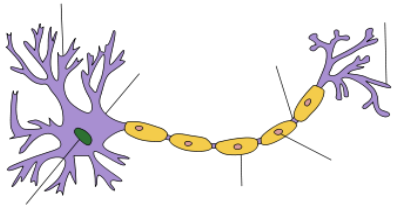
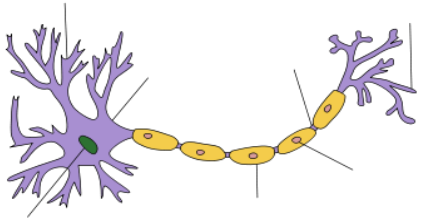
UMONS
Université de Mons


# ADVANCED DEEP LEARNING (I-ILIA-202)

# CHAPTER 01 : PERFORMANCE ANALYSIS AND OPTIMIZATION OF DEEP NEURAL NETWORKS





Sidi Ahmed Mahmoudi

POLYTECH MONS

sidi.mahmoudi@umons.ac.be

03 December 2024

# PLAN

Introduction & recall

I. Performance analysis

II. Performance optimization : regularization

- L2 & L1 regularization
- Dropout
- Data augmentation
- Early stopping

III. Batch normalization

Conclusion

# PLAN

Introduction & recall

I. Performance analysis

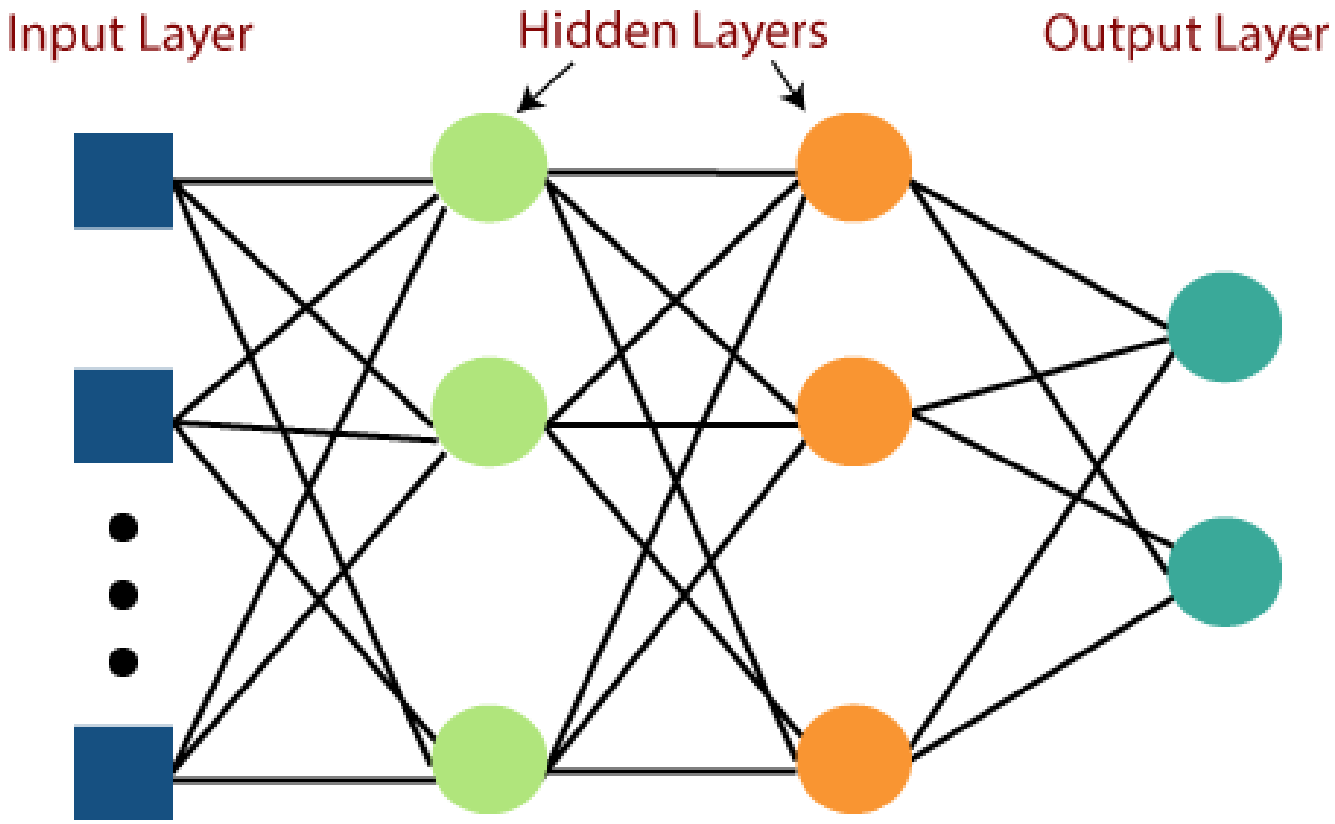
II. Performance optimization : regularization

- L2 & L1 regularization
- Dropout
- Data augmentation
- Early stopping

III. Batch normalization

Conclusion

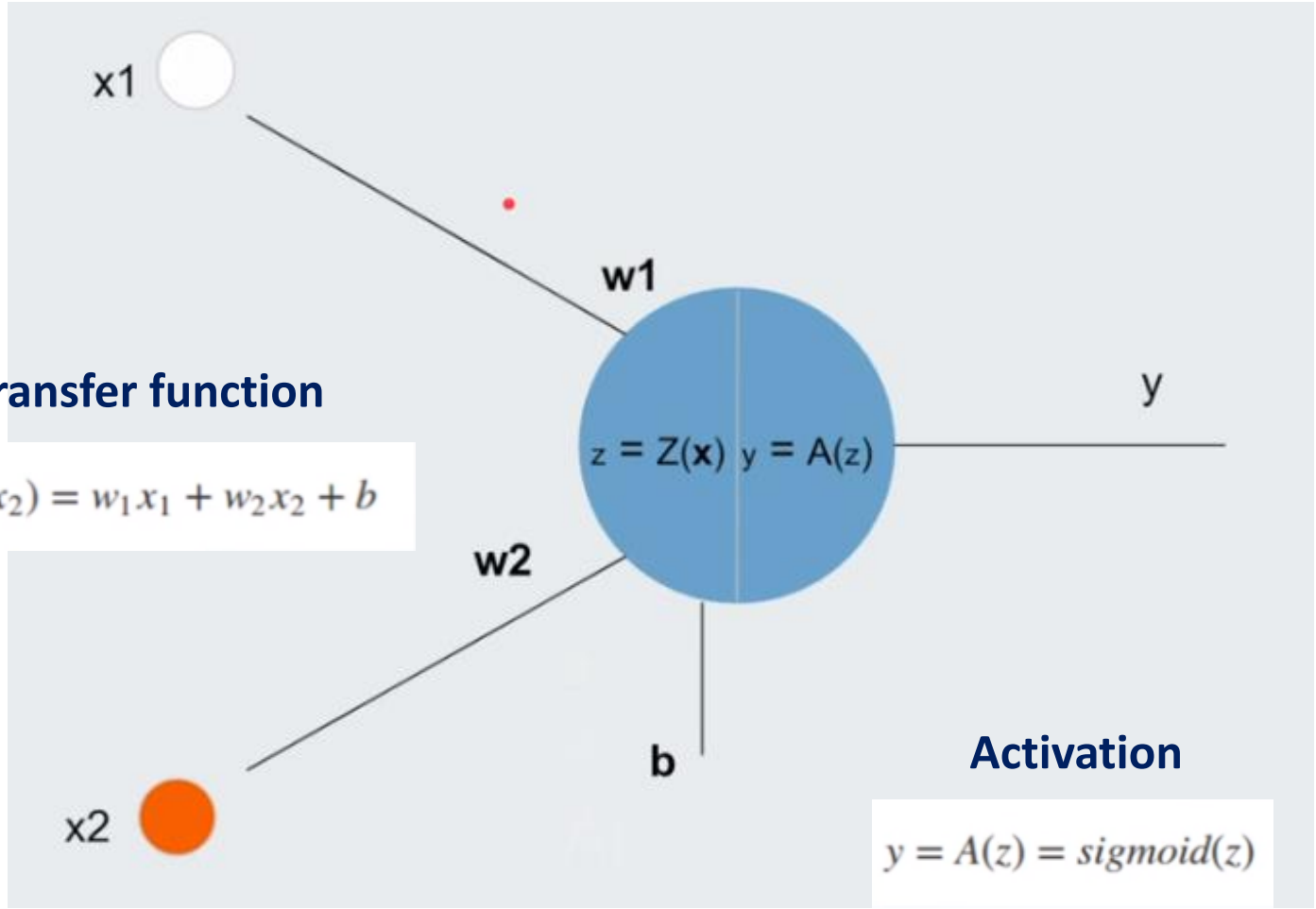
# Introduction




**Multi-Layered Perceptron (MLP)**

RECALL

# Perceptron




**Transfer function**

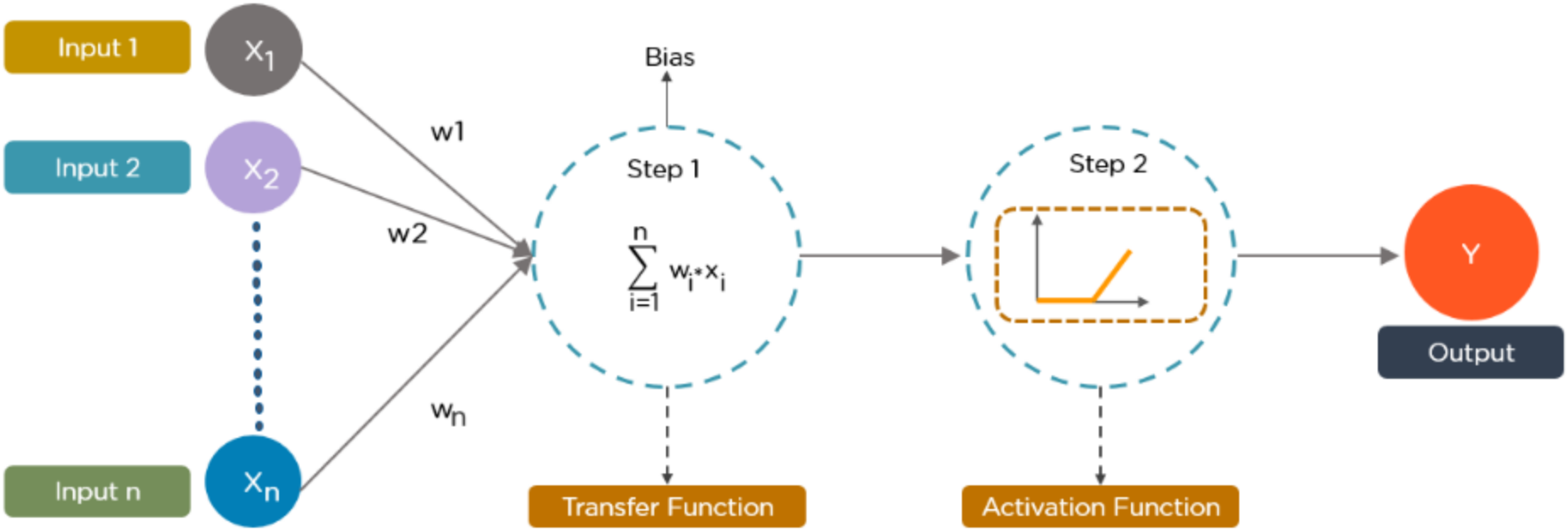
$$Z(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

**Activation**

$$y = A(z) = sigmoid(z)$$

RECALL

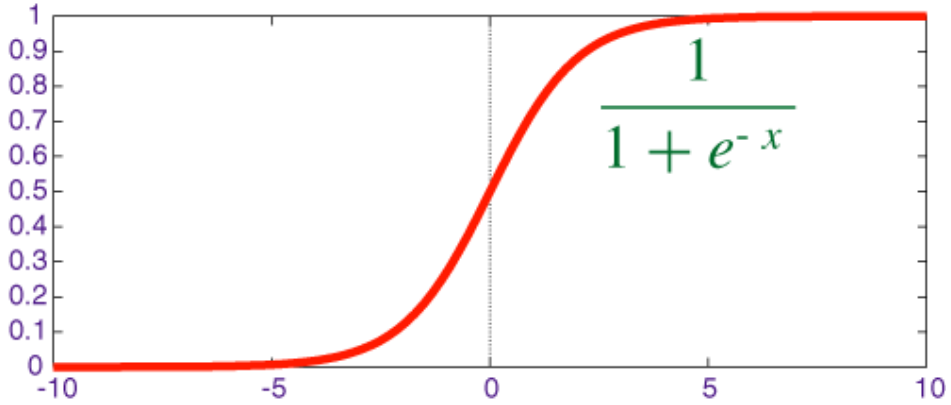
# Perceptron

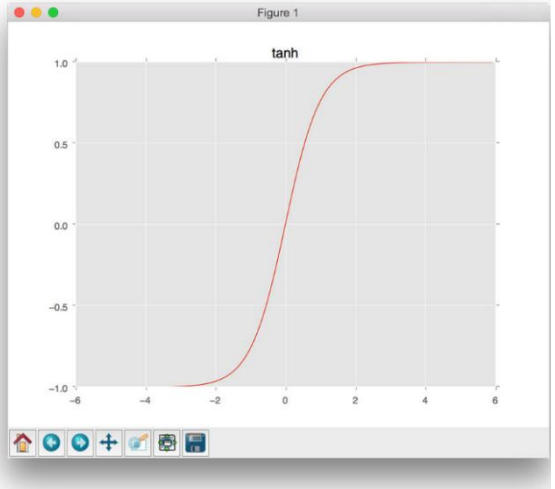


RECALL

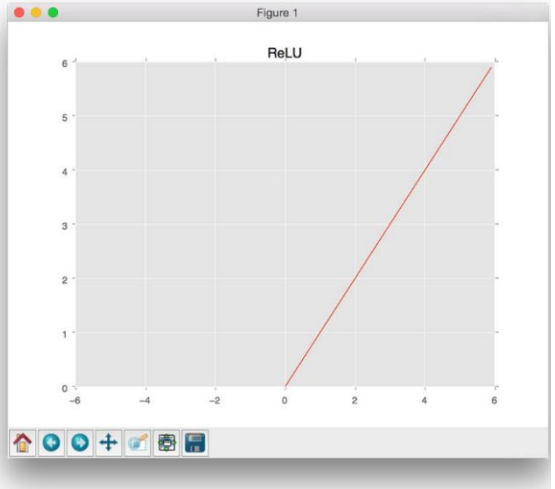
# Activation functions

**Sigmoid**

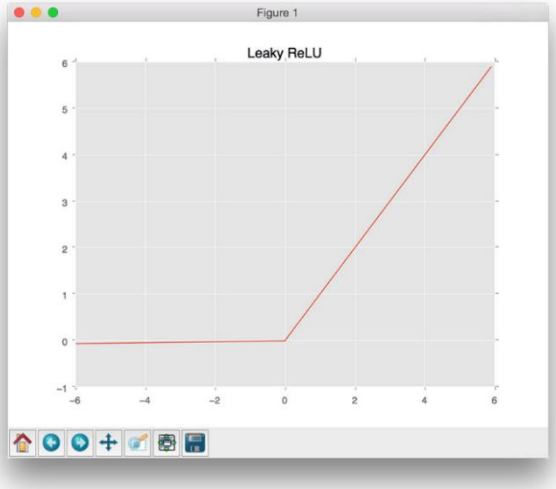



**tanh**




**Relu**




**Leaky Relu**

RECALL

# Learning : main steps

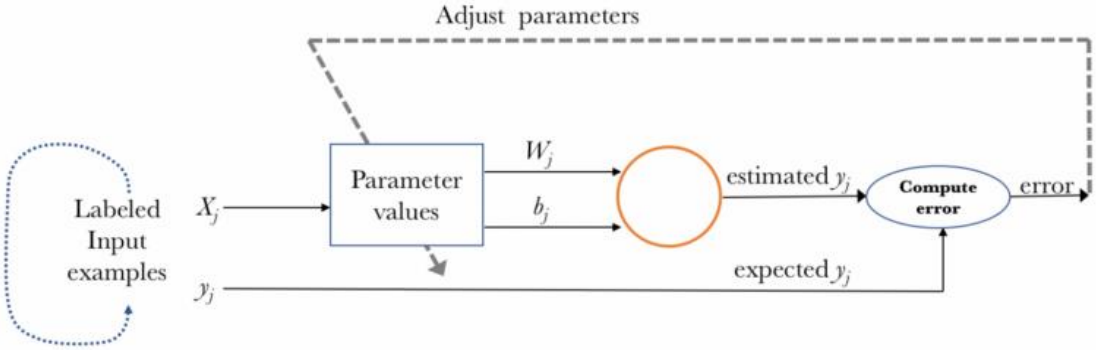
a. **Initialization :** Init weights (W) and bias (b) with random values

b. **Forward Pass :** get predictions using the proposed neural network

c. **Error calculation :** compare predicted values vs. real values

d. **Backpropagation :** update the weights using gradient descent

e. **Iterate :** Repeat the previous steps until get efficient model.



