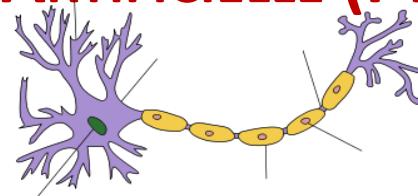
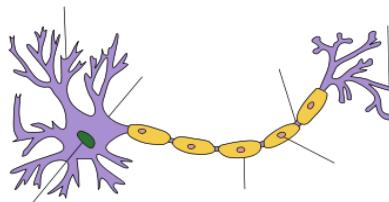




## INTELLIGENCE ARTIFICIELLE (I-ILIA-026)



## CHAPITRE 05 : L'APPRENTISSAGE PROFOND (DEEP LEARNING)



Sidi Ahmed Mahmoudi

# PLAN

## Introduction

I. Le Perceptron

II. Types de réseaux de neurones profonds (ANN, MLP, CNN, RNN, etc.)

III. Analyse des données et évaluation de modèles

IV. Réseaux de neurones convolutionnels (CNNs)

V. Outils de développement et matériel de calcul

## Conclusion

# PLAN

## Introduction

I. Le Perceptron

II. Types de réseaux de neurones profonds (ANN, MLP, CNN, RNN, etc.)

III. Analyse des données et évaluation de modèles

IV. Réseaux de neurones convolutionnels (CNNs)

V. Outils de développement et matériel de calcul

## Conclusion

# Introduction

## ARTIFICIAL INTELLIGENCE

IS NOT NEW

### ARTIFICIAL INTELLIGENCE

Any technique which enables computers to mimic human behavior



1950's

1960's

1970's

1980's

### MACHINE LEARNING

AI techniques that give computers the ability to learn without being explicitly programmed to do so



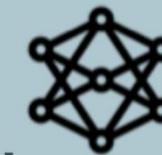
1990's

2000's

2010s

### DEEP LEARNING

A subset of ML which make the computation of multi-layer neural networks feasible

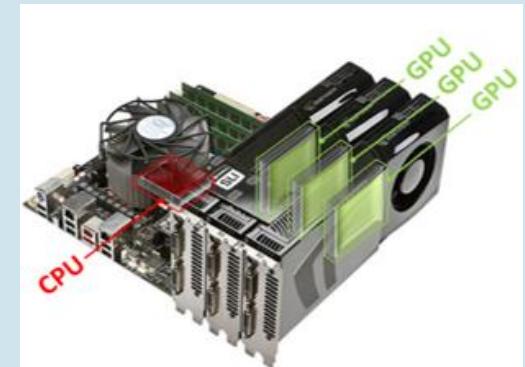


ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved. |

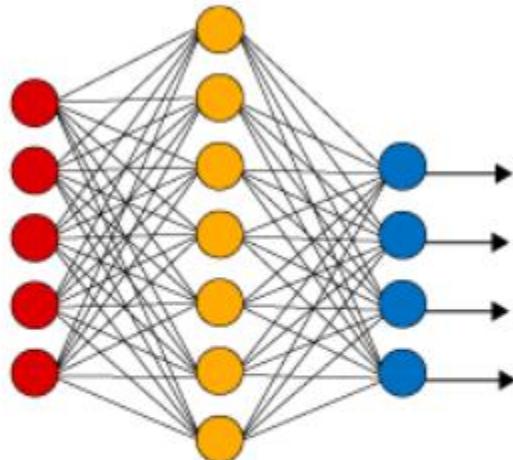
# Introduction

- La forte émergence du Deep Learning est due principalement à :
    - Disponibilité et gestion de très grands jeux de données (Big Data)
    - Enormes capacités de calcul (HPC, GPU, cluster, etc.)
    - Nouveaux modèles d'apprentissage très flexibles permettant de résoudre différents problèmes



