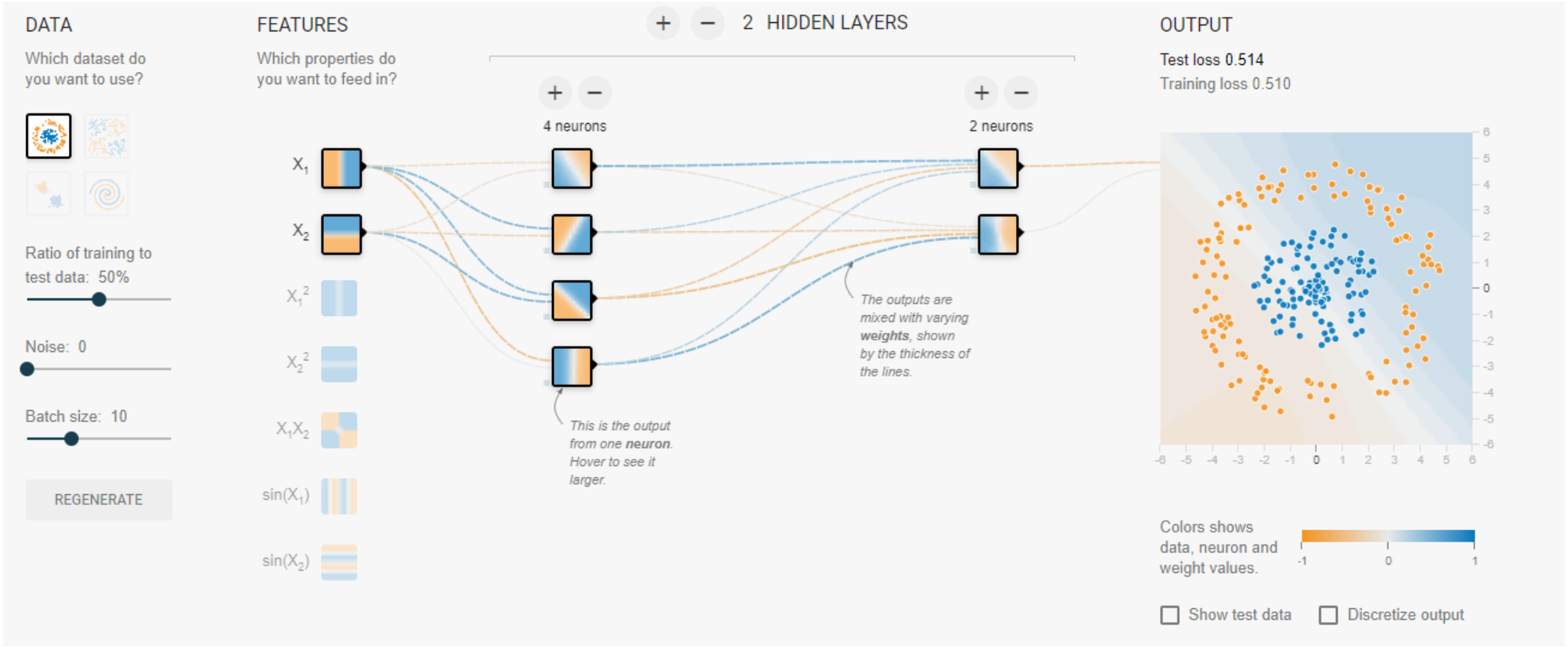
Artificial neural networks (ANNs)

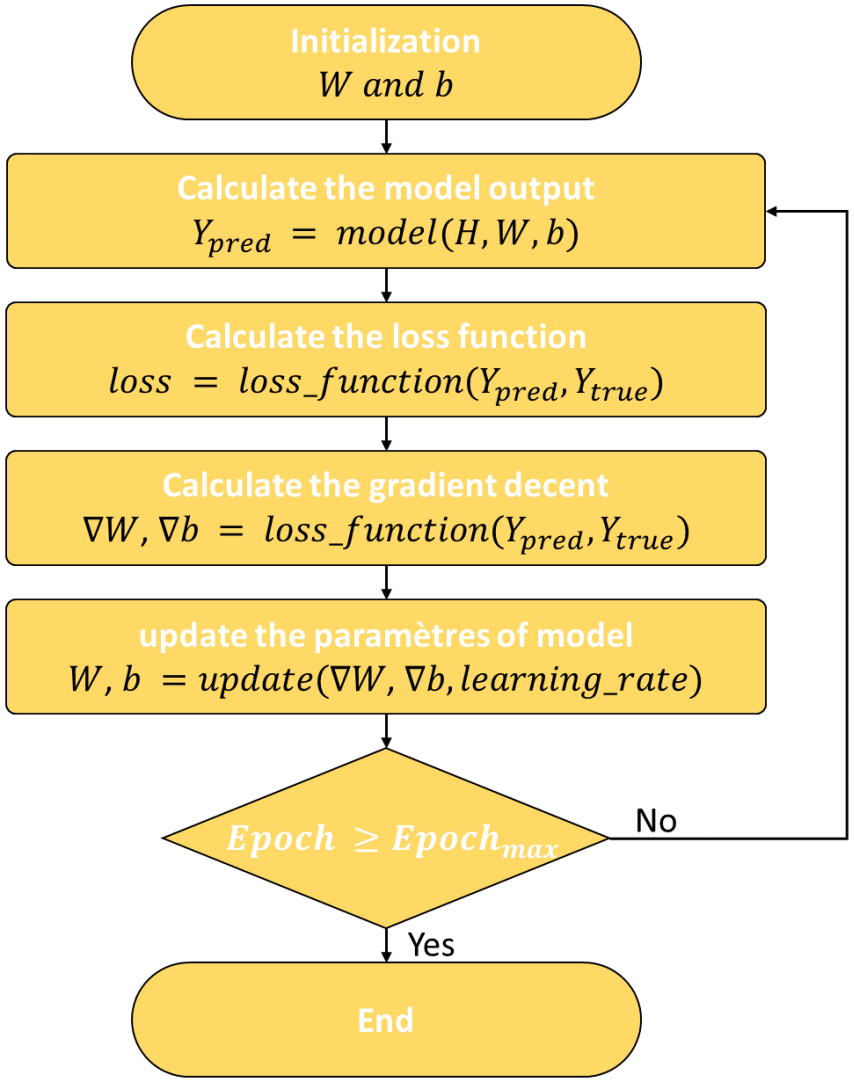


Artificial neural networks (ANNs)



Discover how the neural network works




Initialization
W and b
Calculate the model output
$Y_{pred} = model(H, W, b)$
Calculate the loss function
$loss = loss\_function(Y_{pred}, Y_{true})$
Calculate the gradient decent
$\nabla W, \nabla b = loss\_function(Y_{pred}, Y_{true})$
update the paramètres of model
$W, b = update(\nabla W, \nabla b, learning\_rate)$
$Epoch \geq Epoch_{max}$
No
Yes
End