RECALL

# Learning : exemple



Glioma tumor



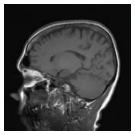
No tumor



Meningioma Tumor

Data


No tumor



Forward

Backward

$Y = f(x)$

30 % 30 % 40 %

RECALL

# Learning : exemple

Glioma tumor

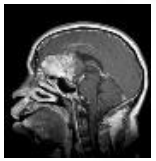
No tumor

Meningioma Tumor

Data

Training

Giloma tumor →



Forward

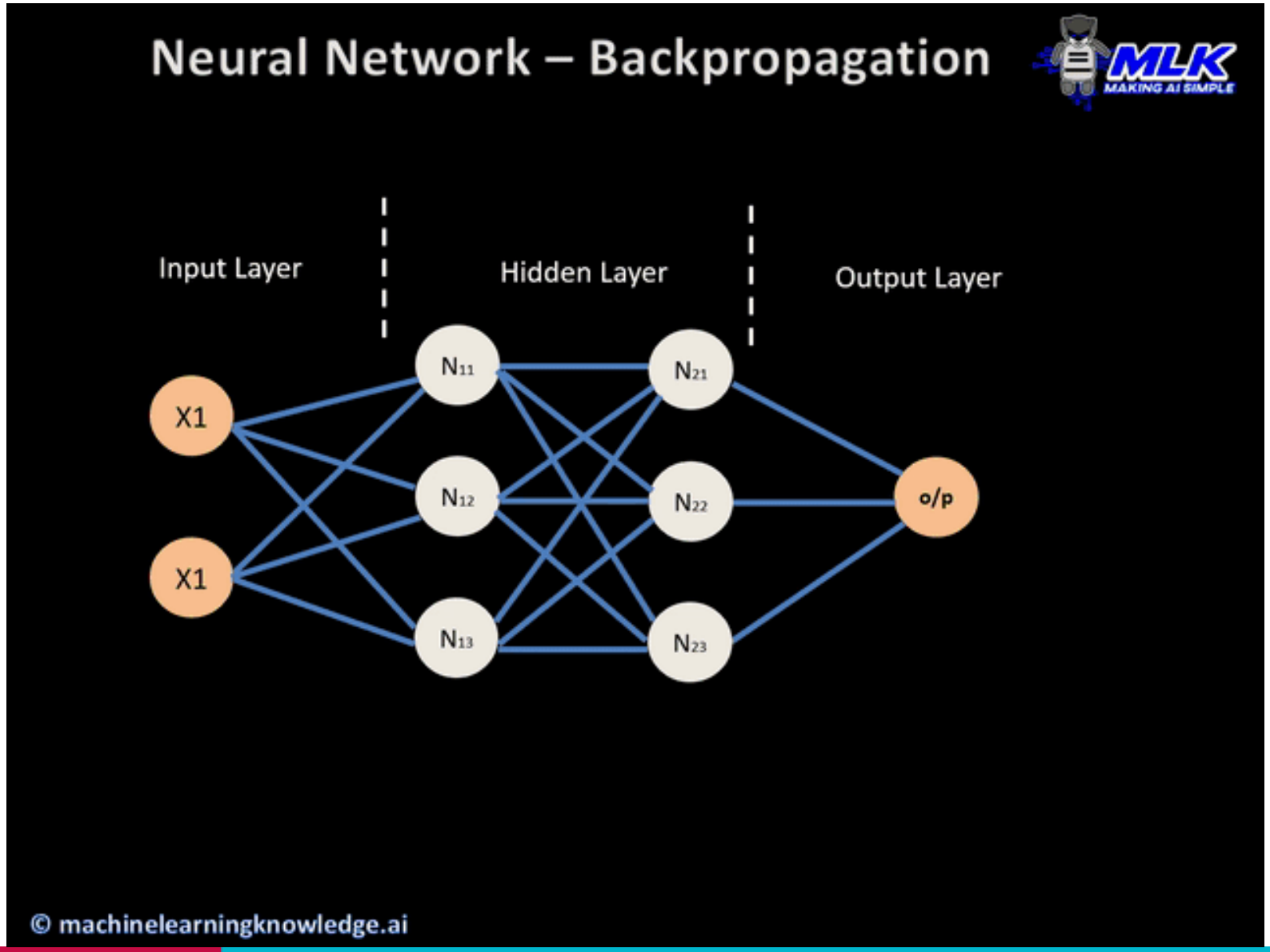
Backward

$Y = f(x)$

100 % 0 % 0 %

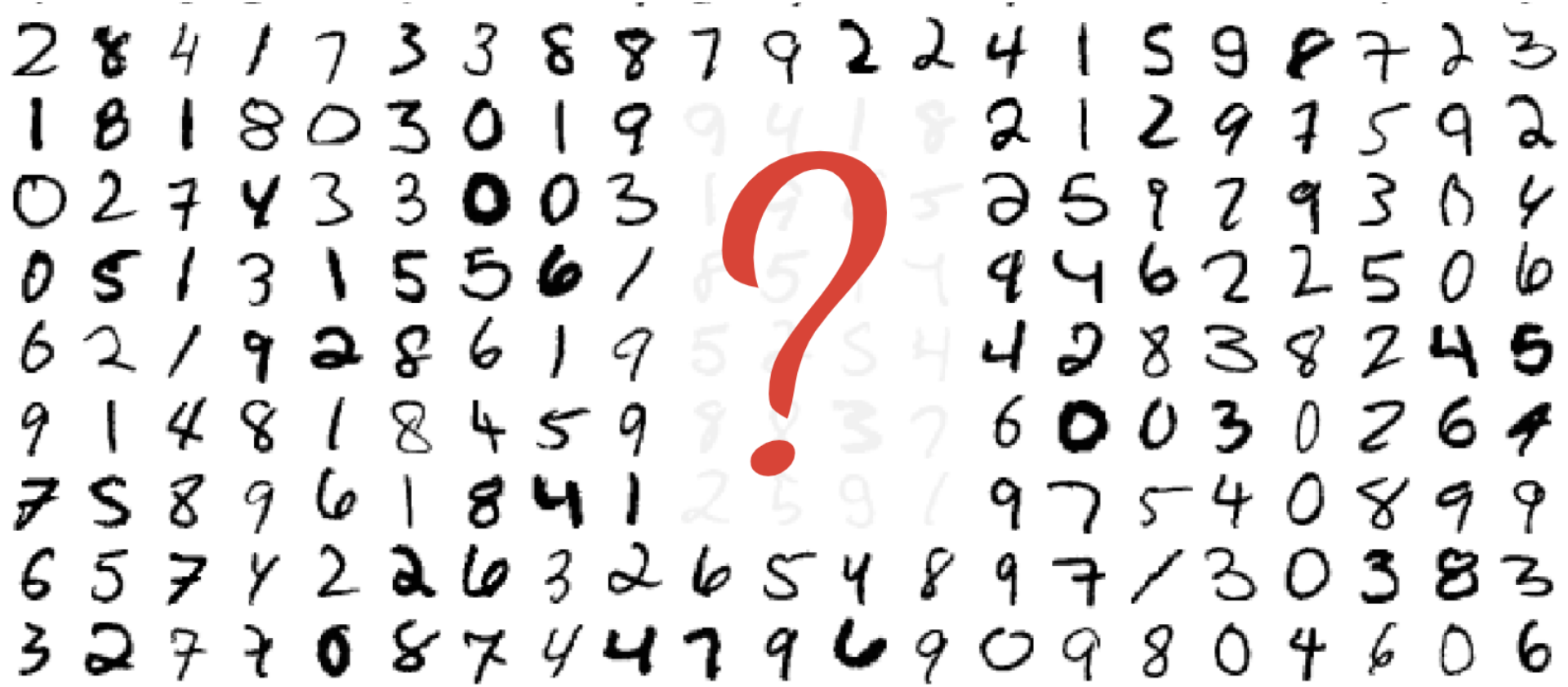
RECALL

# Backpropagation process



RECALL

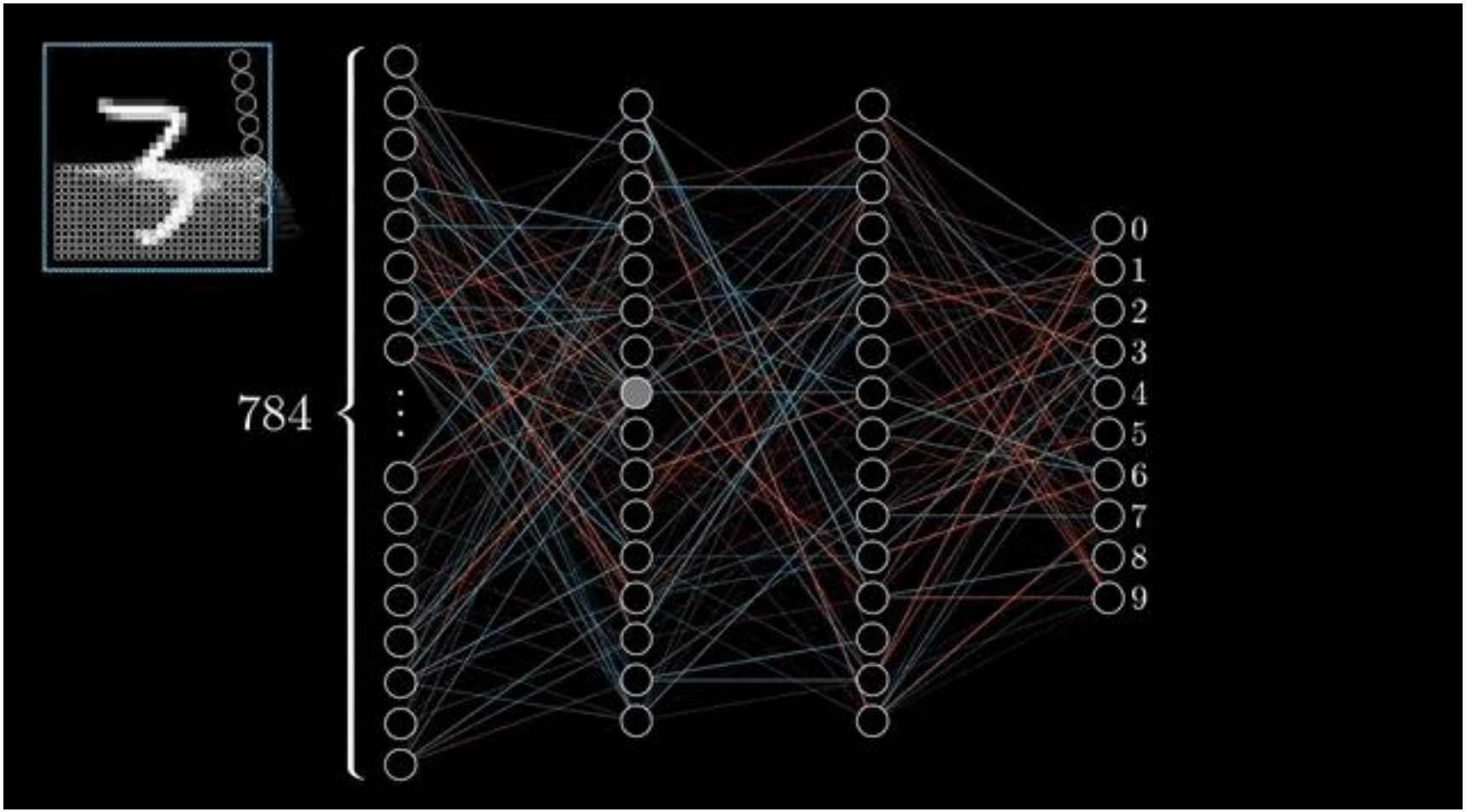
# Let’s take an example




**MNIST : Mixed National Institute of Standards and Technology [1]**

RECALL

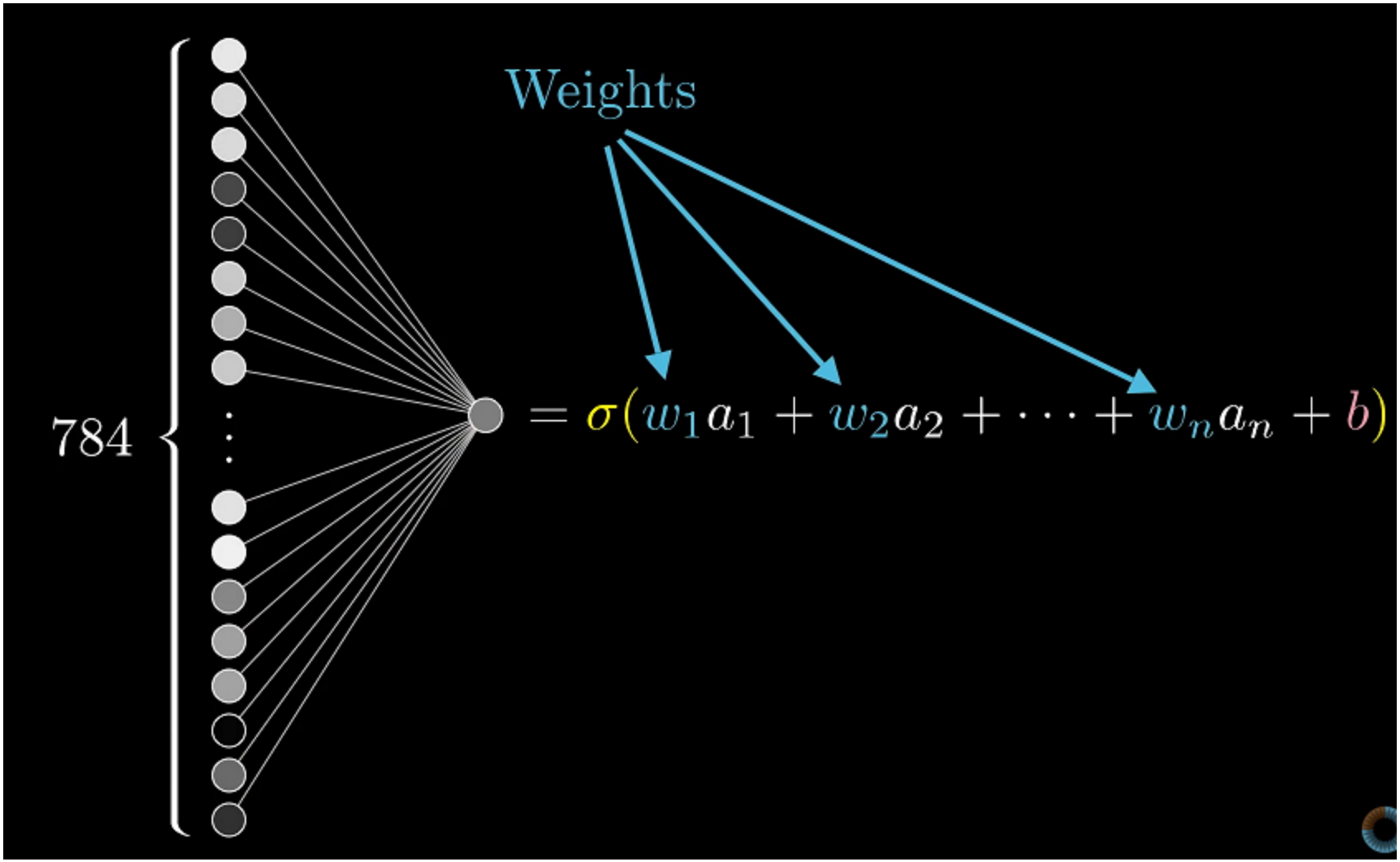
# MLP for MNIST classification




**MLP : images classification**

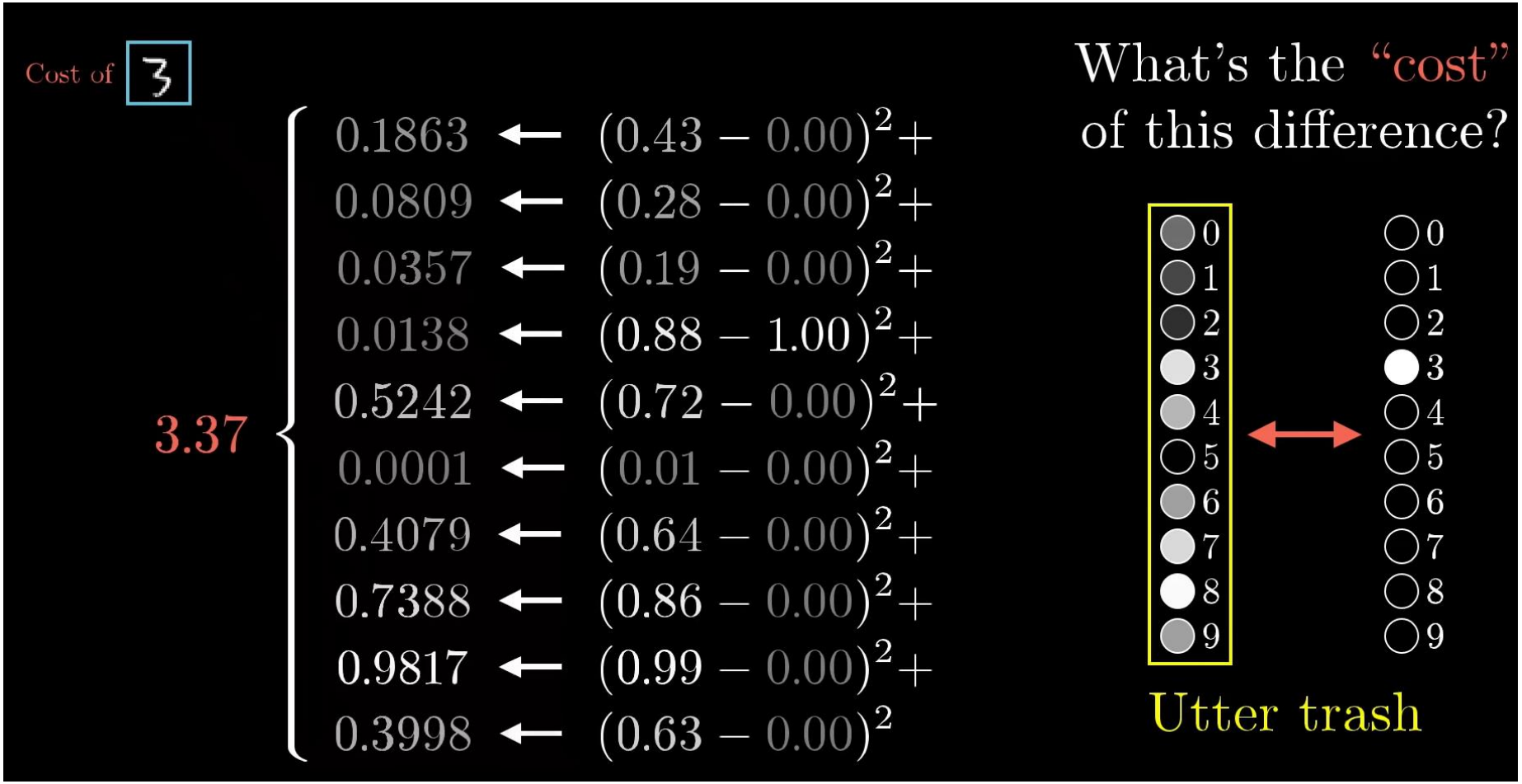
RECALL

# MNIST example



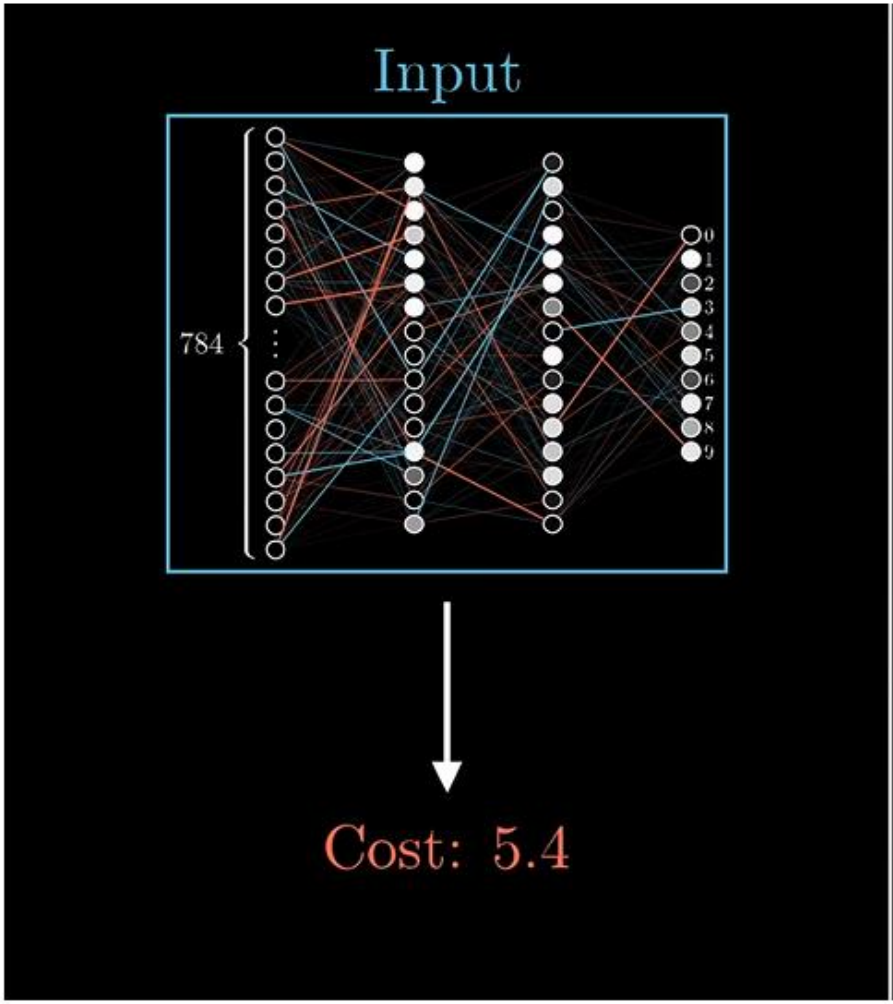
RECALL

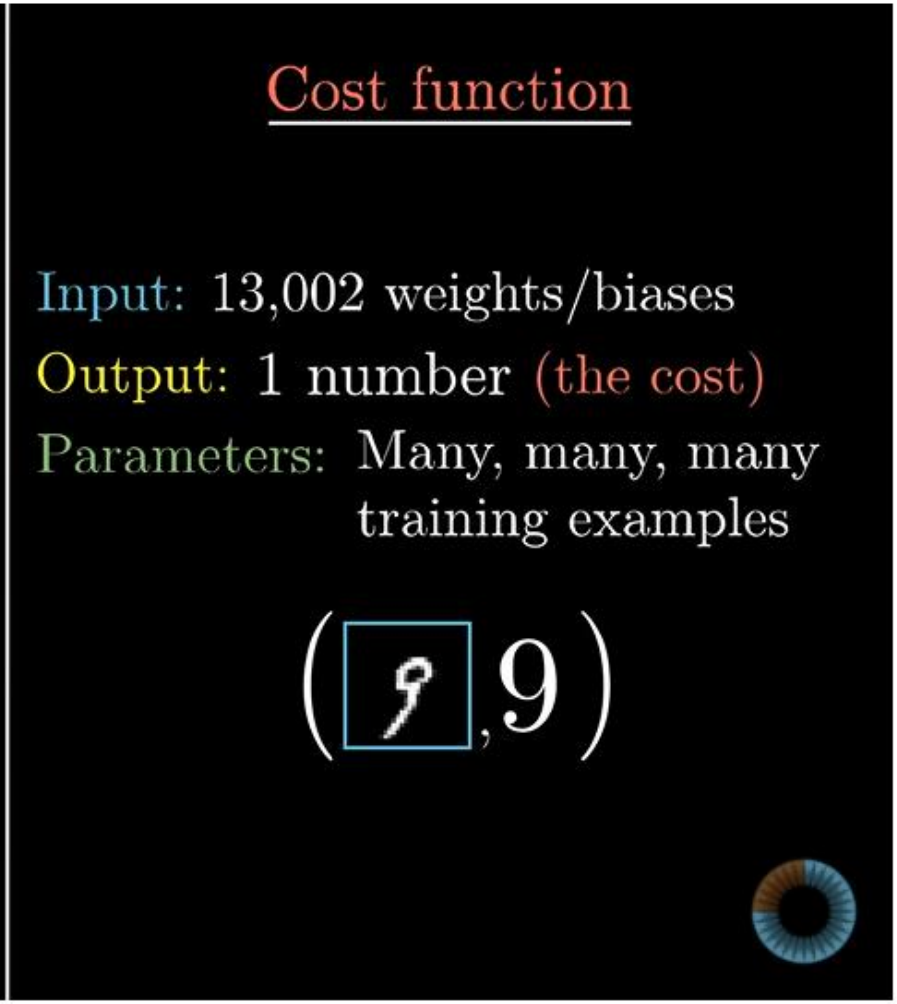
# Loss function



RECALL

# Loss function



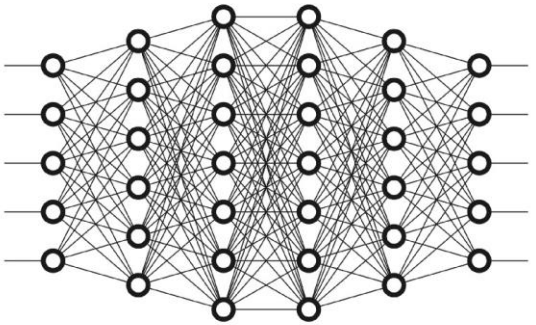



RECALL

# Data preparation, division and training

## Training Set

```
# Model 2
model=Sequential()
model.add (Dense(200,input_dim=trainX.shape[1], activation='sigmo
model.add (Dense(100, input_dim=200, activation='sigmoid'))
model.add (Dense(60, input_dim=100, activation='sigmoid'))
model.add (Dense(30, input_dim=60, activation='sigmoid'))
model.add(Dense(nb_classes, activation='softmax'))
```



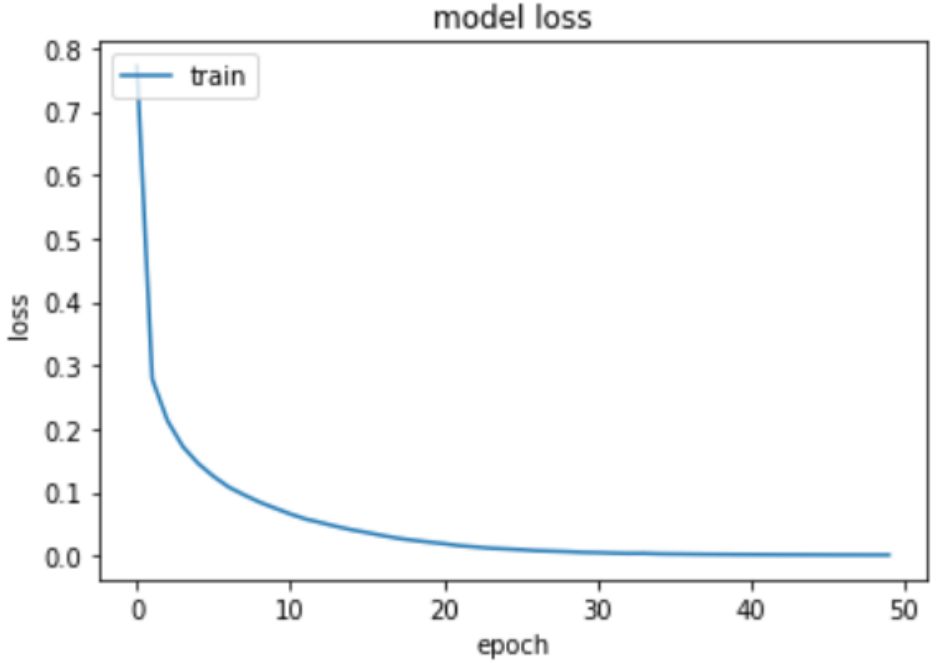
```
history=model.fit(x_train, y_train, epochs=2)
```

```
Epoch 1/2
60000/60000 [==============================] - 7s 111us/step - loss: 0.3656 - acc: 0.8994
Epoch 2/2
60000/60000 [==============================] - 7s 111us/step - loss: 0.3524 - acc: 0.9023
```

RECALL

# Data preparation, division and training

## Training Set




- Can we evaluate the model ?

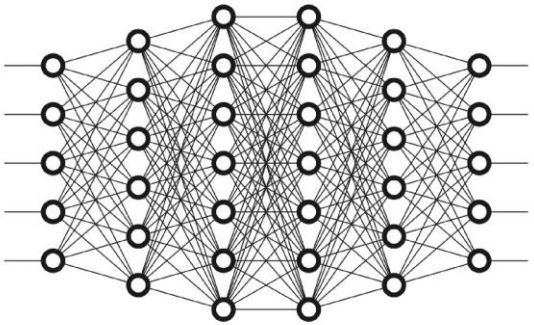
RECALL

# Data preparation, division and training

**Training Set**

**Validation Set**

```
# Model 2
model=Sequential()
model.add (Dense(200,input_dim=trainX.shape[1], activation='sigmo