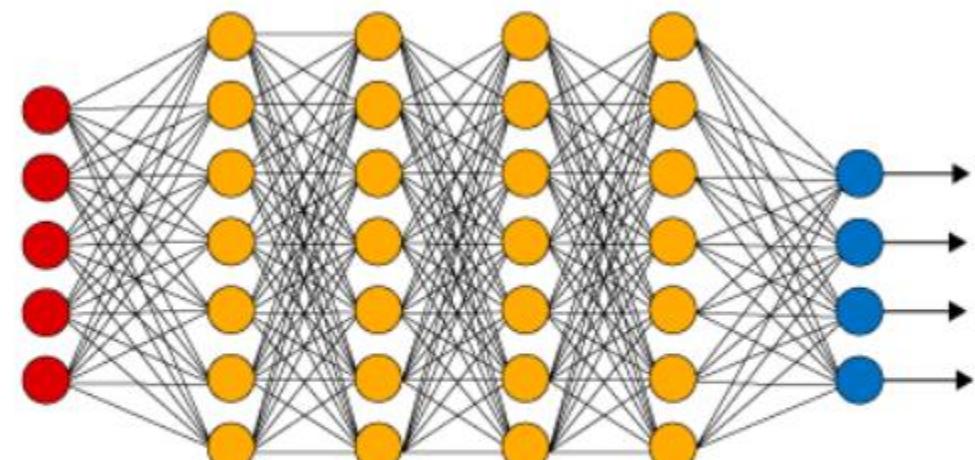
# Introduction

Réseau de neurons non profond



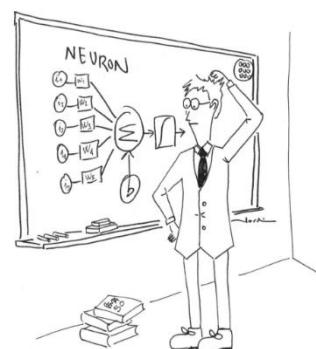
● Input Layer

Réseau de neurons profond



● Hidden Layer

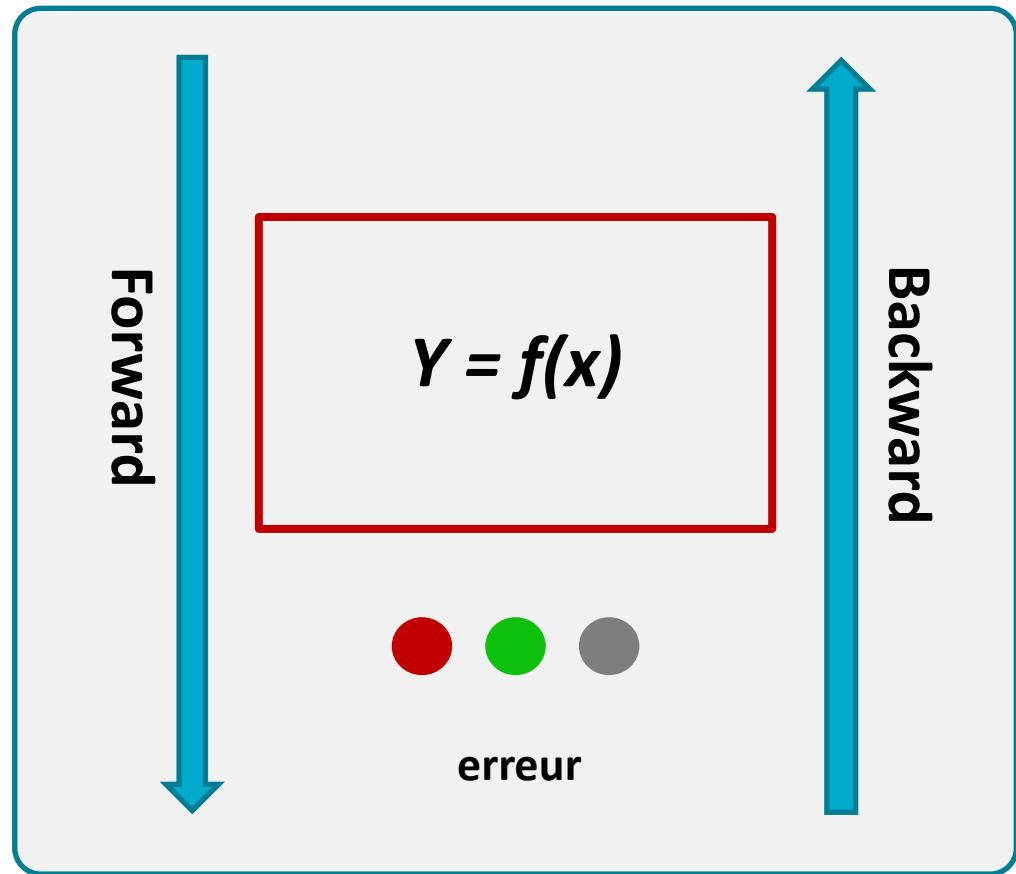
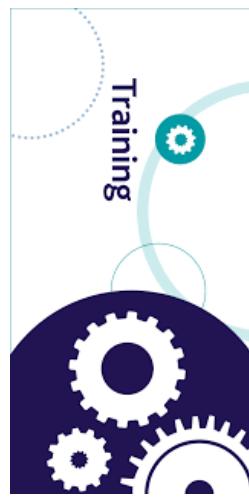
● Output Layer



# Exemple



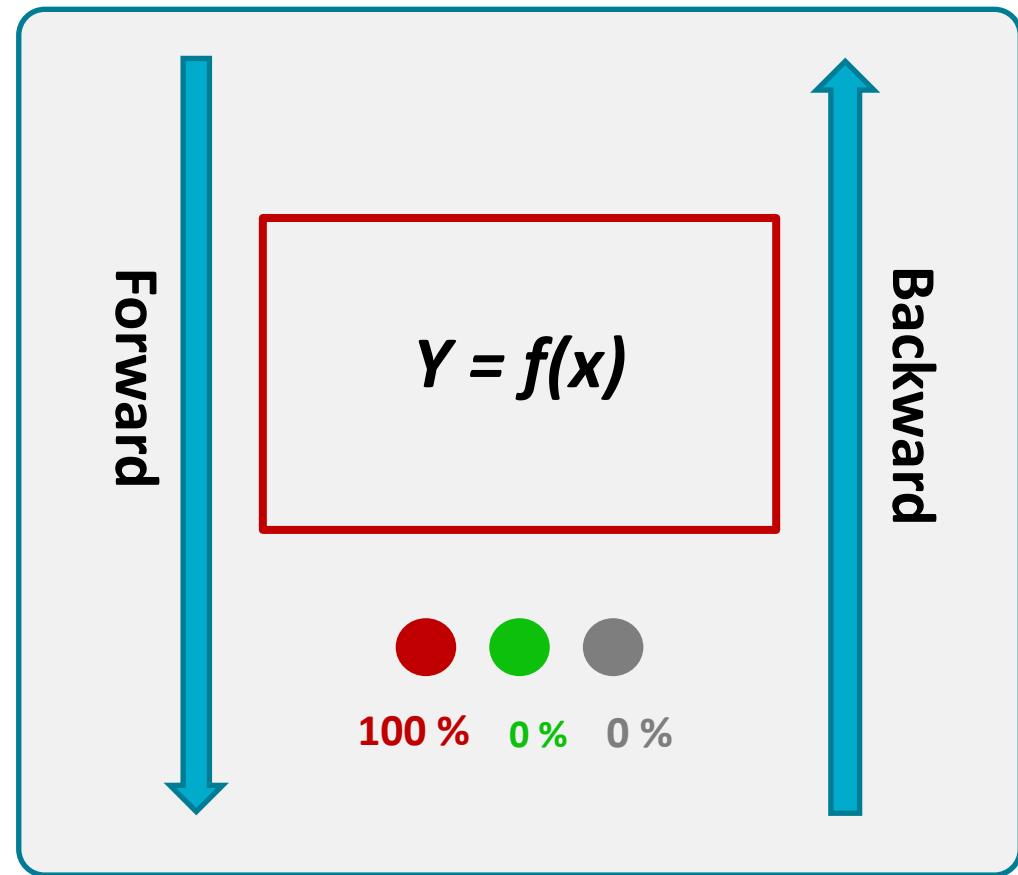
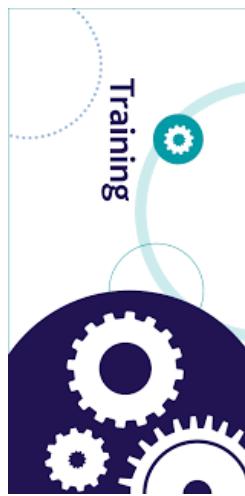
Data



# Exemple



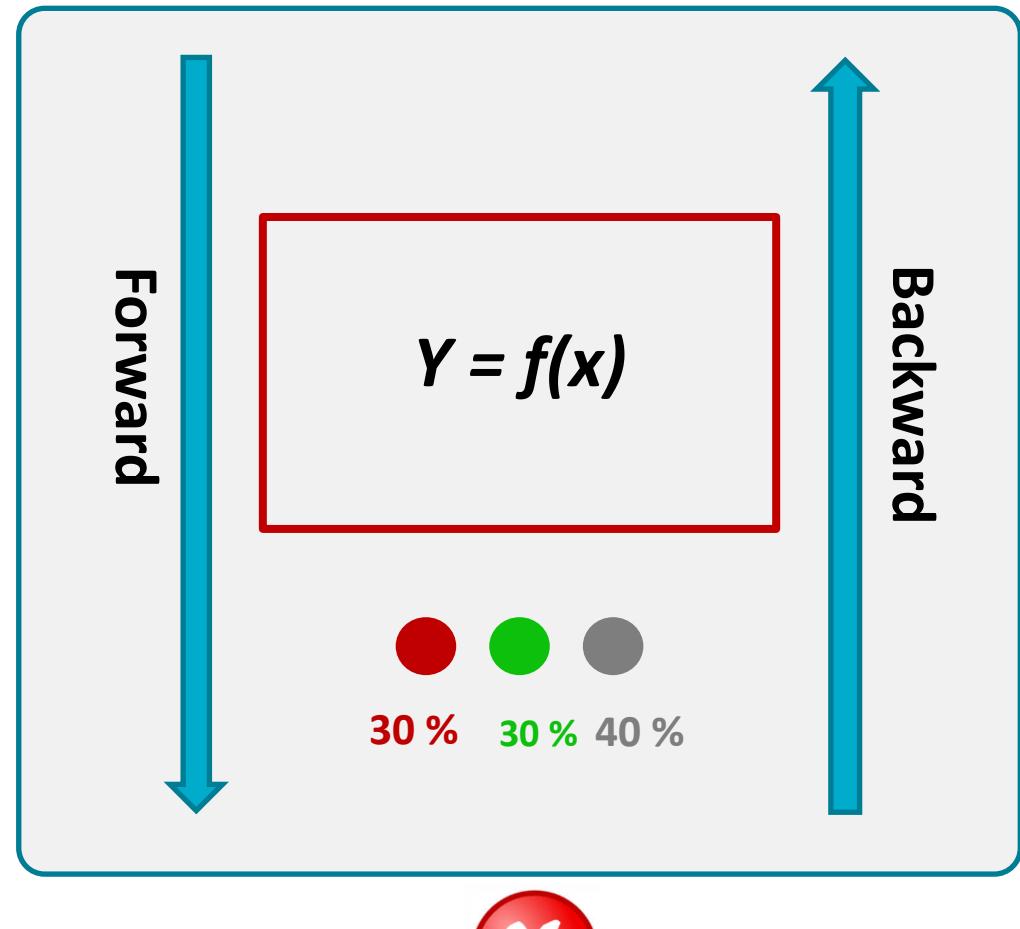
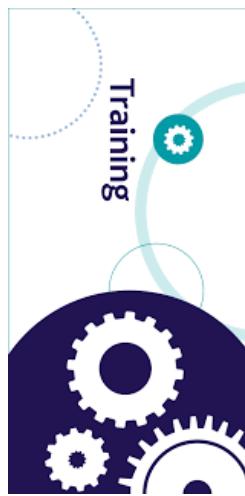
Data