model.add (Dense(100, input_dim=200, activation='sigmoid'))
model.add (Dense(60, input_dim=100, activation='sigmoid'))
model.add (Dense(30, input_dim=60, activation='sigmoid'))
model.add(Dense(nb_classes, activation='softmax'))
```



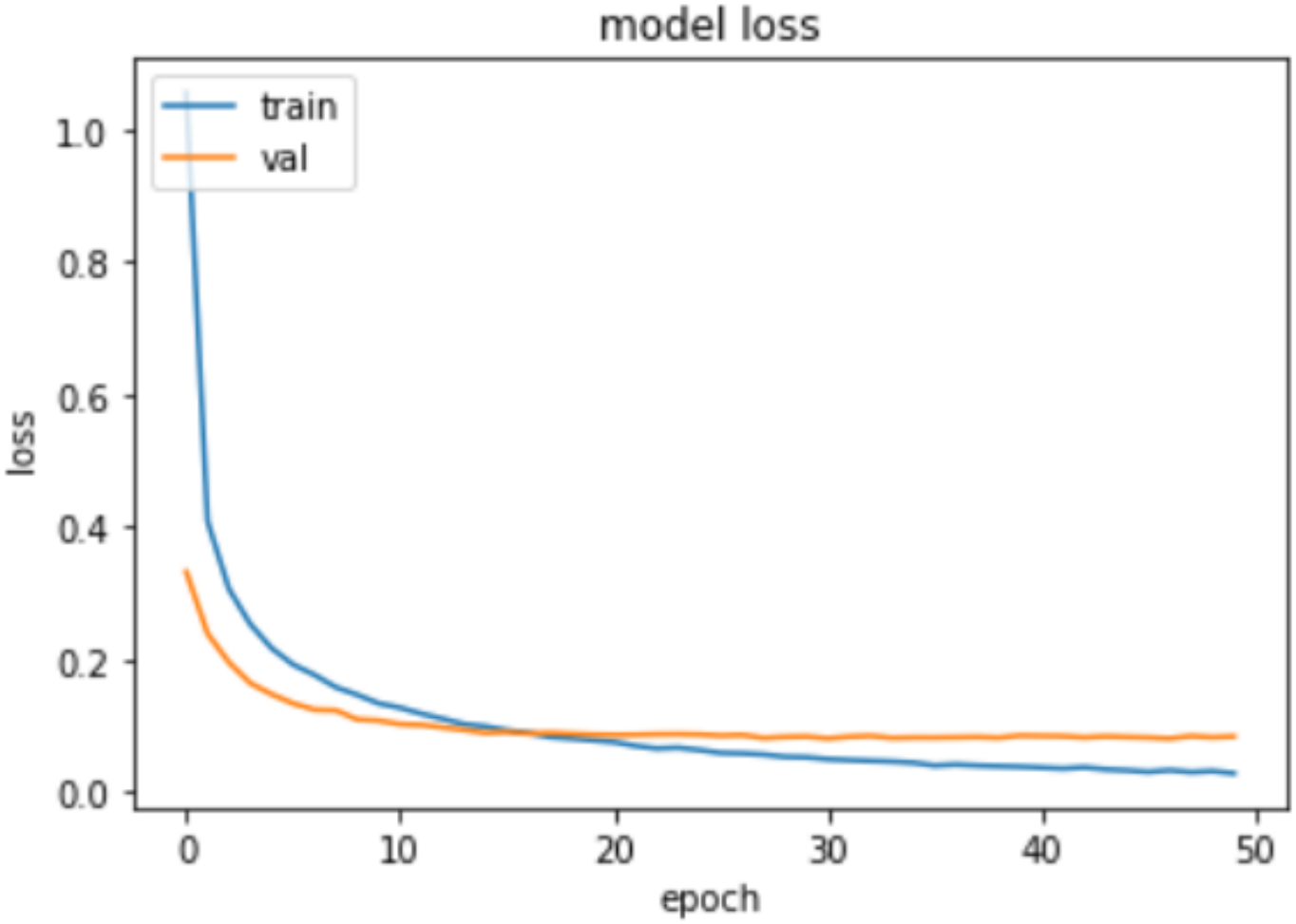
```
history=model.fit(x_train, y_train, validation_split = 0.2, epochs=2)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/2
48000/48000 [==============================] - 6s 123us/step - loss: 0.3483 - acc: 0.9028 - val_loss: 0.3206 - val_acc: 0.9113
Epoch 2/2
48000/48000 [==============================] - 6s 119us/step - loss: 0.3415 - acc: 0.9047 - val_loss: 0.3160 - val_acc: 0.9123
```

- Where is the difference ? Is it enough ?

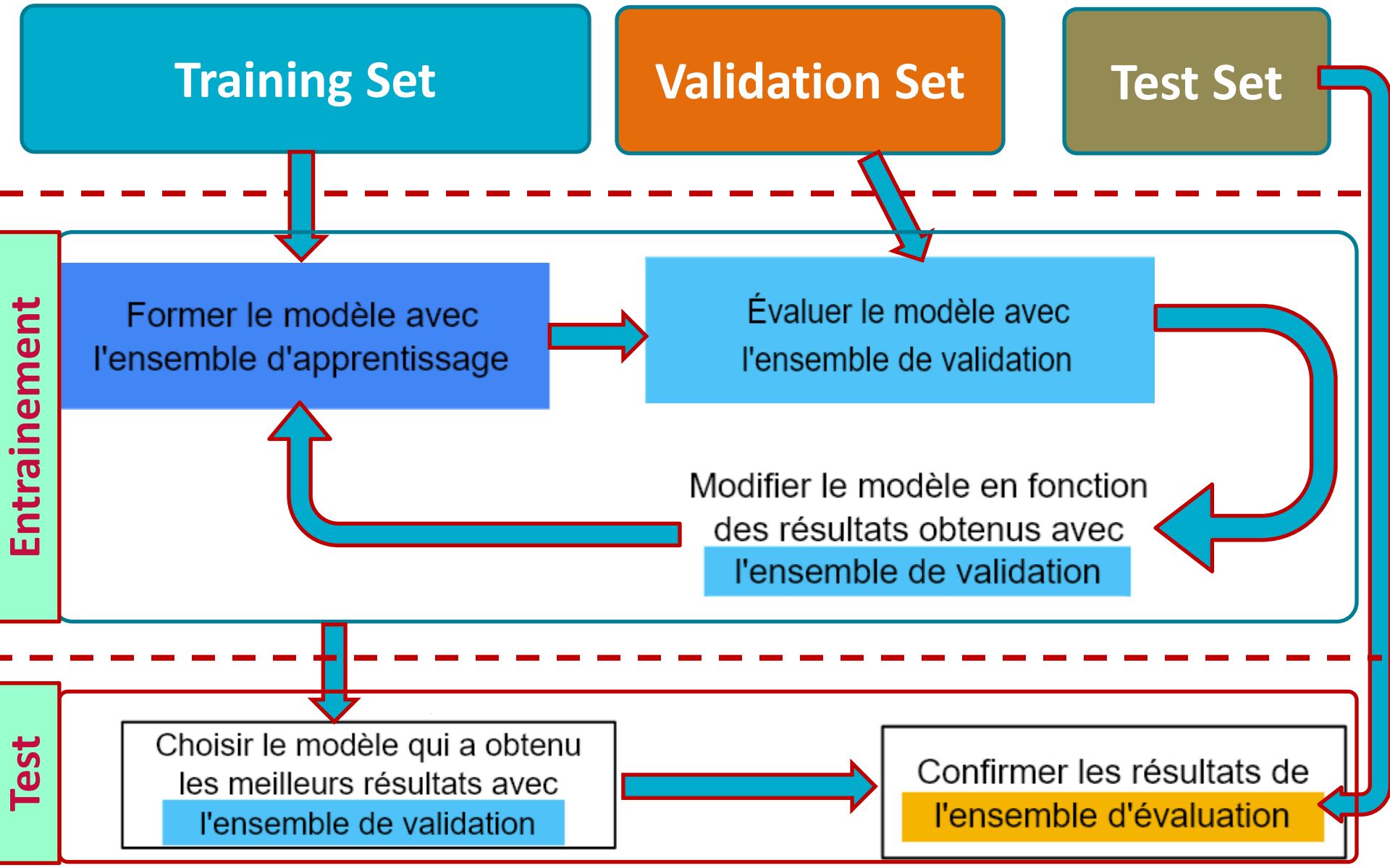
RECALL

# Data preparation, division and training



RECALL

# Training Process



RECALL

# Learning parameters

## Epocs

- 1 epoc : the entire dataset is passed through the network one time

## Batch_size

- Number of training data per batch
- Data divided within several batch

## Iterations

- Number of batch per epoc : dataset size/batch_size

RECALL

# Classic Gradient Descent

- **Data :** $(X_i, y_i)$ $i = 1 \ldots N$

  $X_i \in R^I$ : **Input** $y_i \in R$ : **expected output**

- **Objectif :** $F : R^I \rightarrow R$ $F(X_i) = y_i$
- **$F(X_i)$ :** predicted output
- F depends of parameters $a_1, a_2, \ldots, a_N$
- **Local error :** $E_i = (y_i - F(X_i))^2$
- **Total error :** $E = \sum Ei(a_1, a_2, \ldots, a_N)$ $(i= 1, \ldots, N)$
- But : **minimize the error using the le gradient**
- **Classic gradient:** grad $E = \sum grad\ Ei$ $(i= 1, \ldots, N)$

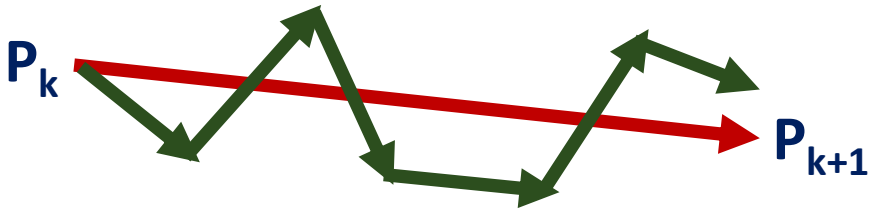
RECALL

# Classic Gradient Descent

- **Classic gradient descent**
  - $P_0 = (a_1, a_2, \ldots, a_N)$
  - $P_1 = P_0 - \sigma\ grad\ E(P_0)$
  - $P_2 = P_1 - \sigma\ grad\ E(P_1)$
  - E depend of all data
  - Problems of memory
  - Problems computation time

- **Stochastic Gradient Descent**
  - $P_0 = (a_1, a_2, \ldots, a_N)$
  - $P_1 = P_0 - \sigma\ grad\ E_1(P_0)$
  - $P_2 = P_1 - \sigma\ grad\ E_2(P_1)$