# Exemple



Data



# PLAN

Introduction

I. Le Perceptron

II. Types de réseaux de neurones profonds (ANN, MLP, CNN, RNN, etc.)

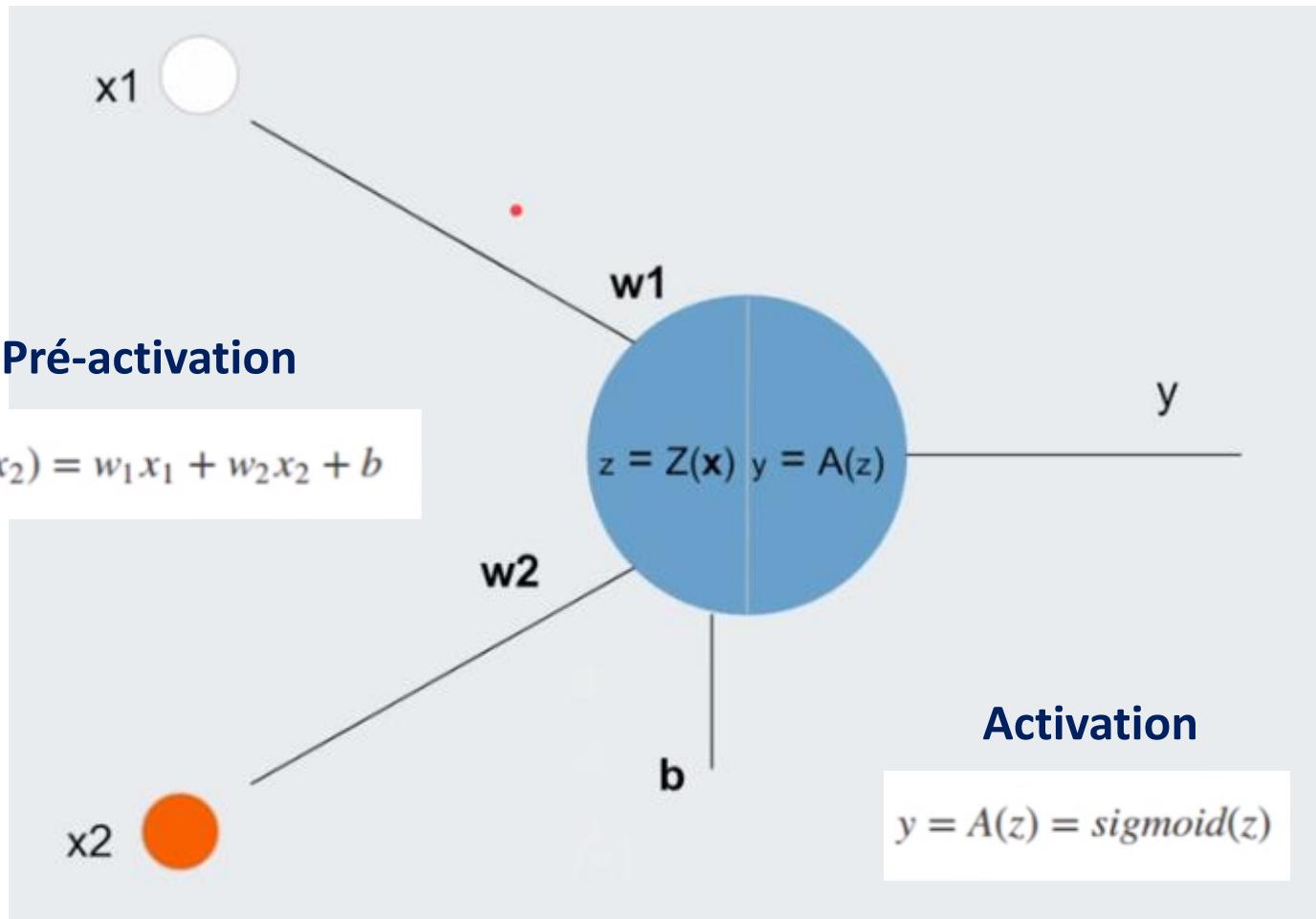
III. Analyse des données et évaluation de modèles

IV. Réseaux de neurones convolutionnels (CNNs)

V. Outils de développement et matériel de calcul

Conclusion

# Perceptron



# Perceptron

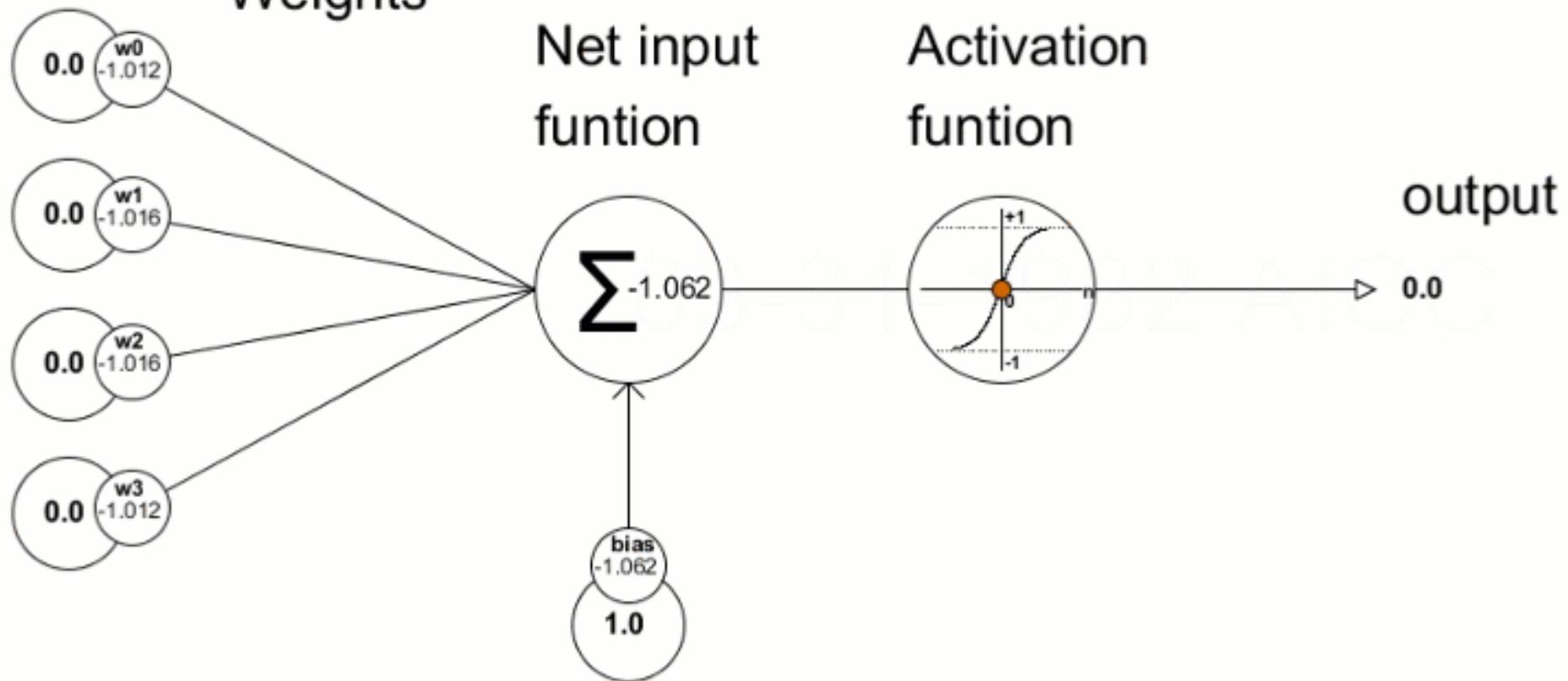
Inputs

Weights

Net input  
funtion

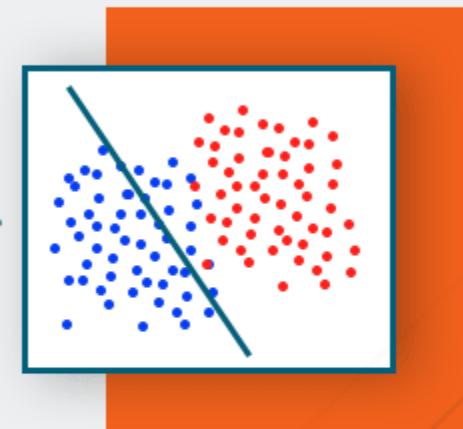
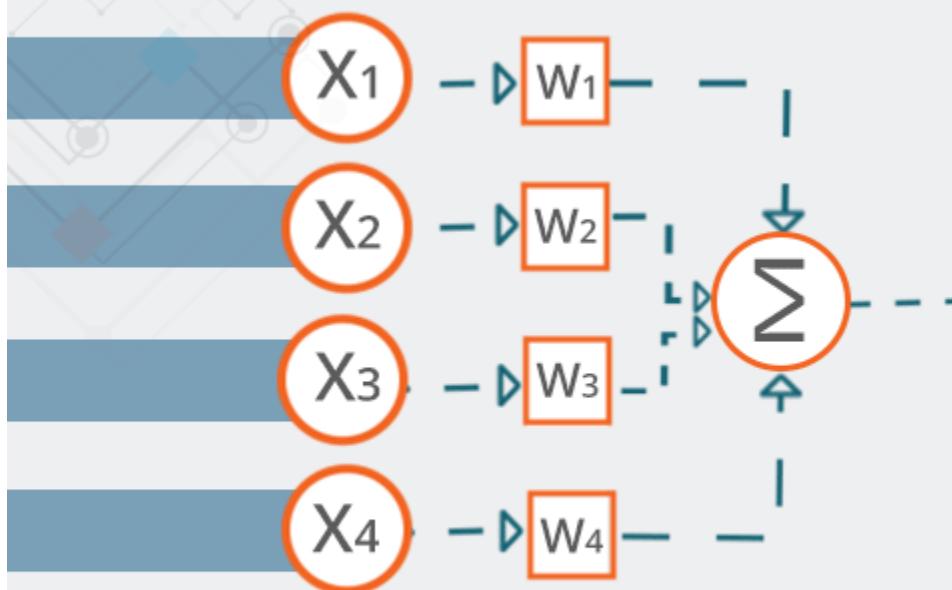
Activation  
funtion

output



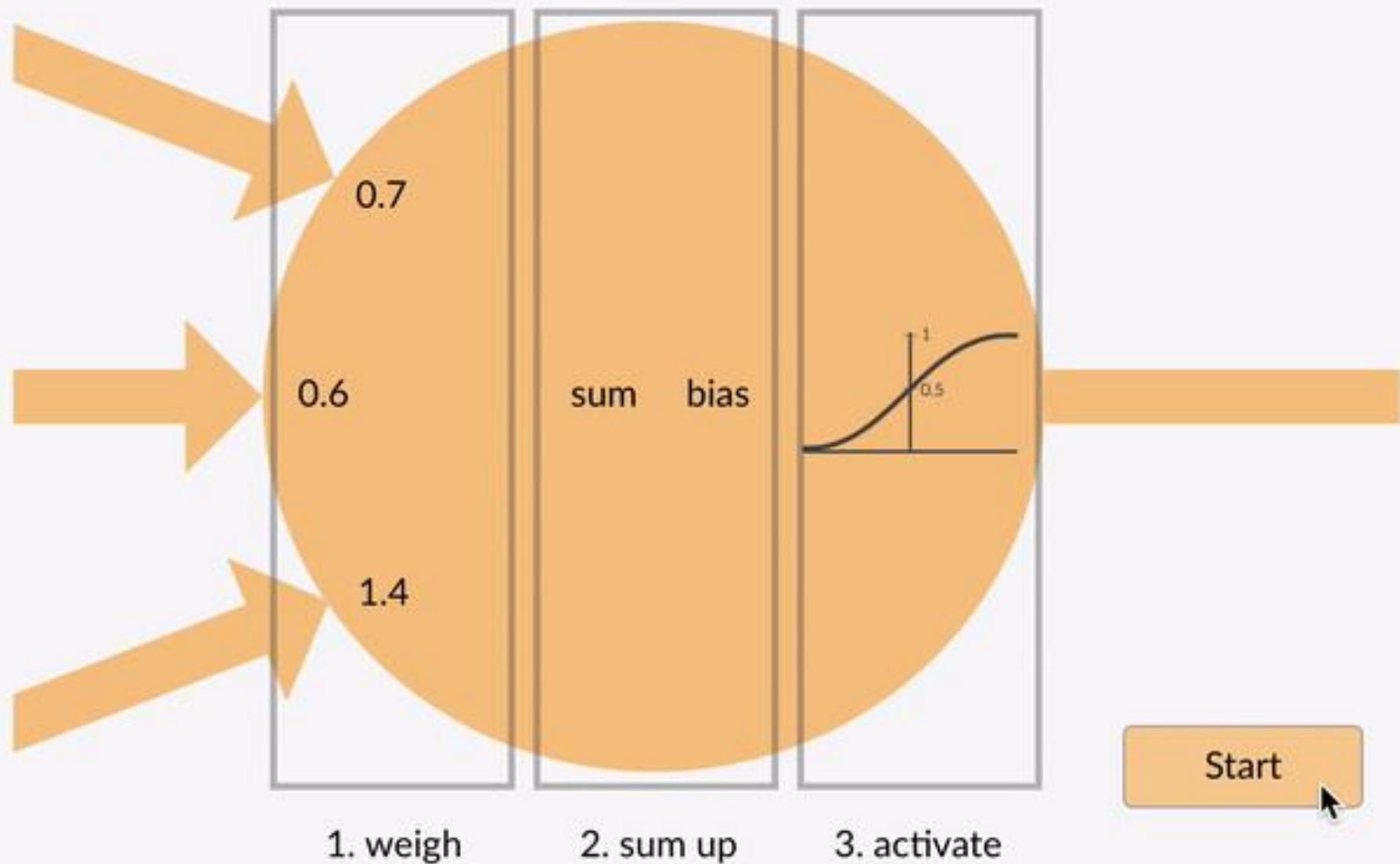
# Perceptron

edureka!