  But :
  - Local error
  - Less intensive and progressive

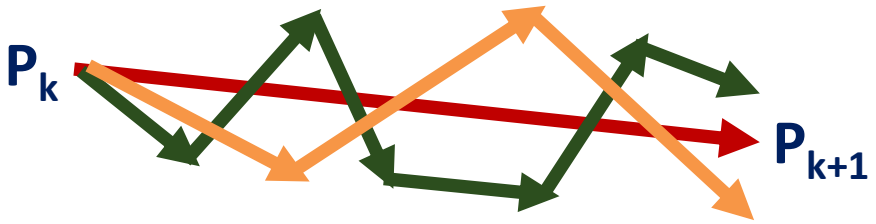


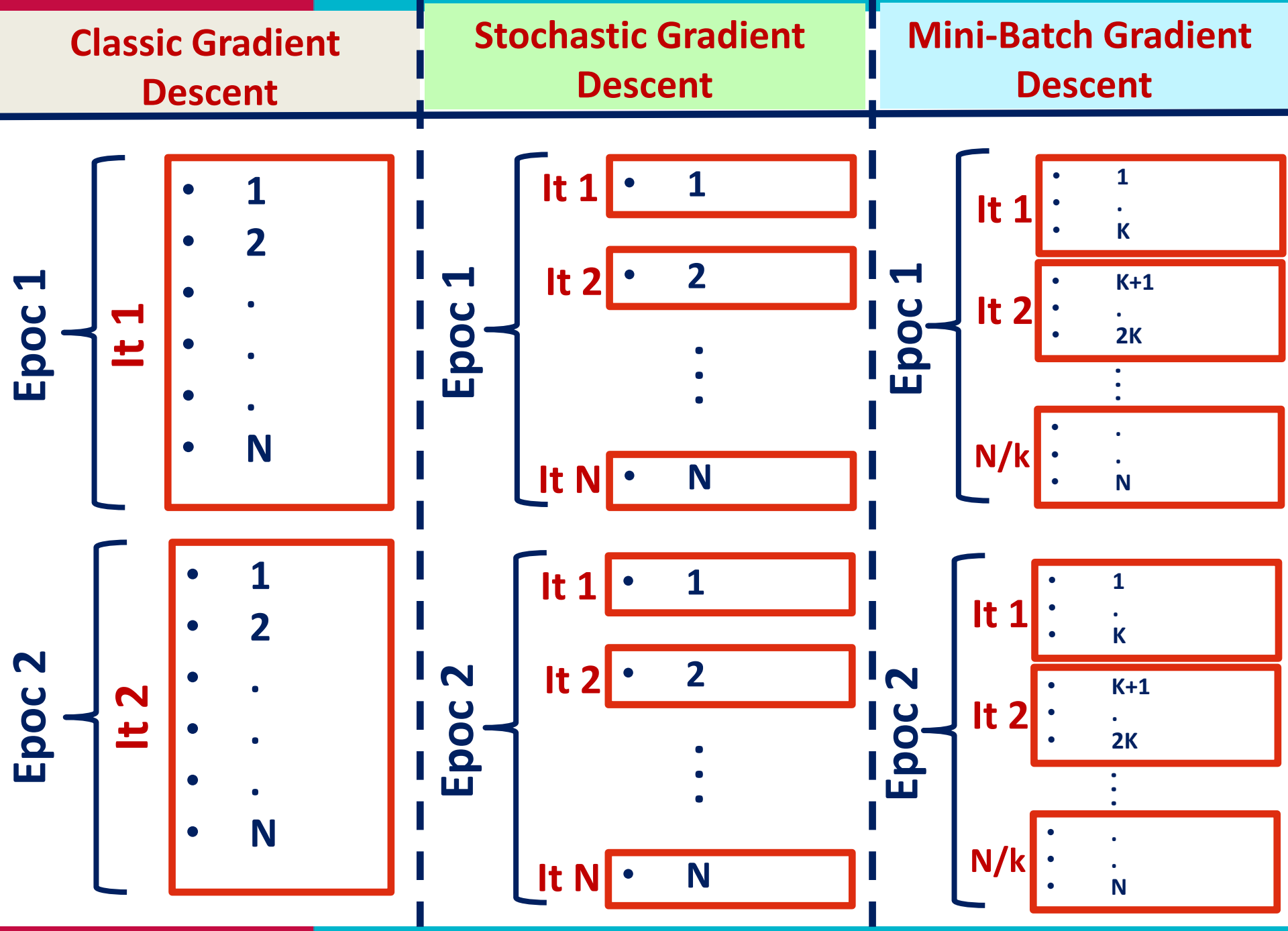
RECALL

# Batch Gradient Descent

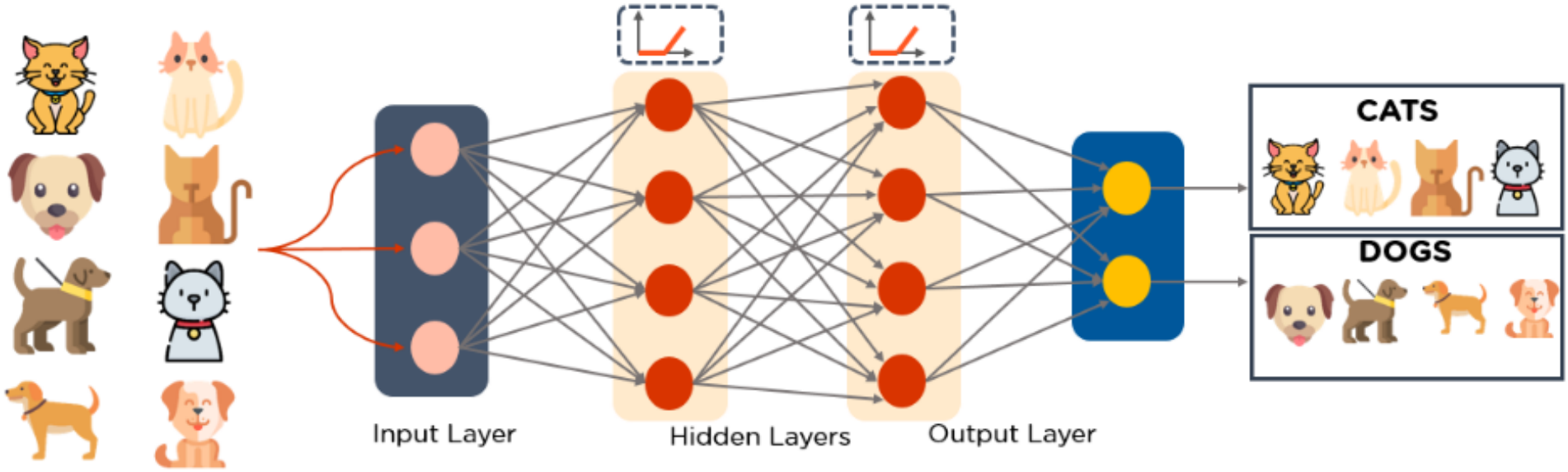
- **Batch Gradient Descent**
  - Intermediate solution
  - N data
  - $P_0 = (a_1, a_2, \ldots, a_N)$
  - $P_1 = P_0 - \sigma\, grad\,(E_1 + E_2 + Ek)\,(P_0)$
  - $P_2 = P_1 - \sigma\, grad\,(E_{k+1} + E_{k+2} + E_{2k})\,(P_1)$

N/k iterations : all data have served on time : **1 époc**



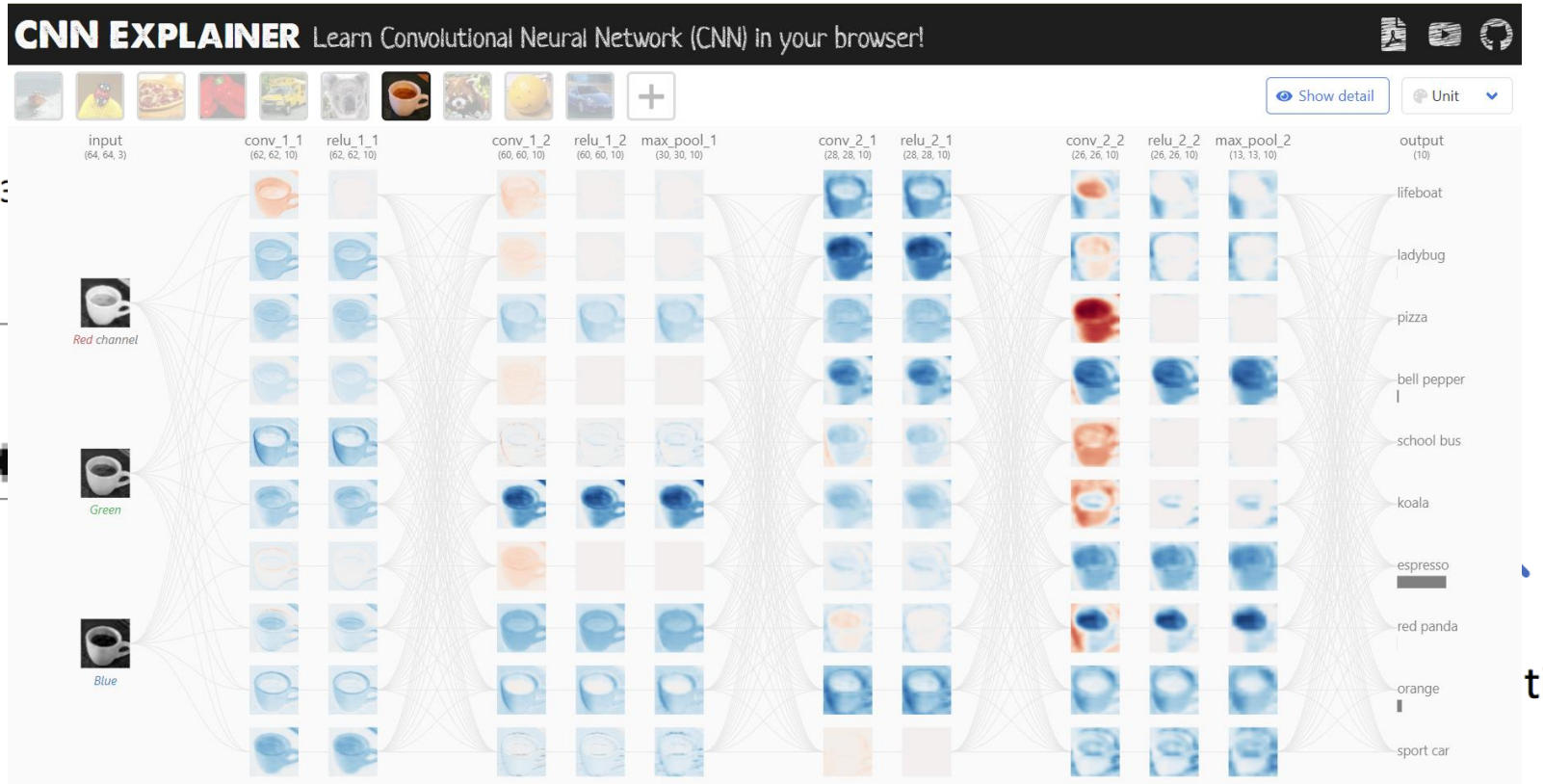

Classic Gradient Descent
Stochastic Gradient Descent
Mini-Batch Gradient Descent
Epoc 1
It 1
1
2
.
.
.
N
Epoc 2
It 2
1
2
.
.
.
N
Epoc 1
It 1
1
It 2
2
It N
N
Epoc 2
It 1
1
It 2
2
It N
N
Epoc 1
It 1
1
.
K
It 2
K+1
.
2K
N/k
.
.
N
Epoc 2
It 1
1
.
K
It 2
K+1
.
2K
N/k
.
.
N

# Types of Deep Neural Networks : MLP




- **Input layer** : One dimension vector (needs data adaptation)
- **Applications :** classification, regression
- **Use case examples :** energy consumption prediction, speech recognition, image classification, etc.

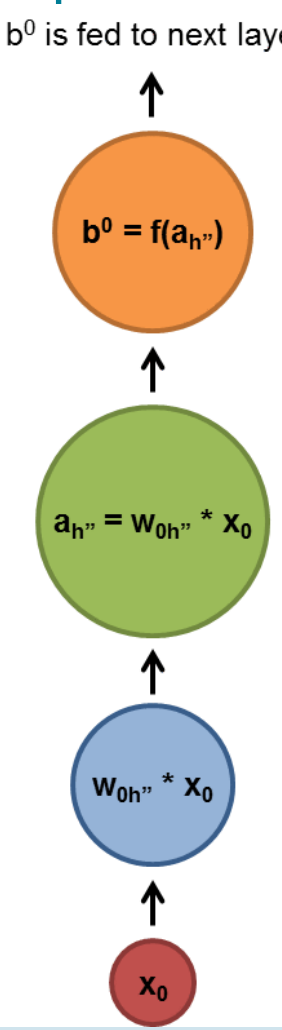
# Types of Deep Neural Networks : CNN




https://poloclub.github.io/cnn-explainer/

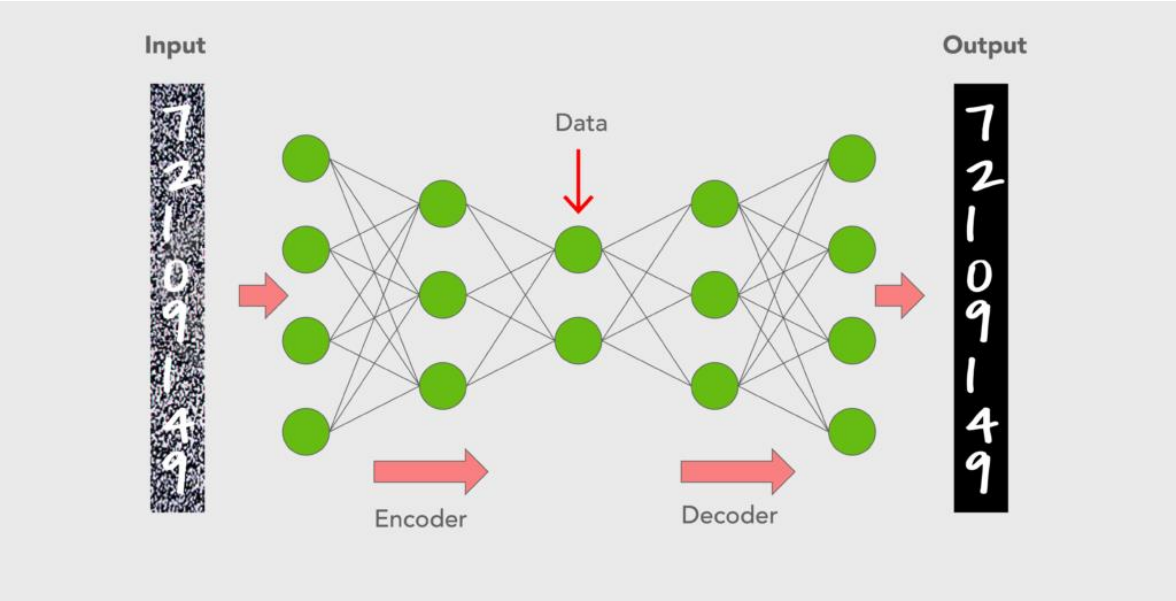
- **Input layer** : 2D image (CNN 2D) or 3D image (CNN 3D)
- **Applications :** classification, regression
- **Use case examples :** images classification, object detection, etc.

# Types of Deep Neural Networks : RNN




- Inp
- App
- Use on, text classification, energy prediction, etc.

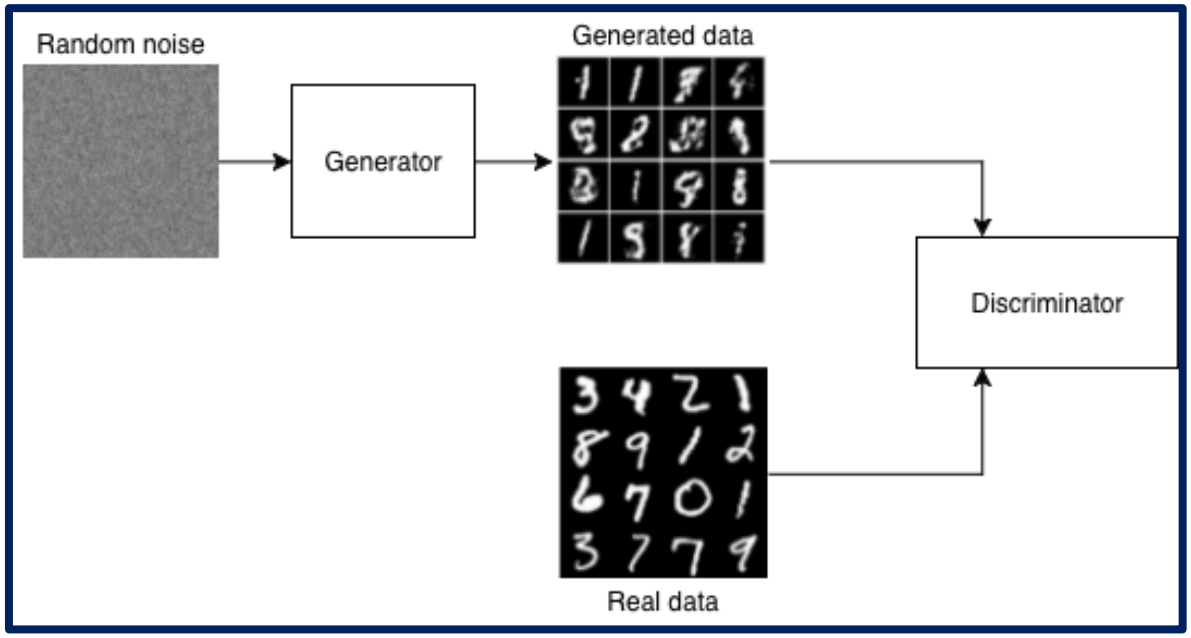
# Types of Deep Neural Networks : Auto-encoders




## Auto-encoders

- **Input layer :** shape similar to the output layer
- **Applications :** reconstruction, image segmentation, etc.
- **Use case examples :** anomalies detection in industry 4.0, medical images segmentation, etc.

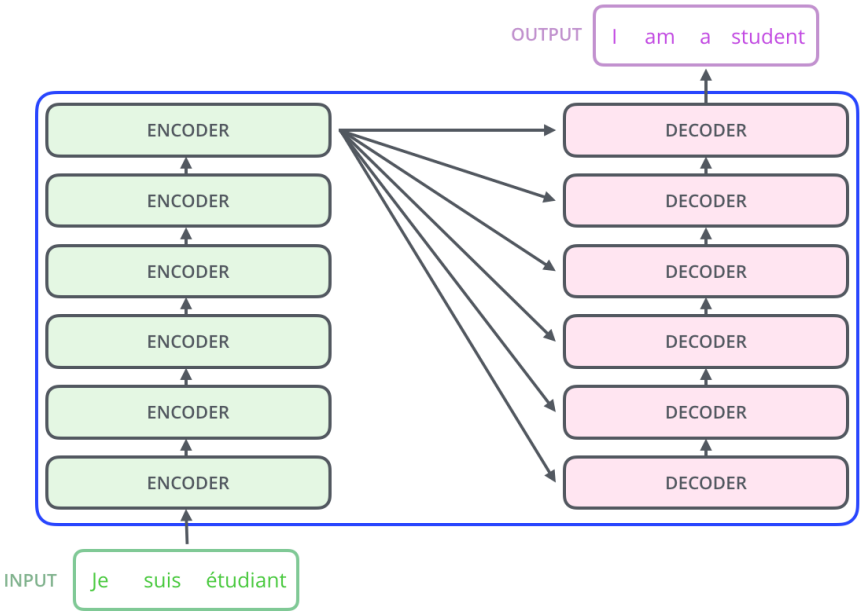
# Types of Deep Neural Networks : GAN

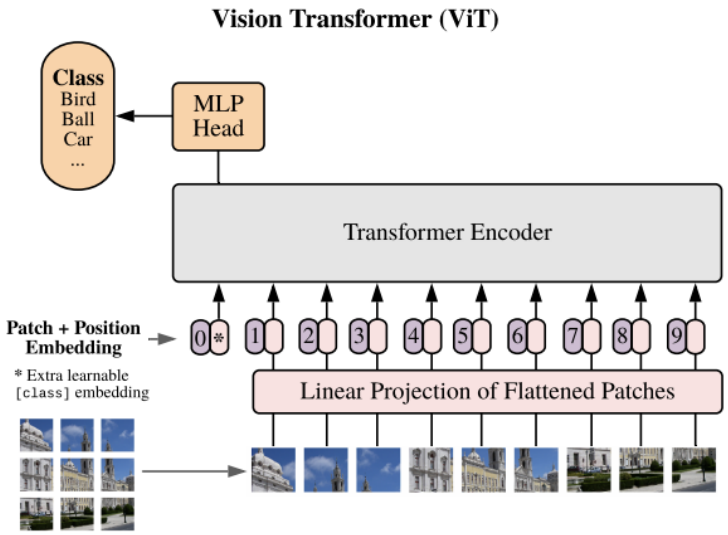



## GAN : Generative Adversarial Network

- **Input layer :** images, texts
- **Applications :** images generation, translation, etc.
- **Use case examples :** fake face generation, data augmentation, super resolution, etc.

# Types of Deep Neural Networks : Transformers




**Transformers** **Vision Transformers**

https://poloclub.github.io/transformer-explainer/

- **Input layer** : images, texts, etc.
- **Applications :** classification, regression, reconstruction, etc.
- **Use case examples :** images classification, text classification, translation, etc.