## Perceptron Learning Algorithm

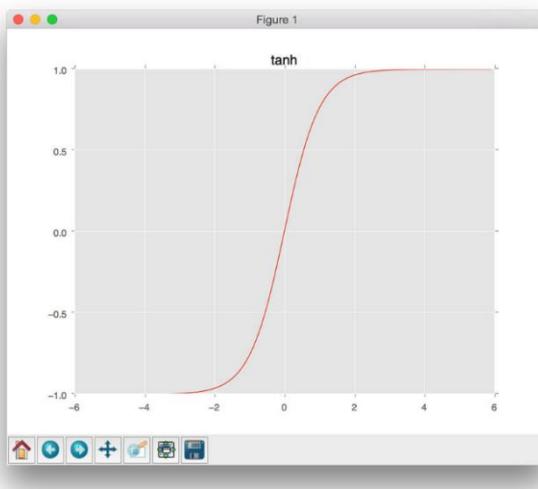
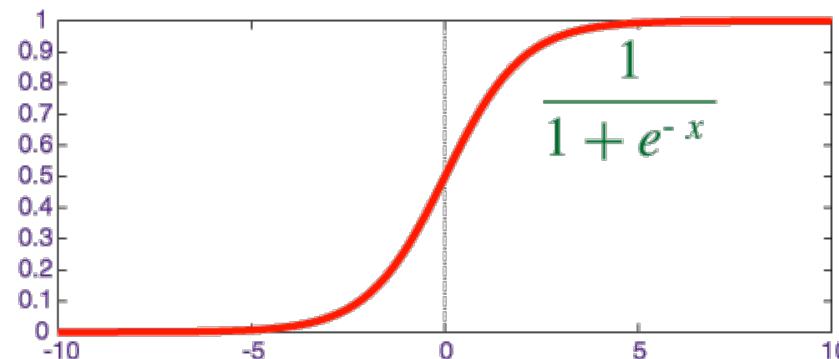
# Fonctions d'activation et non linéarité



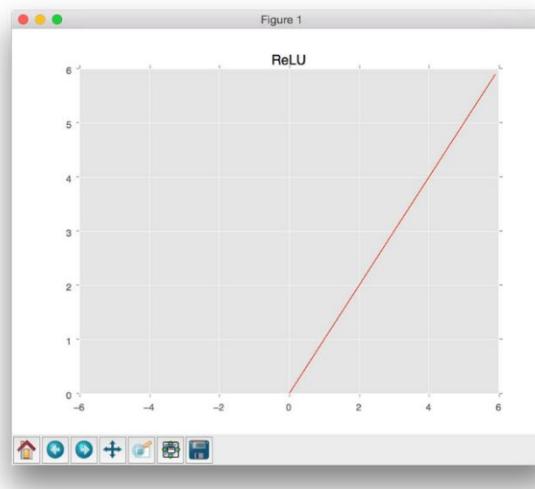
# Fonctions d'activation

- Non linéarité: apprendre des motifs complexes

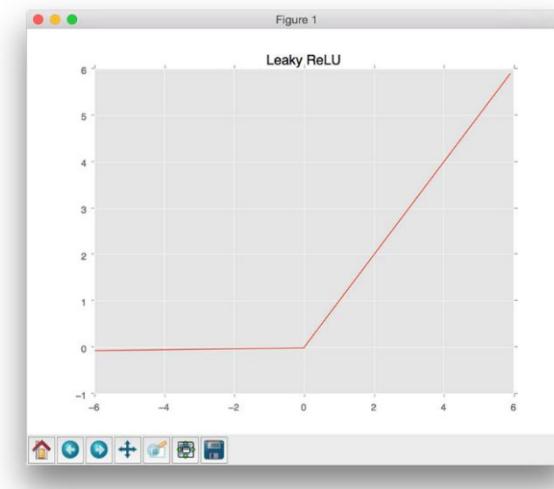
Sigmoid



tanh



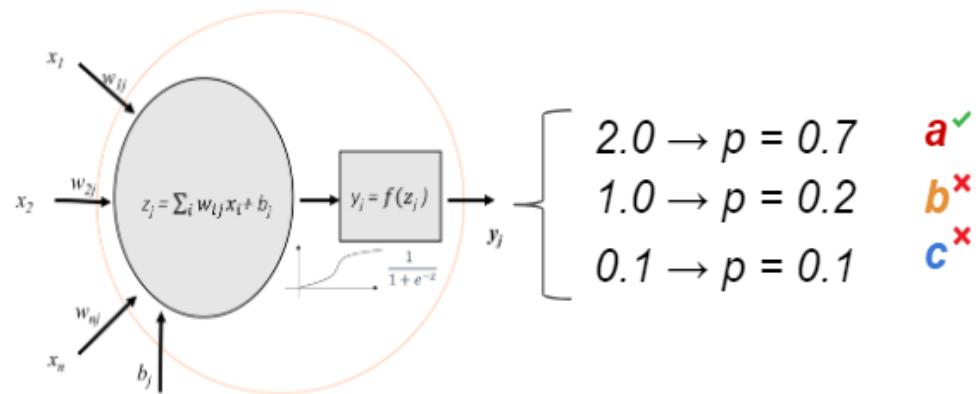
Relu



Leaky Relu

# Perceptron (fonction softmax)

- Softmax: traduire les scores de la dernière couche en probabilités



**Softmax :**  $S(y_i) = \frac{e^{y_i}}{\sum_i e^{y_i}}$

- Une entrée ne peut pas avoir qu'un seul label possible (a)
- But : avoir une probabilité proche de 1 pour cette classe  
probabilité proche de 0 pour les autres classes

# Fonctions d'activation : problèmes rencontrés

## Vanishing Gradient:

- gradients deviennent trop petits voire nuls
- Apprentissage très lent voire non évolutif
- Solution : **Relu, Leaky Relu, etc.**

## Exploding Gradient:

- gradients trop grands → apprentissage instable
- Solution : **Régularisations, Batch-normalisation, etc.**

## Dead neurones (neurones morts) :

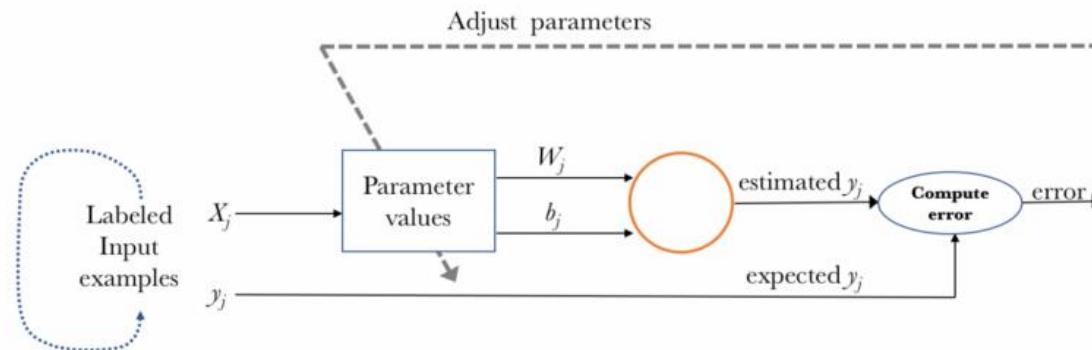
- Avec Relu: certaines unités produire que des zéros
- Solution : **Leaky Relu, Parametric Relu.**

# Fonctions d'activation : recommandation de choix

Fonction	Formule	Avantages	Inconvénients
Sigmoïde	$\frac{1}{1+e^{-x}}$	Bon pour la probabilité ( entre 0 et 1).	Vanishing gradient, non centrée en zéro.
Tanh	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	Centrée en zéro, meilleur que Sigmoïde.	Vanishing gradient.
ReLU	$\max(0, x)$	Rapide, évite en partie le vanishing gradient.	Neurones morts (pour $x < 0$ ).
Leaky ReLU	$x \text{ si } x > 0, 0.01x \text{ sinon}$	Corrige le problème des neurones morts	Paramètre $\alpha$ à régler
Softmax	$\frac{e^{x_i}}{\sum e^{x_j}}$	Convertit en probabilités (utile pour classification)	Sensible aux valeurs extrêmes.

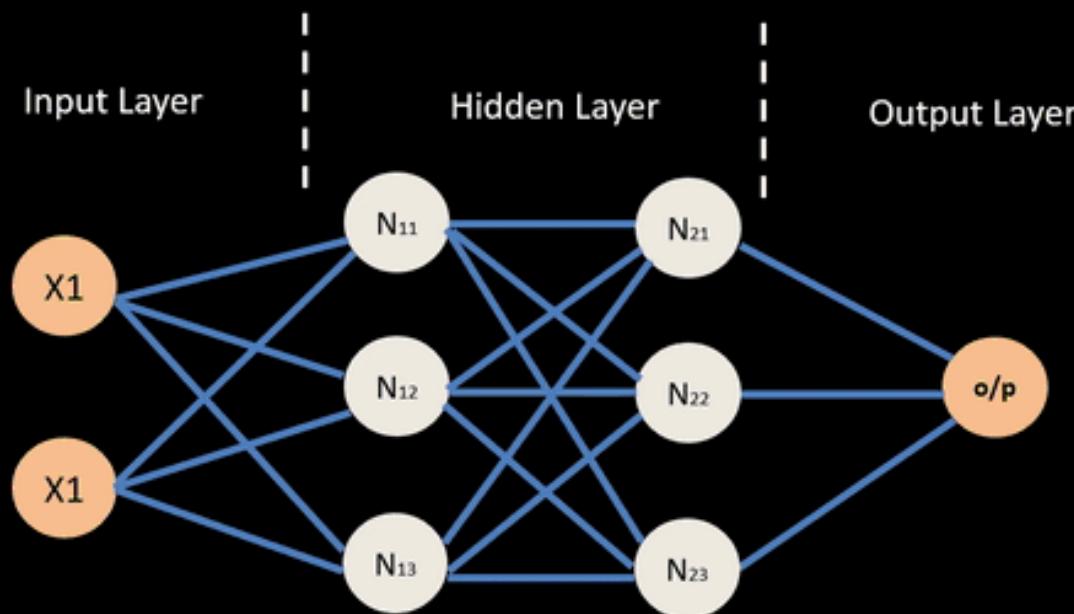
# Comment ça marche ?

- 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.