# PLAN

Introduction

I. **Performance analysis**

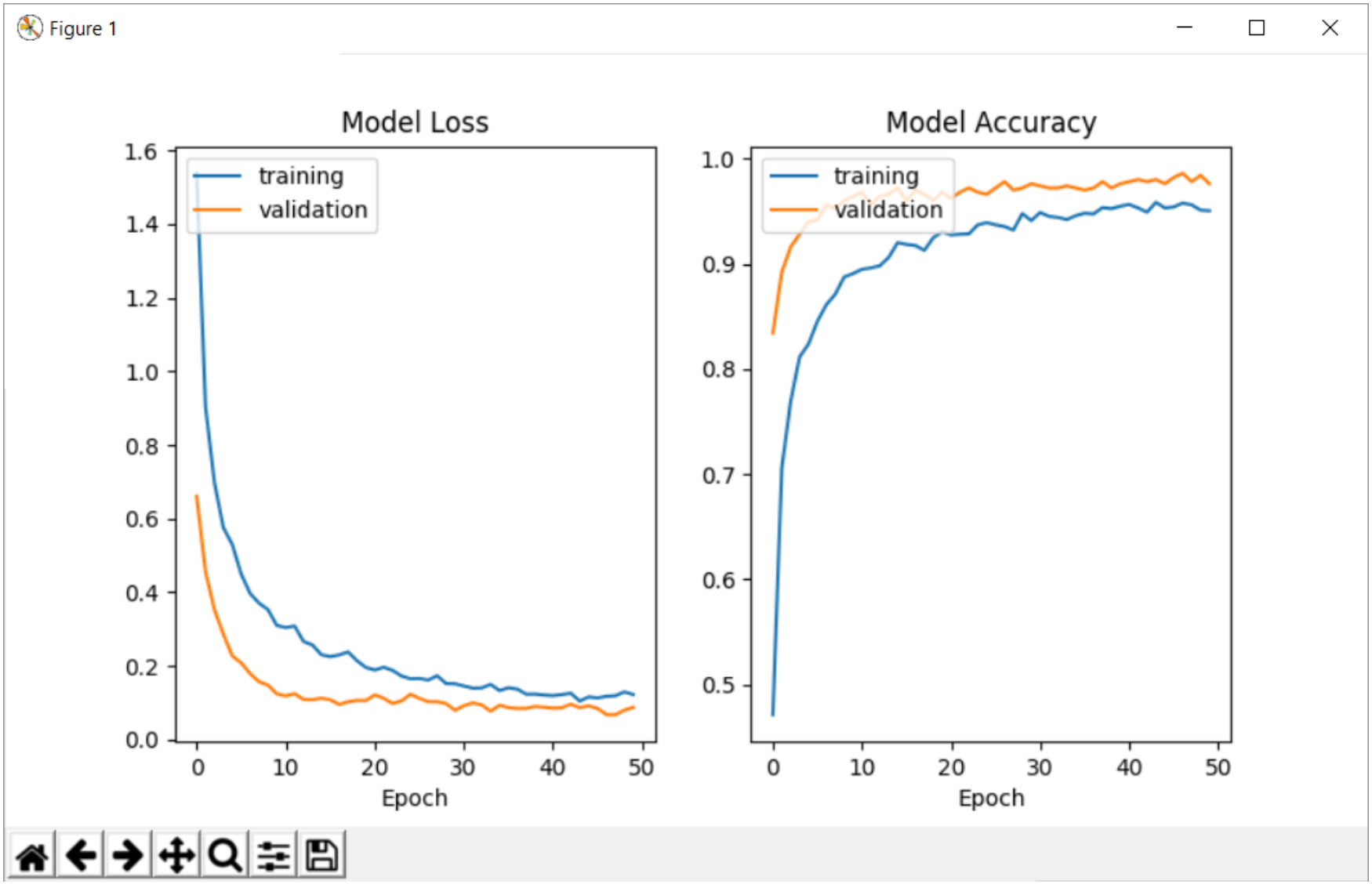
II. Performance optimization : regularization

- L2 & L1 regularization
- Dropout
- Data augmentation
- Early stopping

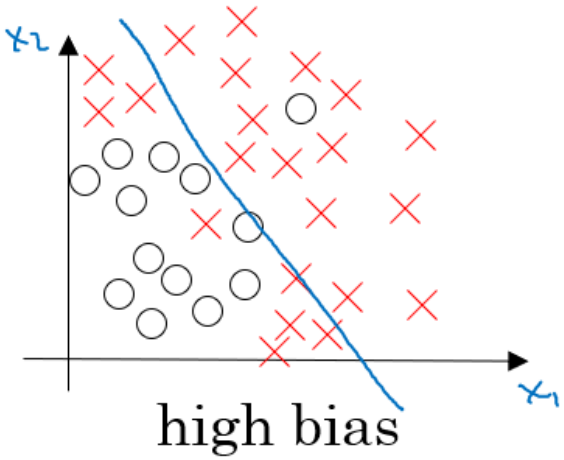
III. Batch normalization

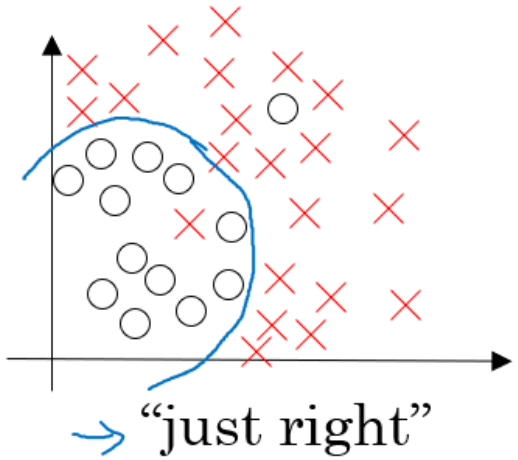
Conclusion

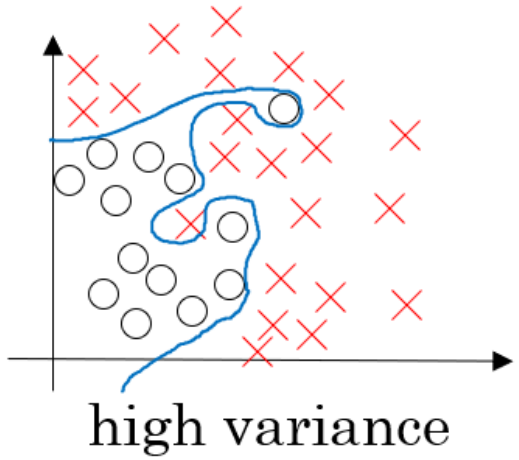
# Performance analysis : Loss/accuracy curve



# Bias & Variance




**Underfitting** **Optimum** **Overfitting**

# Bias

- **Bias :** difference between the average prediction of our model and the correct values that we are trying to predict.
- **High bias :** little attention to the training data, where the model oversimplifies the solution.
- **High bias :** high error on training and test data

# Variance

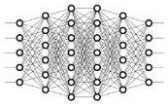
- **Variance :** error that occurs due to a model's sensitivity to small fluctuations in the training set.
- **High variance:** a lot of attention to training data, which can not generalize on data that have never been seen before
- **High variance:** the models perform very well on training data but has high errors with test (or validation) data

# Bias & Variance

Cat classification

| | | | | |
|---|---|---|---|---|
| **Training error** | **1 %** | **15 %** | **20 %** | **1 %** |
| **Validation error** | **12 %** | **1 %** | **30 %** | **1,5 %** |
| | **High variance** | **High Bias** | **High Bias & Var** | **Low Bias & Var** |

# Deep Learning Process



**High Bias**

YES

**Bigger network**

Training error

Layers & architecture

NO

**High Variance**

YES

**More data & regularization**

Test error

Layers & architecture

**DONE**

# Lesson 2

# Cross-validation


- Training and validation using the same datasets among iterations
- The use of the same data may cause **overfitting**
- Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set
- At the end, each point (or observation) has been used once in the test dataset and (k-1) times in the training dataset

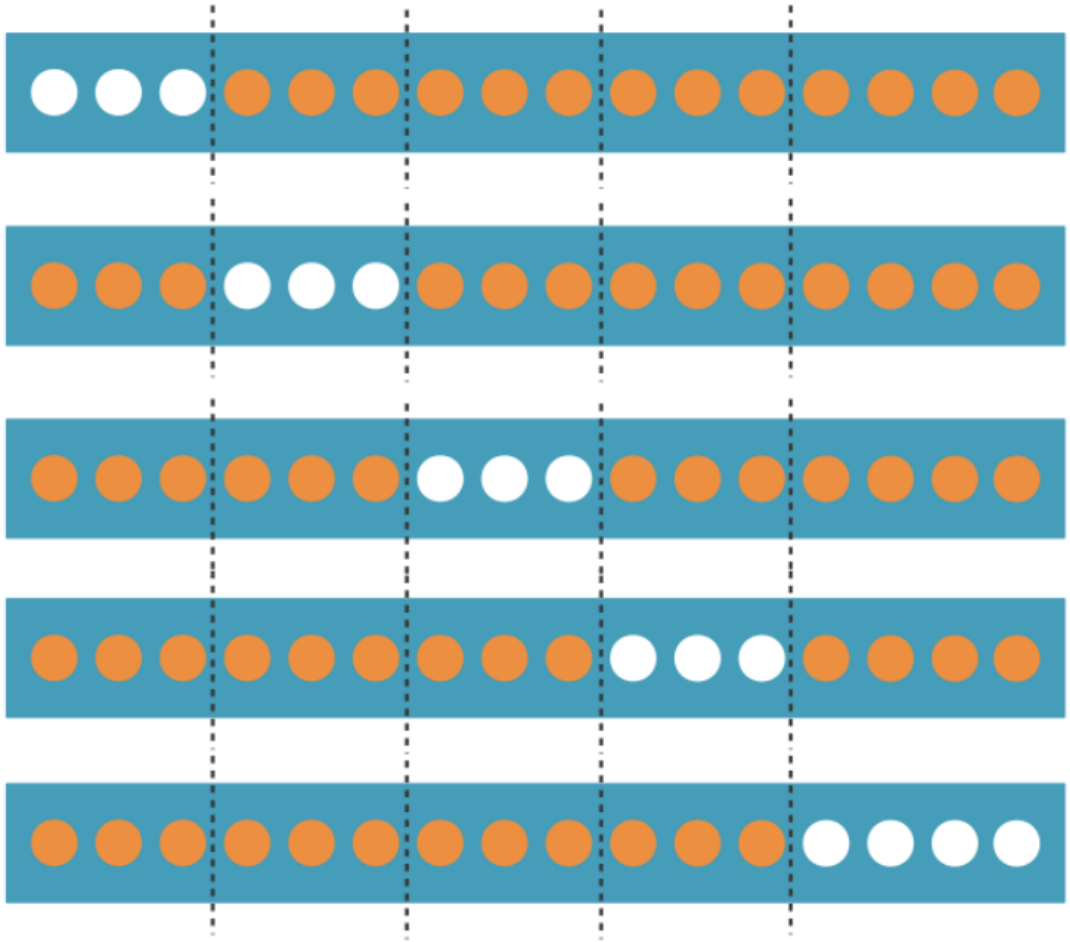
# Cross-validation

- Cross-validation will allow us to use the entire dataset for training and validation
  - The data set is divided into approximately equal parts (folds )
  - In turn, each of the k parts is used as a test dataset.
  - The rest (that is, the union of the other k-1 parts) is used for training
- Called also : K-fold cross-validation

# Cross-validation

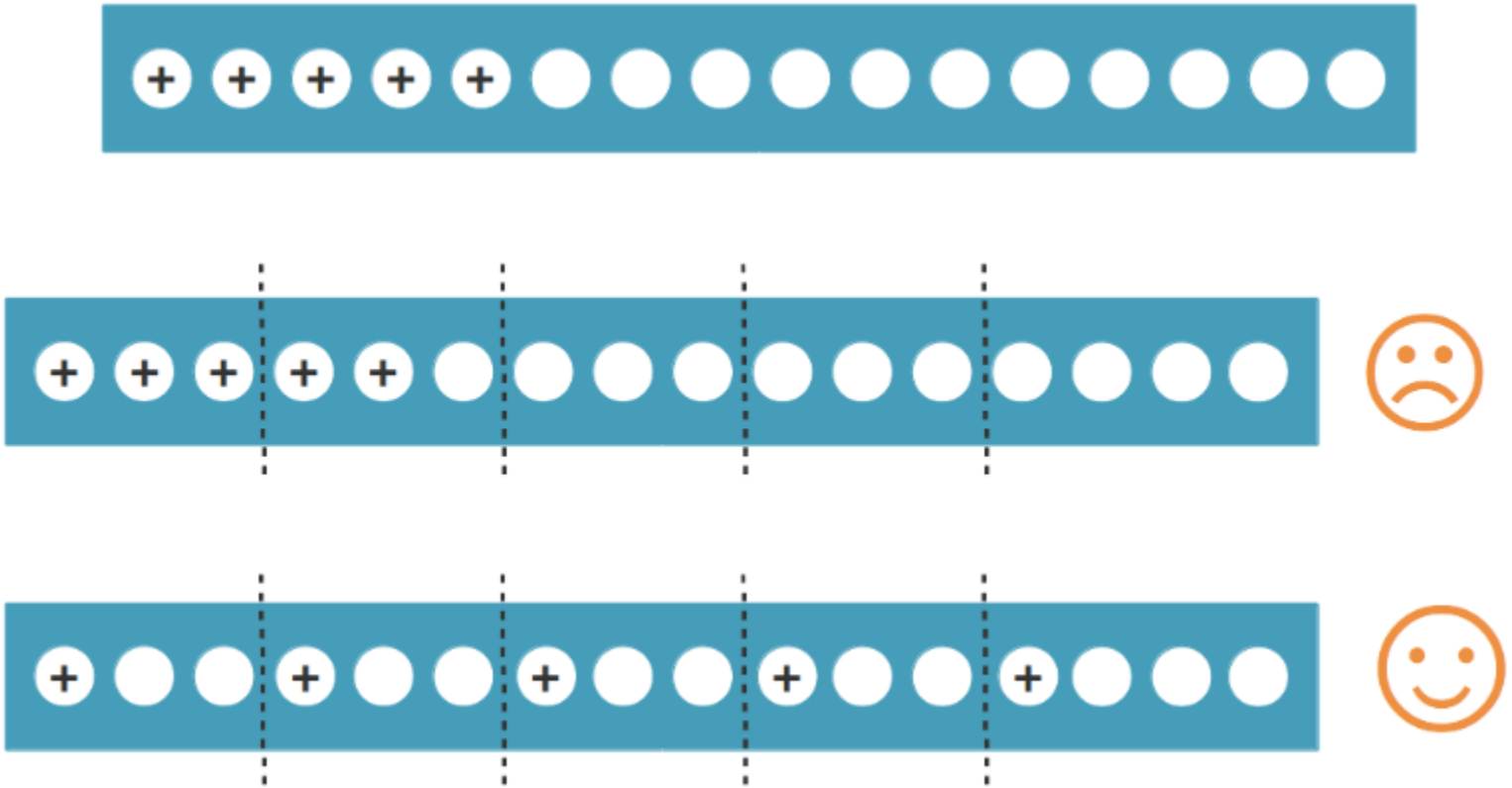
1 fold
(test data)

K-1 fold
(training data)



k=5

# Cross-validation



- With Sickit learn : **model_selection.StratifiedKFold**

# Cross-validation process

- shuffle the dataset randomly.
- split the dataset into k groups
- For each unique group:
  - Take the group as a hold out or test data set
  - Take the remaining groups as a training data set
  - Fit a model on the training set and evaluate it on the test set
  - Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

# Performance analysis : confusion matrix

- **Error matrix :** table that describes the performance of a **classification**
- Computation based on test data from which true values are known
- Visualize the model (or algorithm) performance
- Identify confusion between classes
- **Example :** One class commonly mislabeled as other one
- Most performance measures are computed from the confusion matrix

# Confusion matrix

| | | Predicted class | |
|---|---|---|---|
| | | **Courriel** | **Spam** |
| **Real class** | **Courriel** | **95**<br>TP (True positives) | **05**<br>(FN) (False negatives) |
| | **Spam** | **03**<br>(FP) (False positives) | **97**<br>(TN) (True negatives) |

# Confusion matrix

| Predicted \ Truth | Asphalt | Concrete | Grass | Tree | Building | Total |
|---|---|---|---|---|---|---|
| Asphalt | 2385 | 4 | 0 | 1 | 4 | 2394 |
| Concrete | 0 | 332 | 0 | 0 | 1 | 333 |
| Grass | 0 | 1 | 908 | 8 | 0 | 917 |
| Tree | 0 | 0 | 0 | 1084 | 9 | 1093 |
| Building | 12 | 0 | 0 | 6 | 2053 | 2071 |
| Total | 2397 | 337 | 908 | 1099 | 2067 | 6808 |

- Accuracy : 6792/6808 = **0.993243**

# PLAN

Introduction

I. Performance analysis

II. **Performances optimization : regularization**

- L2 & L1 regularization
- Dropout
- Data augmentation
- Early stopping

III. Batch normalization

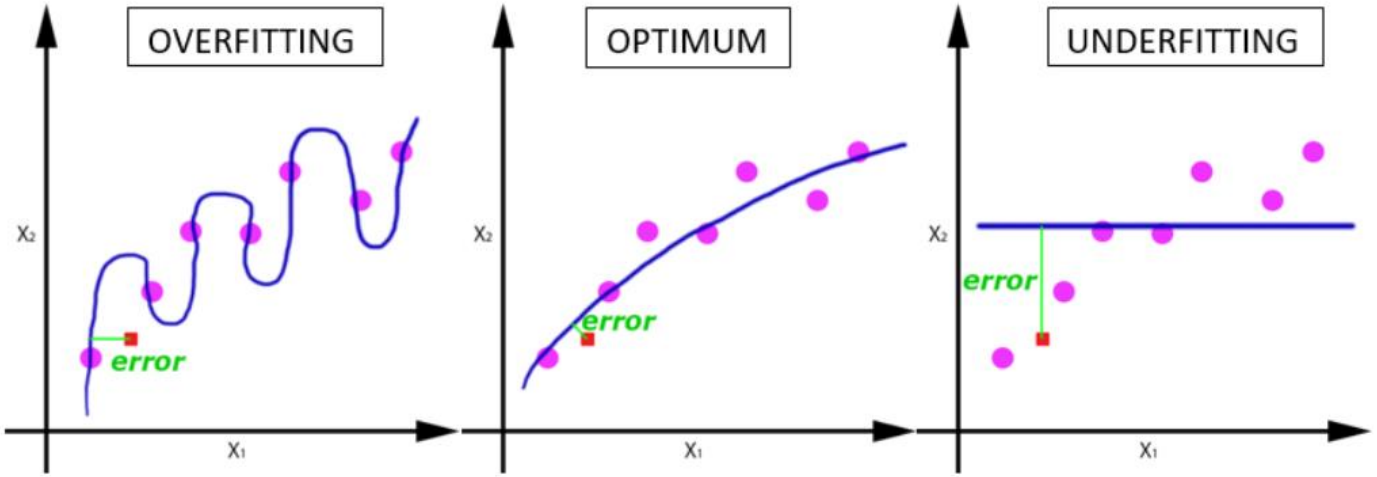
Conclusion

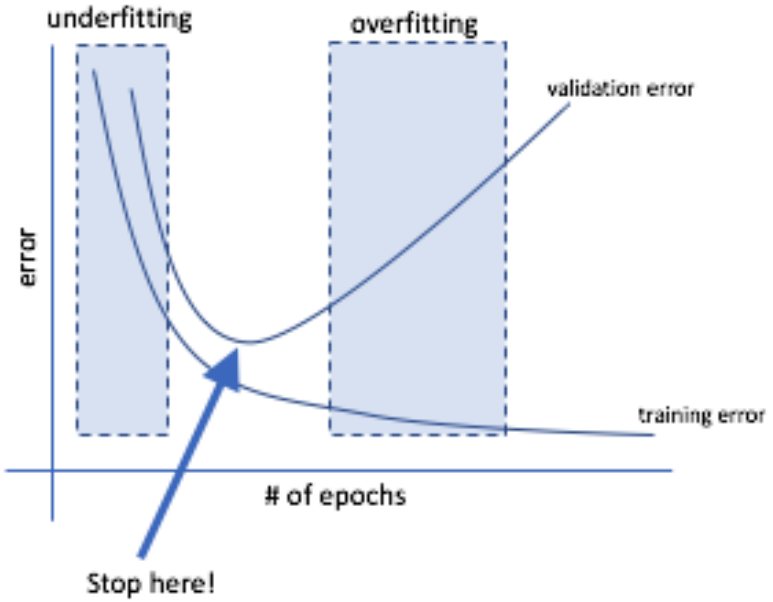
# Regularization

- **Why regularization ?**
- What is regularization ?