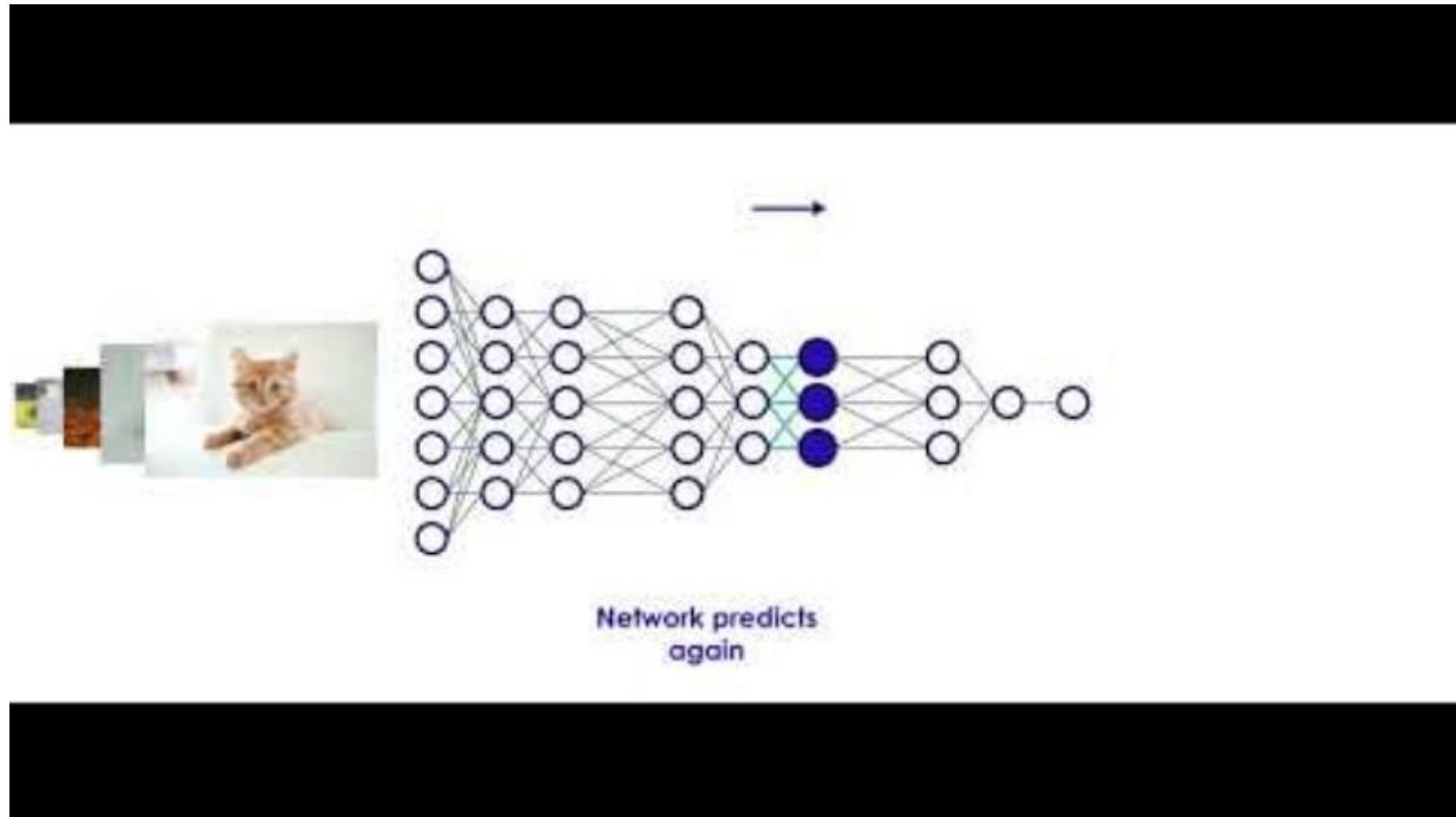


# Backpropagation process

## Neural Network – Backpropagation



# Neural network training : summary



# PLAN

Introduction

I. Le Perceptron

II. Types de réseaux de neurones profonds (ANN, MLP, CNN, RNN, etc.)

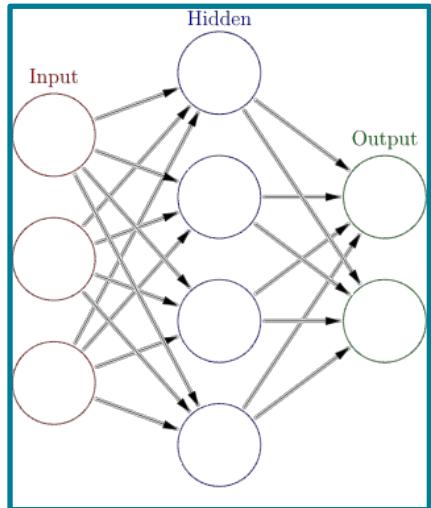
III. Analyse des données et évaluation de modèles

IV. Réseaux de neurones convolutionnels (CNNs)

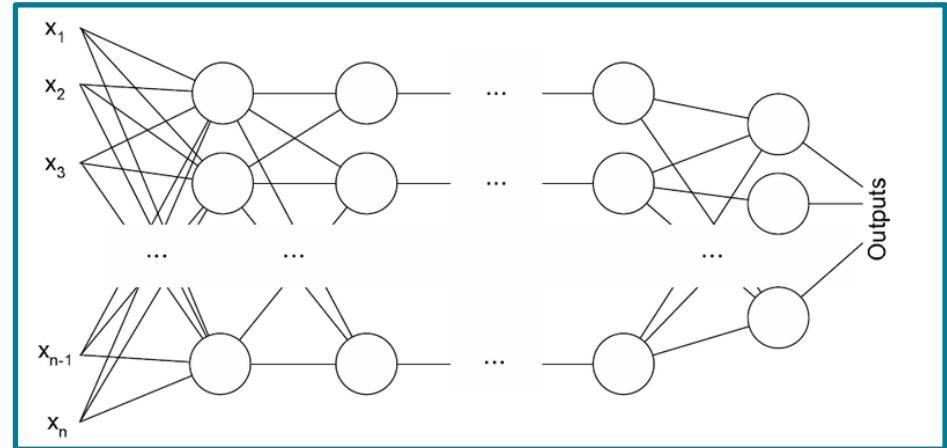
V. Outils de développement et matériel de calcul

Conclusion

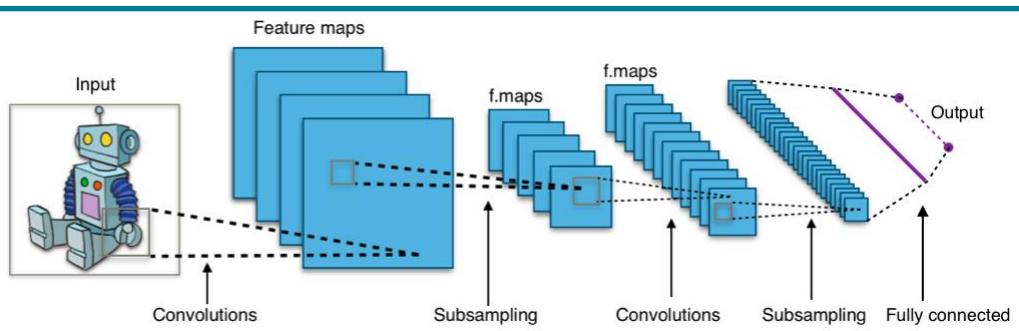
# Types de réseaux de neurones profonds



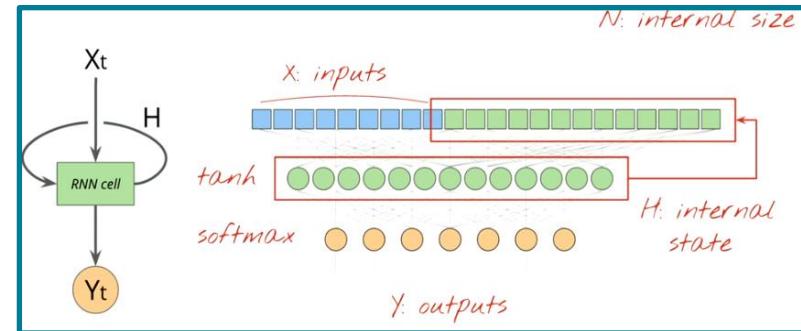
ANN



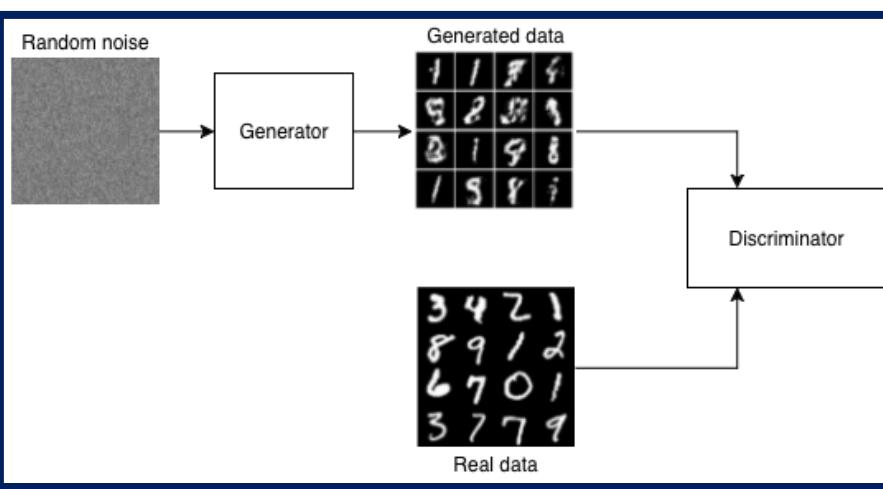
MLP (Multilayer Perceptron)