To solve one of the most common problem of data science and machine learning : **Overfitting**

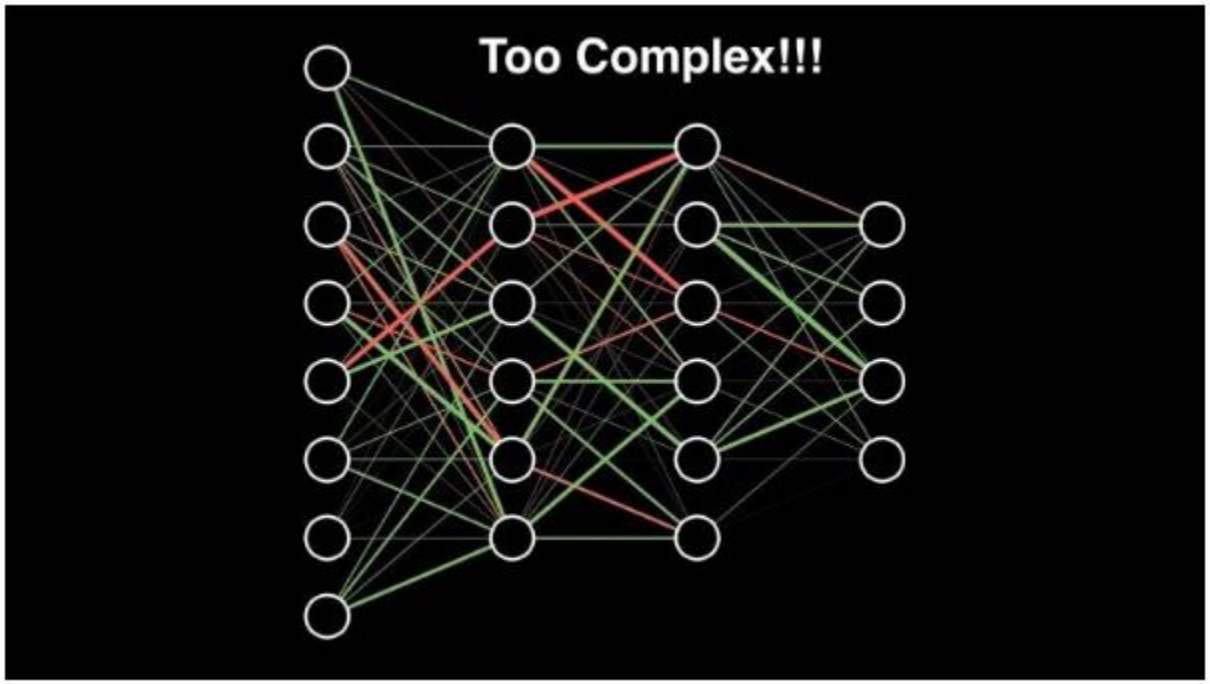
# Regularization



# Regularization

- Overfitting as a result of the use of too complex neural networks

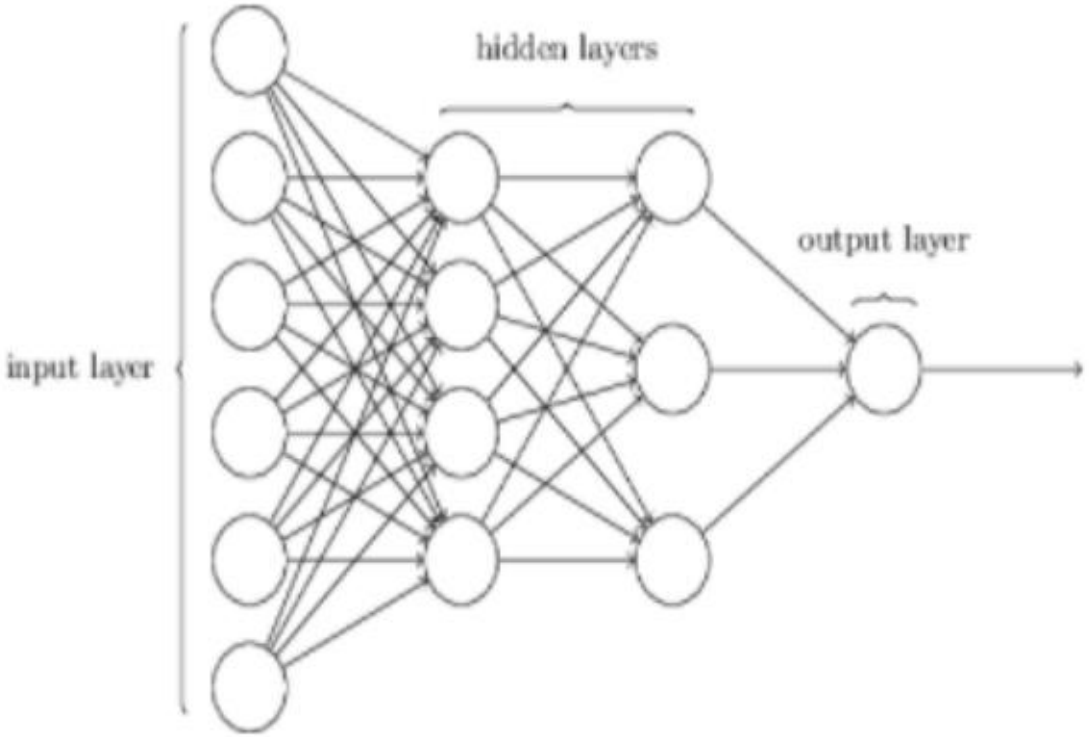


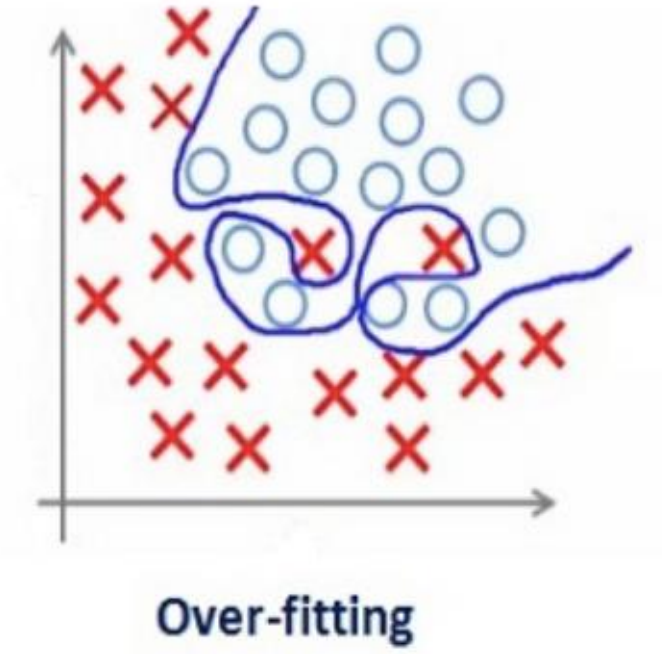
# Regularization

- Why regularization ?
- **What is regularization ?**

A model that performs very well on train data but performs very bad with test data.

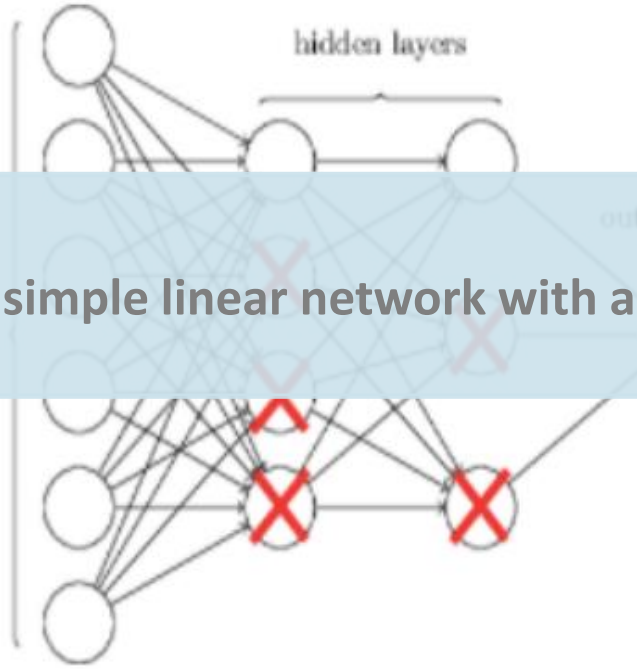
# Regularization : example



# Regularization : example

- Regularization penalizes the coefficients.
- In deep learning, it penalizes the weight matrices of the nodes
- **High** regularization coefficient : weight matrices are near to zero value

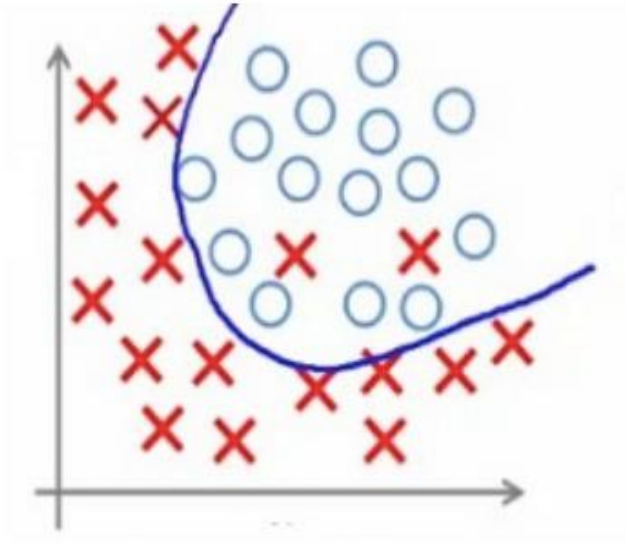




**Very simple linear network with a slight underfitting of the training data**

# Regularization : example

- Large regularization values : not so useful
- **Optimize the values** of regularization coefficient to get well-fitted model



Appropriate-fitting

# Regularization techniques

- L2 & L1 regularization
- Dropout
- Data augmentation
- Early stopping

# PLAN

Introduction

I. Performance analysis

II. **Performances optimization : regularization**

- **L2 & L1 regularization**
- Dropout
- Data augmentation
- Early stopping

III. Batch normalization

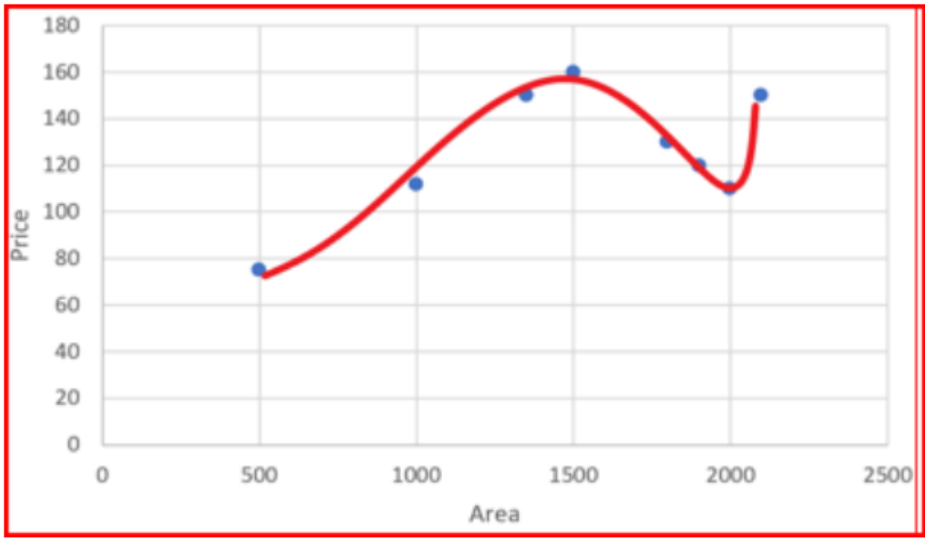
Conclusion

# L2 & L1 regularization

$$L(x, y) = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$f(x_i) = h_\theta x = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$$

**Loss function**




**Overfitting**

# L2 & L1 regularization

$$f(x_i) = h_\theta x = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$$

$$f(x_i) = h_\theta x = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$$

**Penalize some weights**

**Regularization : smaller weights generate simpler model that avoid overfitting.**

# Regularization solution

$$L(x,y) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

where $h_\theta x_i = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$

$$L(x,y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

# L2 & L1 regularization

- **L1 & L2 :** most common types of regularization. These update the general cost function by adding another term known as the regularization term.

**Cost function = Loss + Regularization term**

- With the regularization term, the values of weight matrices decreases
- It assumes a neural network with smaller weight matrices : simpler models
- Regularization term differ  between L1 and L2

# L1 & L2 regularization

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

# L1 & L2 regularization

- Lambda : the regularization hyperparameter : optimized for better results
- L2 regularization : weight decay as it forces the weights to decay towards zero (**but not exactly zero**).
- L1 regularization : penalize the absolute value, where weights **may be reduced to zero**. Useful when we try **to compress the model**.

# L1 & L2 regularization : mathematically

- Model :

$$\hat{y} = wx + b$$

- Loss :

$$L = (\hat{y} - y)^2 = (wx + b - y)^2$$

- L1 :

$$L_1 = (wx + b - y)^2 + \lambda |w|$$

$$\frac{d|w|}{dw} = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases}$$

- L2 :

$$L_2 = (wx + b - y)^2 + \lambda w^2$$

# L1 & L2 regularization : Gradient Descent

Weights update :

$$w_{\text{new}} = w - \eta \frac{\partial L}{\partial w}$$

L : No regul

$$w_{\text{new}} = w - \eta \frac{\partial L}{\partial w} = w - \eta \cdot \left[2x(wx + b - y)\right]$$

L1 :

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_1}{\partial w} \\ &= w - \eta \cdot \left[2x(wx + b - y) + \lambda \frac{d|w|}{dw}\right] \\ &= \begin{cases} w - \eta \cdot \left[2x(wx + b - y) + \lambda\right] & w > 0 \\ w - \eta \cdot \left[2x(wx + b - y) - \lambda\right] & w < 0 \end{cases} \end{aligned}$$

# L1 & L2 regularization : Gradient Descent

Weights update

$$w_{\text{new}} = w - \eta \frac{\partial L}{\partial w}$$

L : No regul

$$w_{\text{new}} = w - \eta \frac{\partial L}{\partial w} = w - \eta \cdot \left[2x(wx + b - y)\right]$$

L2 :

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_2}{\partial w} \\ &= w - \eta \cdot \left[2x(wx + b - y) + 2\lambda w\right] \end{aligned}$$

# L1 & L2 regularization : Gradient Descent

Let's imagine

$\eta = 1$

$$H = 2x(wx + b - y)$$

L:

$$w_{\text{new}} = w - H \quad \text{——} \quad (0)$$

L1:

$$w_{\text{new}} = \begin{cases} (w - H) - \lambda, & w > 0 \quad \text{——} \quad (1.1) \\ (w - H) + \lambda, & w < 0 \quad \text{——} \quad (1.2) \end{cases}$$

L2:

$$w_{\text{new}} = (w - H) - 2\lambda w \quad \text{——} \quad (2)$$

# L1 & L2 regularization : in practice

- Regularization applied per layer
- The value of 0.01 : regularization parameter : lambda
- The value of lambda need to be optimized
- We can optimize it using the grid-search method.

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01)
```

# PLAN

Introduction

I. Performance analysis

II. **Performances optimization : regularization**

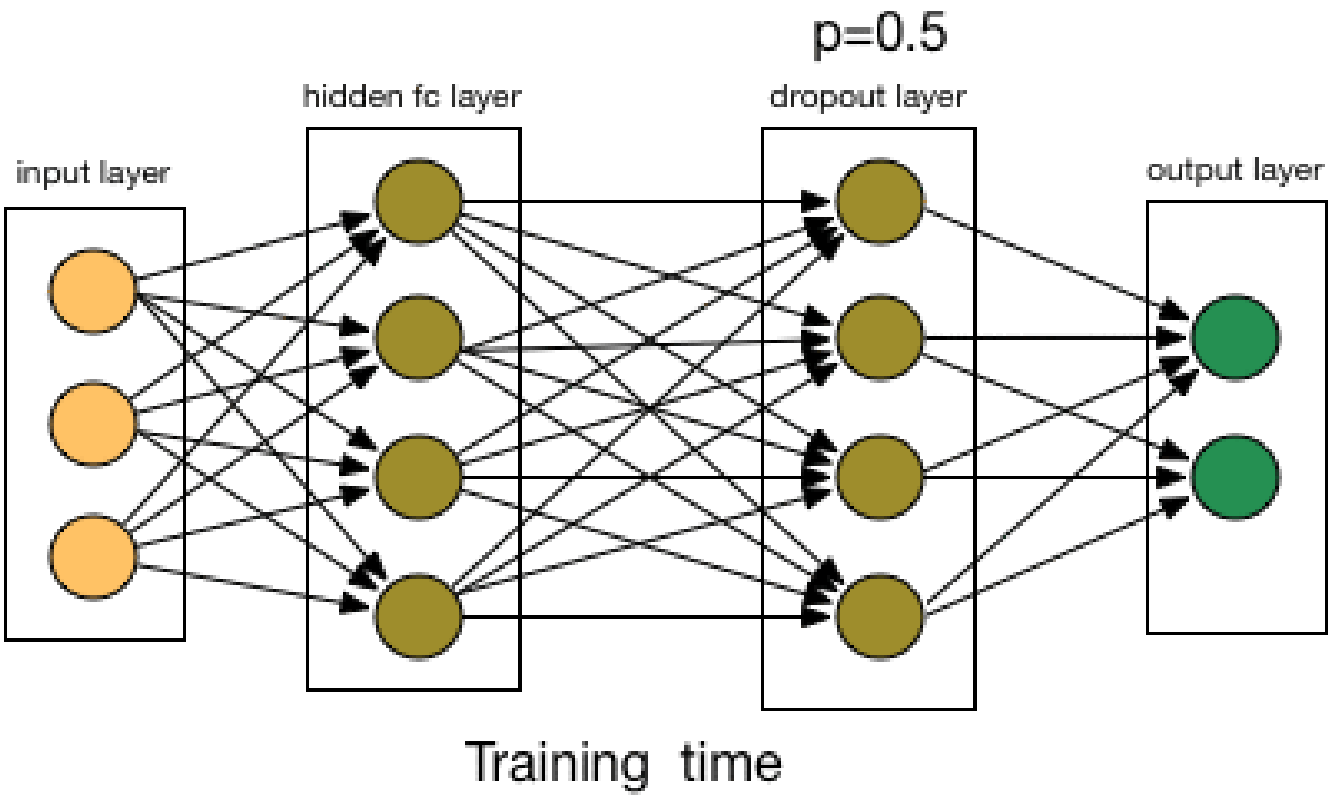
- L2 & L1 regularization
- **Dropout**
- Data augmentation
- Early stopping

III. Batch normalization

Conclusion

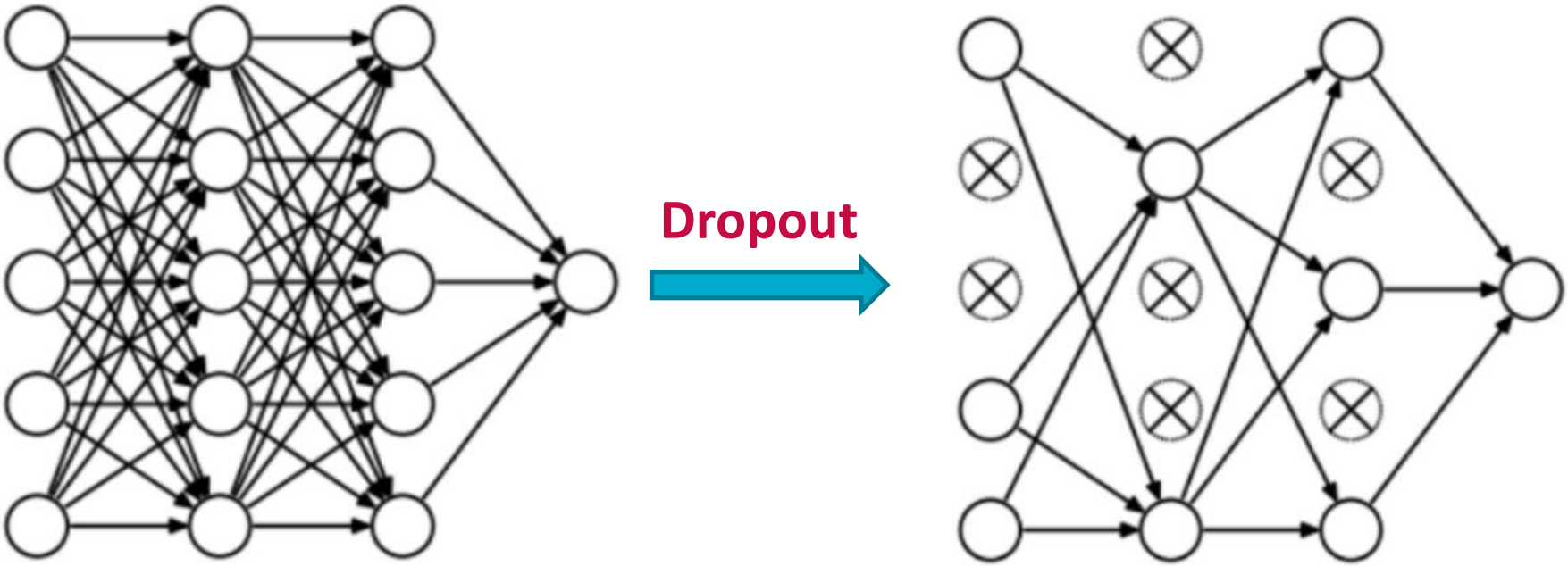
# Dropout

- Usually preferred for **large neural networks** in order to introduce more randomness

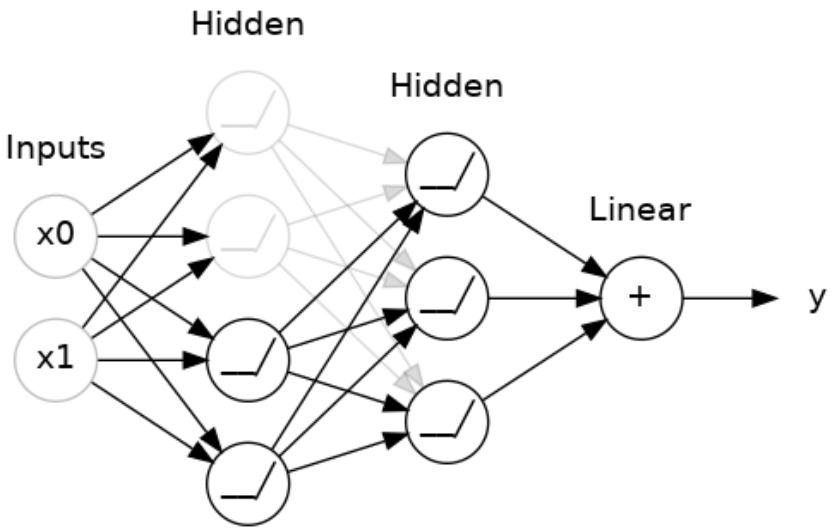


# Dropout

- At each iteration, it randomly selects some nodes and removes them along with all the incoming and outgoing connections.
- It can be applied to both the hidden layers as well as the input layers



# Dropout




```python
class MLP3(nn.Module):
    def __init__(self):
        super(MLP3, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 4)
        self.fc2 = nn.Linear(4, 3)
        self.dropout1 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(3, 1)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = self.dropout1(x)
        x = self.fc3(x)
        return x
```

# PLAN

Introduction

I. Performance analysis

II. **Performances optimization : regularization**

- L2 & L1 regularization
- Dropout
- **Data augmentation**
- Early stopping

III. Batch normalization

Conclusion

# Data augmentation

- Simple way to face overfitting : increase the size of the training data.
- The of increasing of training data size : **cost labelisation**
- Data augmentation : solution of increasing the size of the training data by **rotating the image, flipping, scaling, shifting, etc.**
- Improve the model accuracy

# Data augmentation

- in keras, these transformations can be performed using **ImageDataGenerator**
- The function takes into account a list of arguments that can be used for preprocessing the training data.

```
data_augmentation_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),  # Flip horizontal avec probabilité 50%
    transforms.RandomRotation(degrees=30),  # Rotation aléatoire jusqu'à 30 degrés
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),  # Ajuste les couleurs
    transforms.RandomResizedCrop(size=224, scale=(0.8, 1.0)),  # Recadrage aléatoire
    transforms.ToTensor(),  # Convertit l'image en tenseur
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalisation
])
```

# PLAN

Introduction

I. Performance analysis

II. **Performances optimization : regularization**

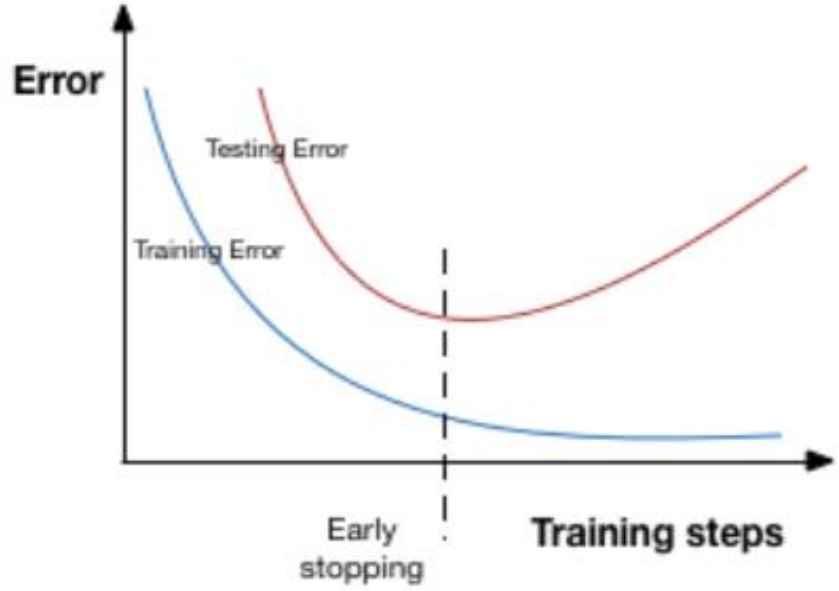
- L2 & L1 regularization
- Dropout
- Data augmentation
- **Early stopping**

III. Batch normalization

Conclusion

# Early Stopping

- A kind of cross-validation : keep part of the training set as validation set
- Once the validation performance is getting worse, we stop immediately the training.



# Early Stopping

```
early_stopping = EarlyStopping(
    monitor="val_loss",
    mode="min",
    patience=5,
    verbose=True
)

checkpoint = ModelCheckpoint(
    monitor="val_loss",
    mode="min",
    dirpath="checkpoints/",
    filename="best_model",
    save_top_k=1
)

trainer = pl.Trainer(
    max_epochs=100,
    callbacks=[early_stopping, checkpoint],
    log_every_n_steps=10
)

trainer.fit(model, train_loader, val_loader)
```

# PLAN

Introduction

I. Performance analysis

II. Performances optimization : regularization

- L2 & L1 regularization
- Dropout
- Data augmentation
- Early stopping

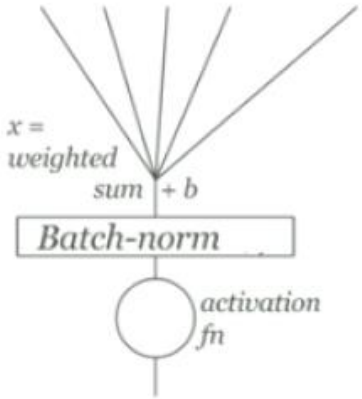
III. **Batch normalization**

Conclusion

# Why Batch Normalization ?

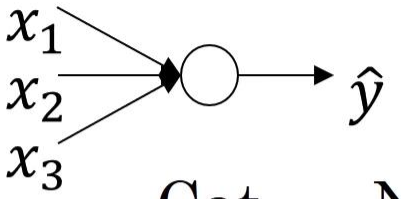
- Normalisation des activations de la couche précédente de chaque batch (sous-ensemble de données)
- Soustraction de la moyenne avent de diviser par l'écart type (standard deviation )
- Apprentissage rapide + amélioration des performances

$$\widehat{x} = \frac{x' - avg_{batch}(x)}{stdev_{batch}(x) + \epsilon}$$

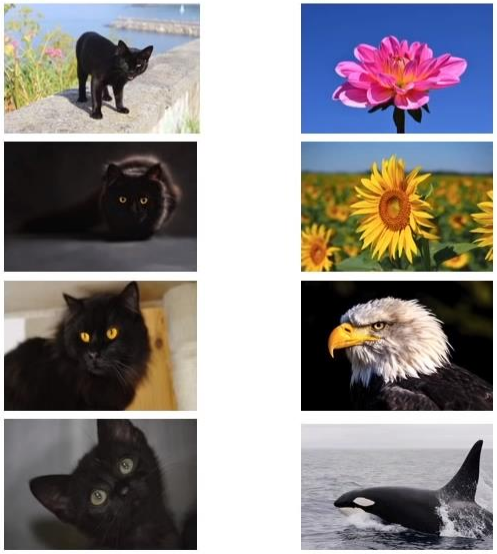


# Why Batch Normalization ?

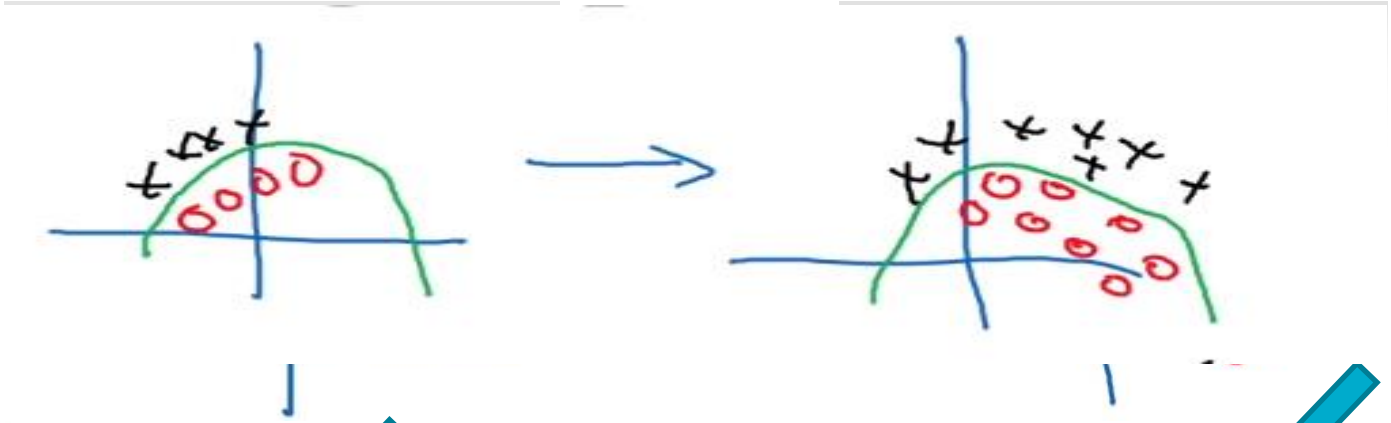
Learning on shifting input distribution



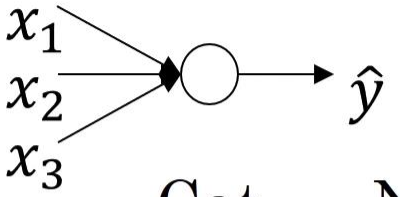

Cat $y = 1$ | Non-Cat $y = 0$



# Why Batch Normalization ?



Learning




Cat
$y = 1$

Non-Cat
$y = 0$

**Covariance Shift**

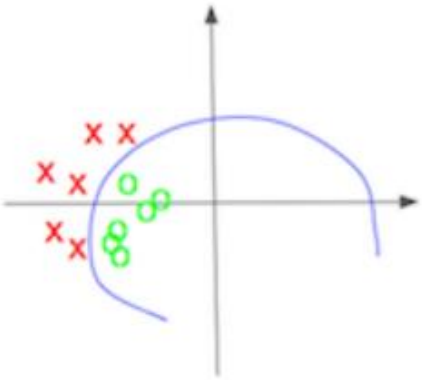
$y = 1$

$y = 0$

# Why Batch Normalization ?

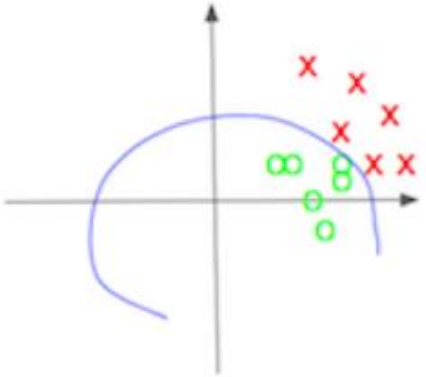
Rose
(y=1)

Not Rose
(y=0)

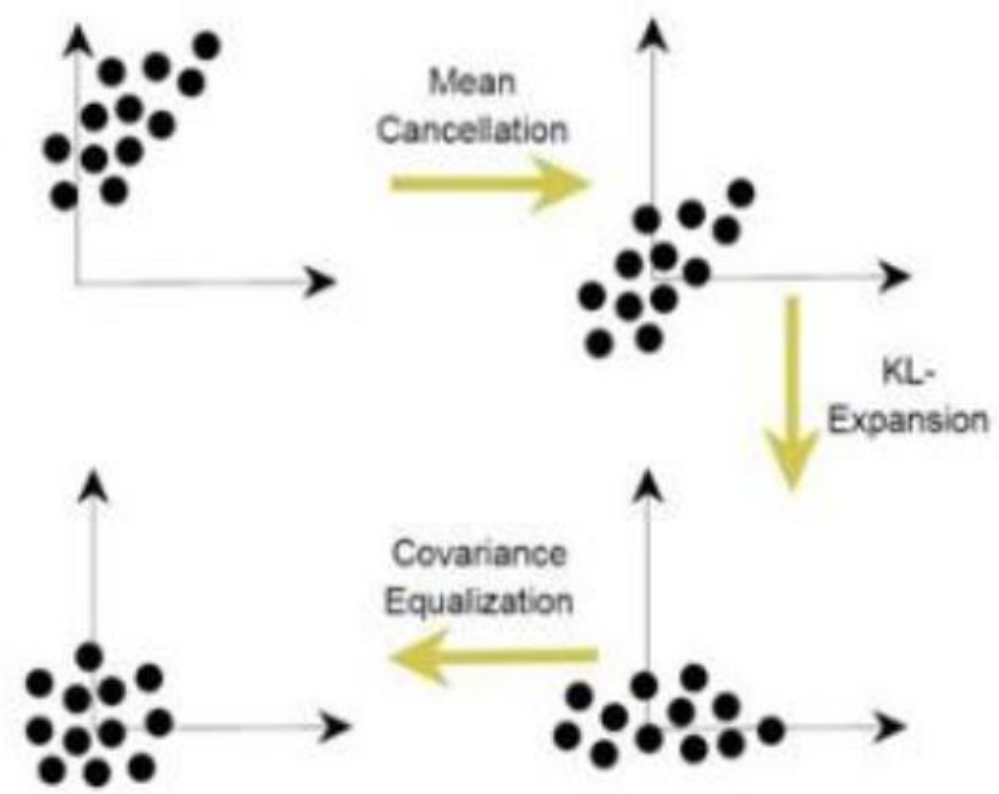


Rose
(y=1)

Not Rose
(y=0)

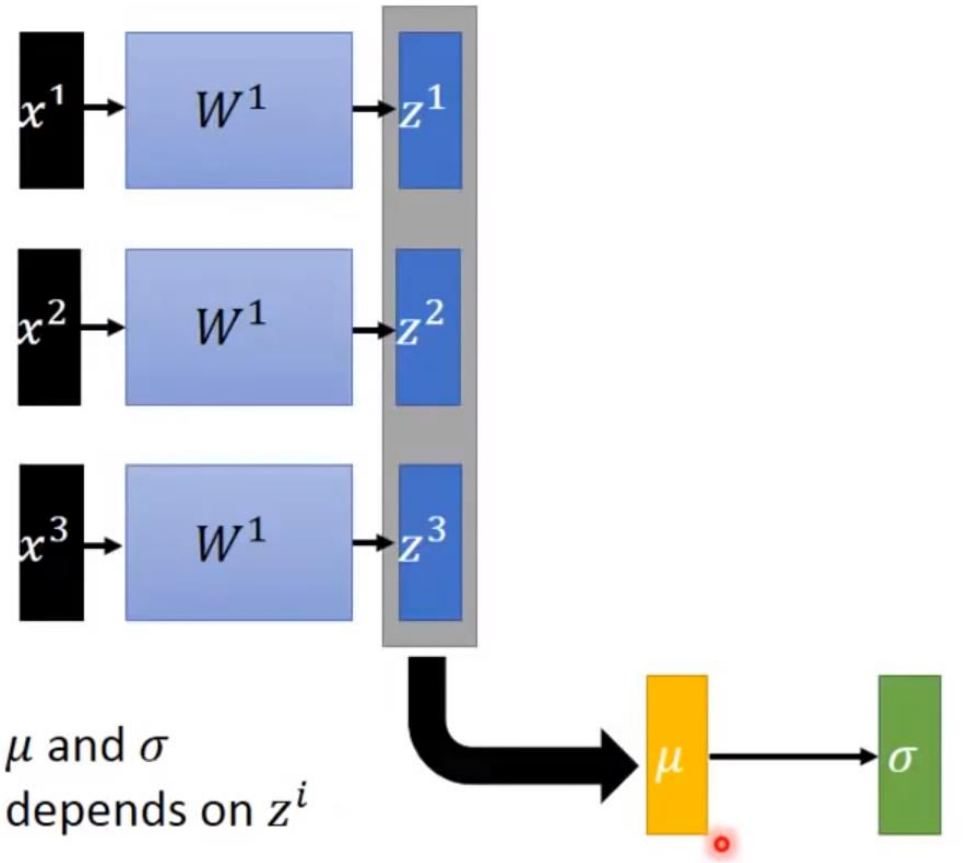


# Why Batch Normalization ?



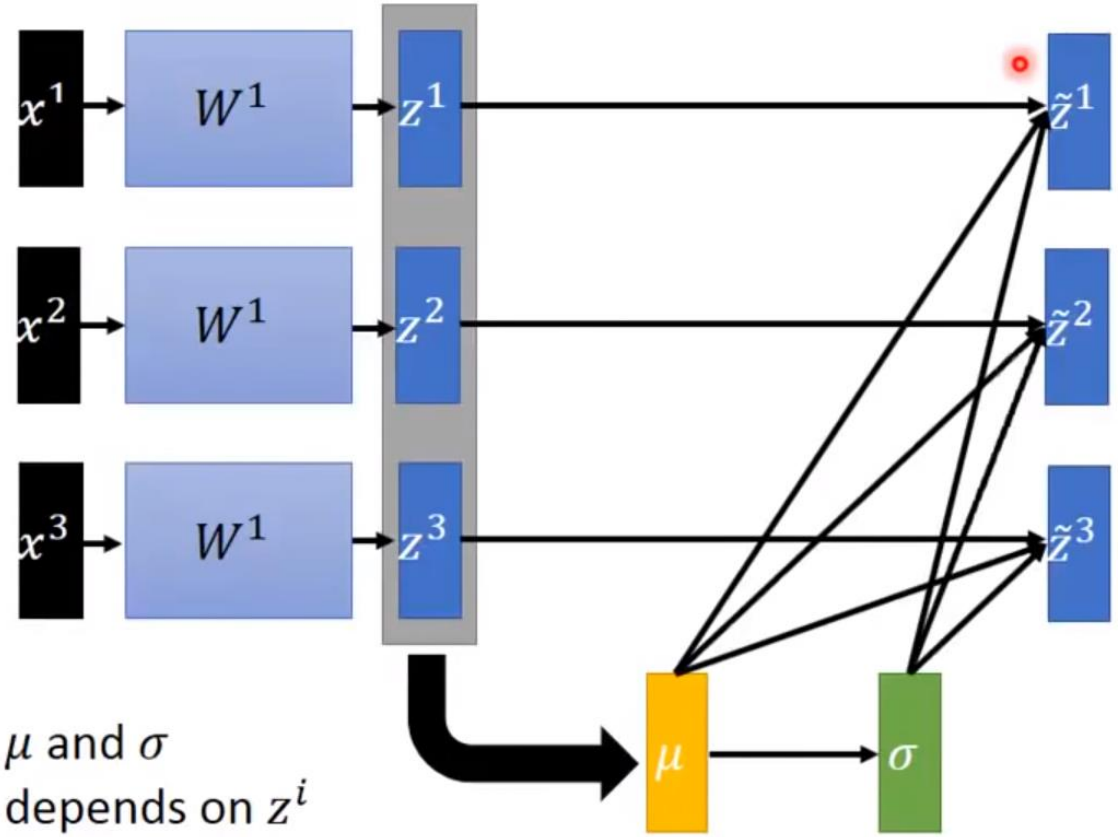
# Batch Normalization

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$



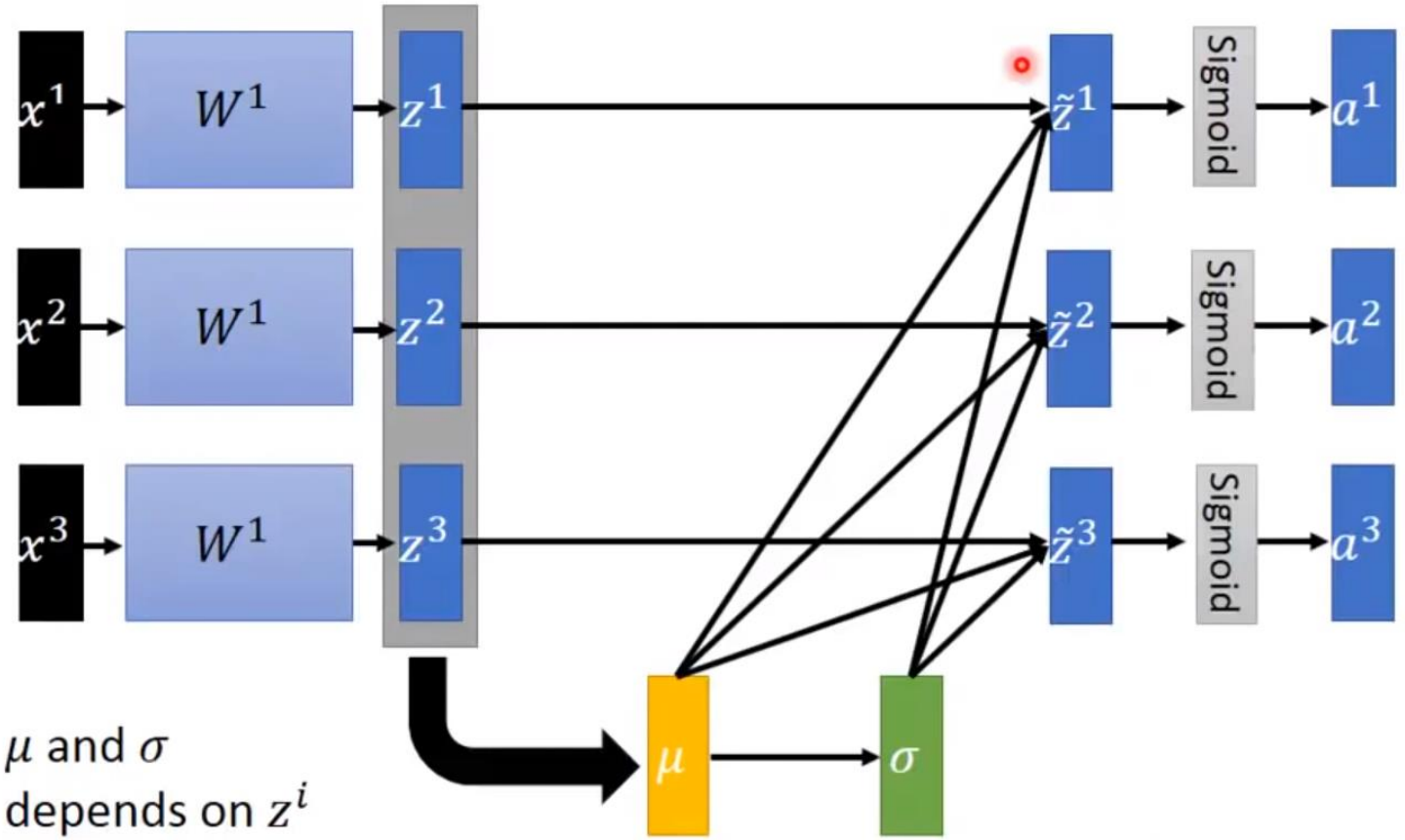
# Batch Normalization

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$



# Batch Normalization

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

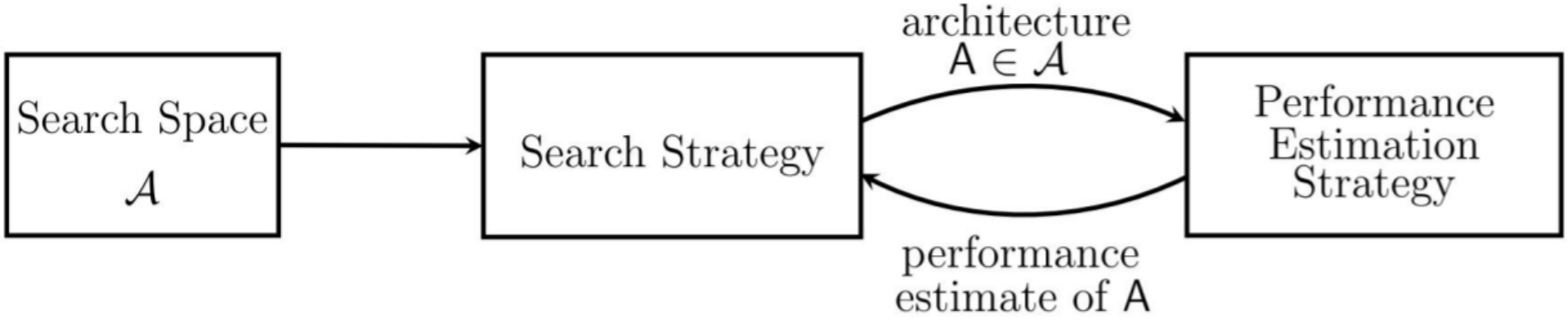


# Neural Architecture Search

- Neural Architecture Search "**NAS**": hot area of research
- **Idea :** given a dataset, define a search space of architectures, then use a strategy to the best architecture for your dataset
- Based on 3 steps : **search space, search strategy and evaluation method**

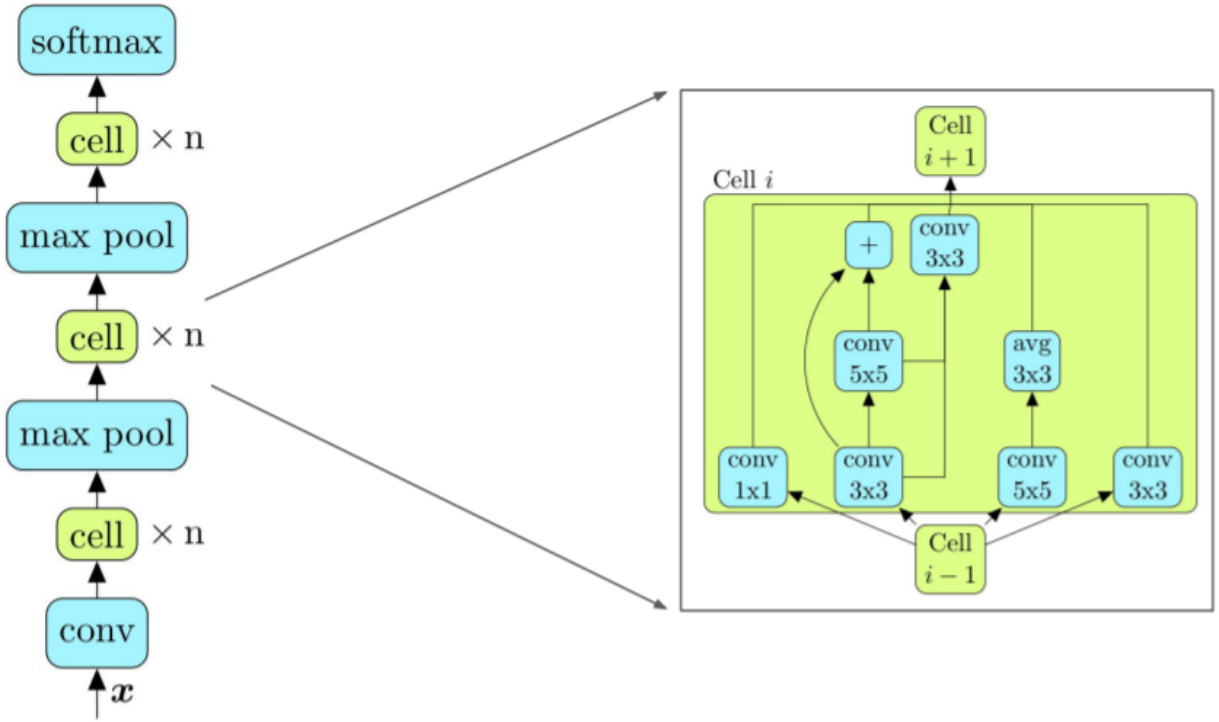
# Neural Architecture Search

- Neural Architecture Search "**NAS**": hot area of research
- **Idea :** given a dataset, define a search space of architectures, then use a strategy to the best architecture for your dataset
- Based on 3 steps : **search space, search strategy and evaluation method**



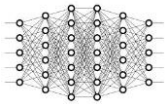
# Neural Architecture Search : example

## NAS: Cell-based Search Space




**BlockQNN**
**Zong et al. (2018)**

# Conclusion



High Bias

YES

Bigger network

Training error

Layers & architecture

NO

High Variance

YES

More data & regularization

Test error

Layers & architecture

DONE

# Thank you

# Deep Neural Network

## MultiLayer Perceptron (MLP)

Medical application

**Input :**

- **X1 :** N° of Red blood cells
- **X2 :** N° of white blood cells

**Output :**

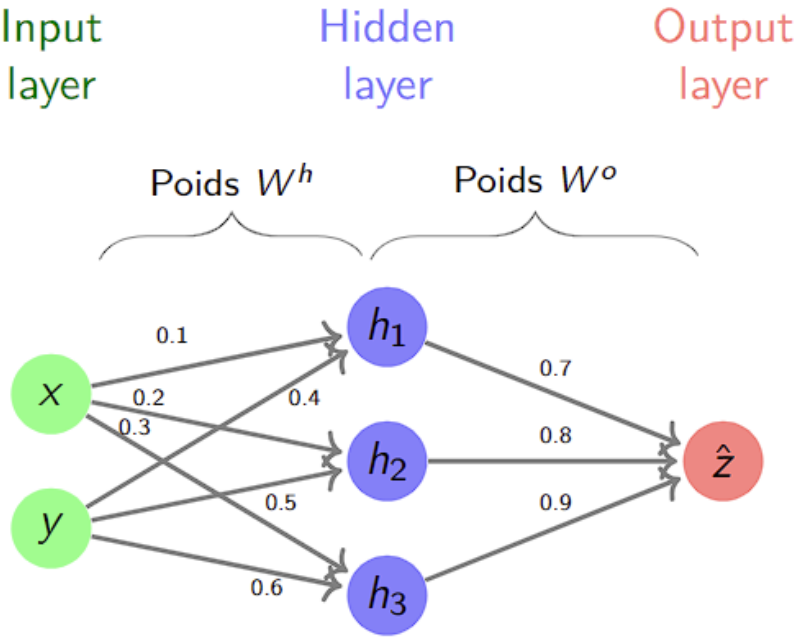
- **Y :** patient condition

**Model function :**

- **Y' = f(X,Y)**

**Training Dataset :**

| X1 | X2 | Y |
|---|---|---|
| 1 | 0 | 0,8 |
| 0,2 | 0,8 | 0,1 |
| 0,9 | 0,1 | 1 |
| ... | ... | ... |
| ... | ... | ... |
| ... | ... | ... |
| 0,8 | 0,2 | 0,9 |




Fonction d'activation dans la couche cachée :

$$\sigma(x) = \text{Sigmoïde}(x) = \frac{1}{1 + e^{-x}}$$

# Deep Neural Network

## MultiLayer Perceptron (MLP)

Medical application

**Input :**

- **X1 :** N° of Red blood cells
- **X2 :** N° of white blood cells

**Output :**

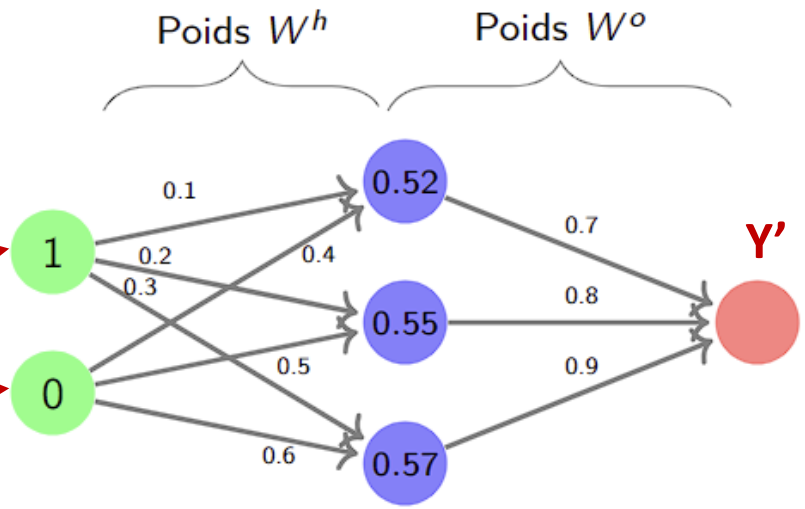
- **Y :** patient condition

**Model function :**

- **Y' = f(X,Y)**

**Training Dataset :**

| X1 | X2 | Y |
|---|---|---|
| 1 | 0 | 0,8 |
| 0,2 | 0,8 | 0,1 |
| 0,9 | 0,1 | 1 |
| ... | ... | ... |
| ... | ... | ... |
| ... | ... | ... |
| 0,8 | 0,2 | 0,9 |




1. $(1 \quad 0) \cdot \overbrace{\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{pmatrix}}^{W^h} = (0.52 \quad 0.55 \quad 0.57)$

# Deep Neural Network

## MultiLayer Perceptron (MLP)

Medical application

**Input :**

- **X1 :** N° of Red blood cells
- **X2 :** N° of white blood cells

**Output :**

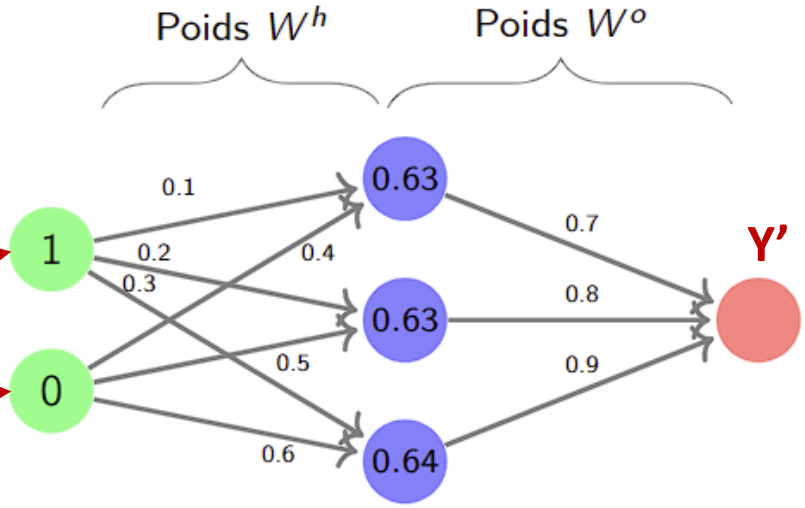
- **Y :** patient condition

**Model function :**

- **Y' = f(X,Y)**

**Training Dataset :**

| X1 | X2 | Y |
|---|---|---|
| 1 | 0 | 0,8 |
| 0,2 | 0,8 | 0,1 |
| 0,9 | 0,1 | 1 |
| ... | ... | ... |
| ... | ... | ... |
| ... | ... | ... |
| 0,8 | 0,2 | 0,9 |




$$\overbrace{\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{pmatrix}}^{W^h}$$

1. $(1 \quad 0) \cdot \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{pmatrix} = (0.52 \quad 0.55 \quad 0.57)$
2. $\sigma\left[(0.52 \quad 0.55 \quad 0.57)\right] = (0.63 \quad 0.63 \quad 0.64)$

# Deep Neural Network

## MultiLayer Perceptron (MLP)

Medical application

**Input :**

- **X1 :** N° of Red blood cells
- **X2 :** N° of white blood cells

**Output :**

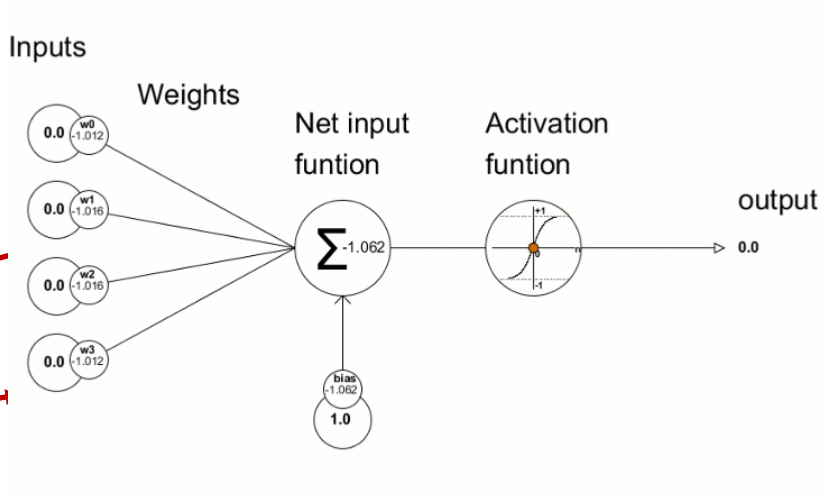
- **Y :** patient condition

**Model function :**

- **Y' = f(X,Y)**

**Training Dataset :**

| X1 | X2 | Y |
|---|---|---|
| 1 | 0 | 0,8 |
| 0,2 | 0,8 | 0,1 |
| 0,9 | 0,1 | 1 |
| ... | ... | ... |
| ... | ... | ... |
| ... | ... | ... |
| 0,8 | 0,2 | 0,9 |




$$\overbrace{W^h}$$

1. $(1 \quad 0) \cdot \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{pmatrix} = (0.52 \quad 0.55 \quad 0.57)$
2. $\sigma\left[(0.52 \quad 0.55 \quad 0.57)\right] = (0.63 \quad 0.63 \quad 0.64)$
3. $(0.63 \quad 0.63 \quad 0.64) \cdot \underbrace{(0.7 \quad 0.8 \quad 0.9)^T}_{W^o} = (1.52)$