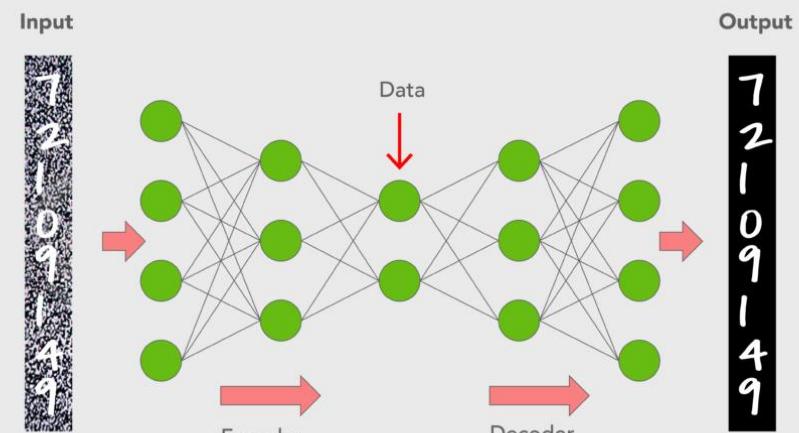
CNN



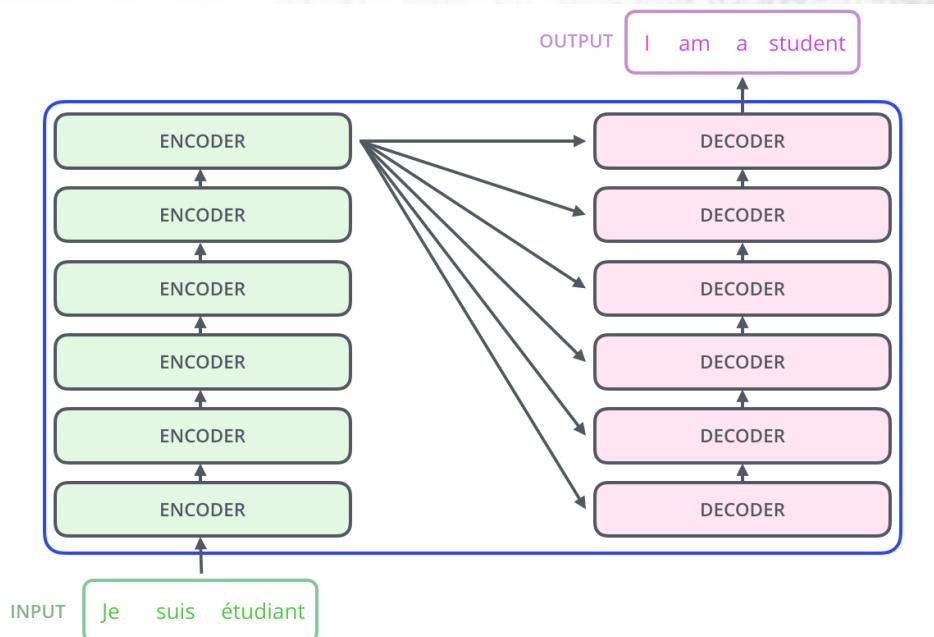
RNN



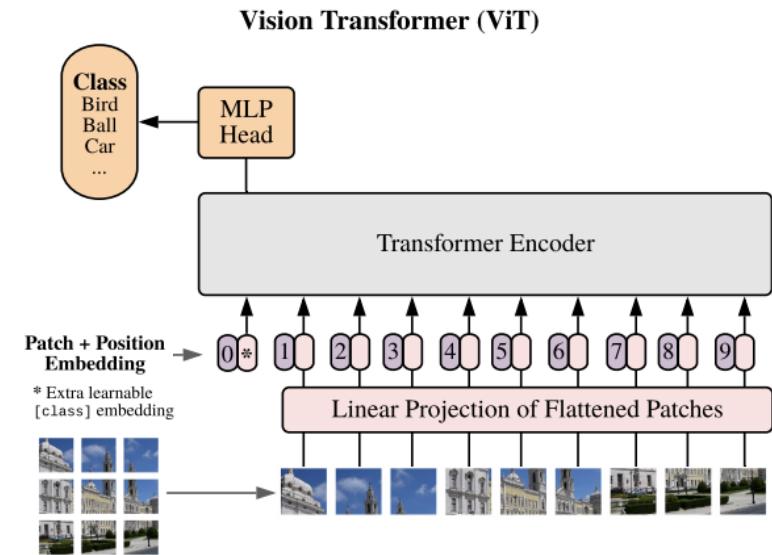
## GAN



## Auto encoders



## Transformers



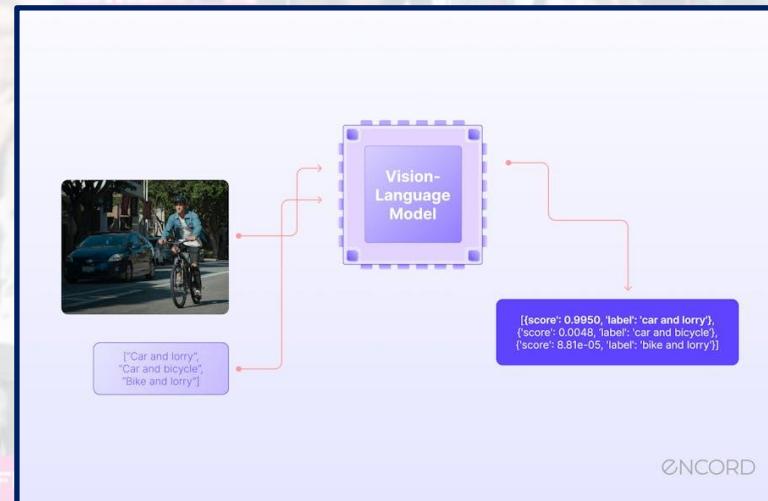
## Vision Transformers

## Models size comparison of GPT models

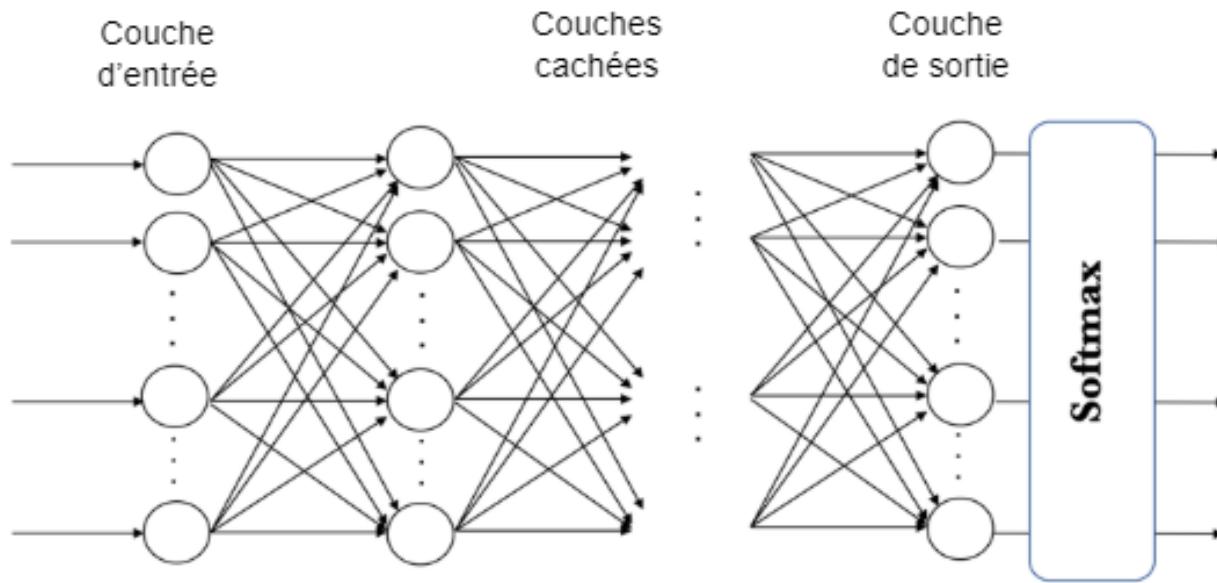
	Parameters	Decoder layers	Context lenght	Hidden layer size
GPT-1	117 million	12	512	768
GPT-2	1.5 billion	48	1024	1600
GPT-3	175 billion	96	2048	12288
GPT-4	1.76 trillion	120	8000*	20k*

\*Data subject to confirmation by OpenAI. Last updated: July 2023.

## VLMs



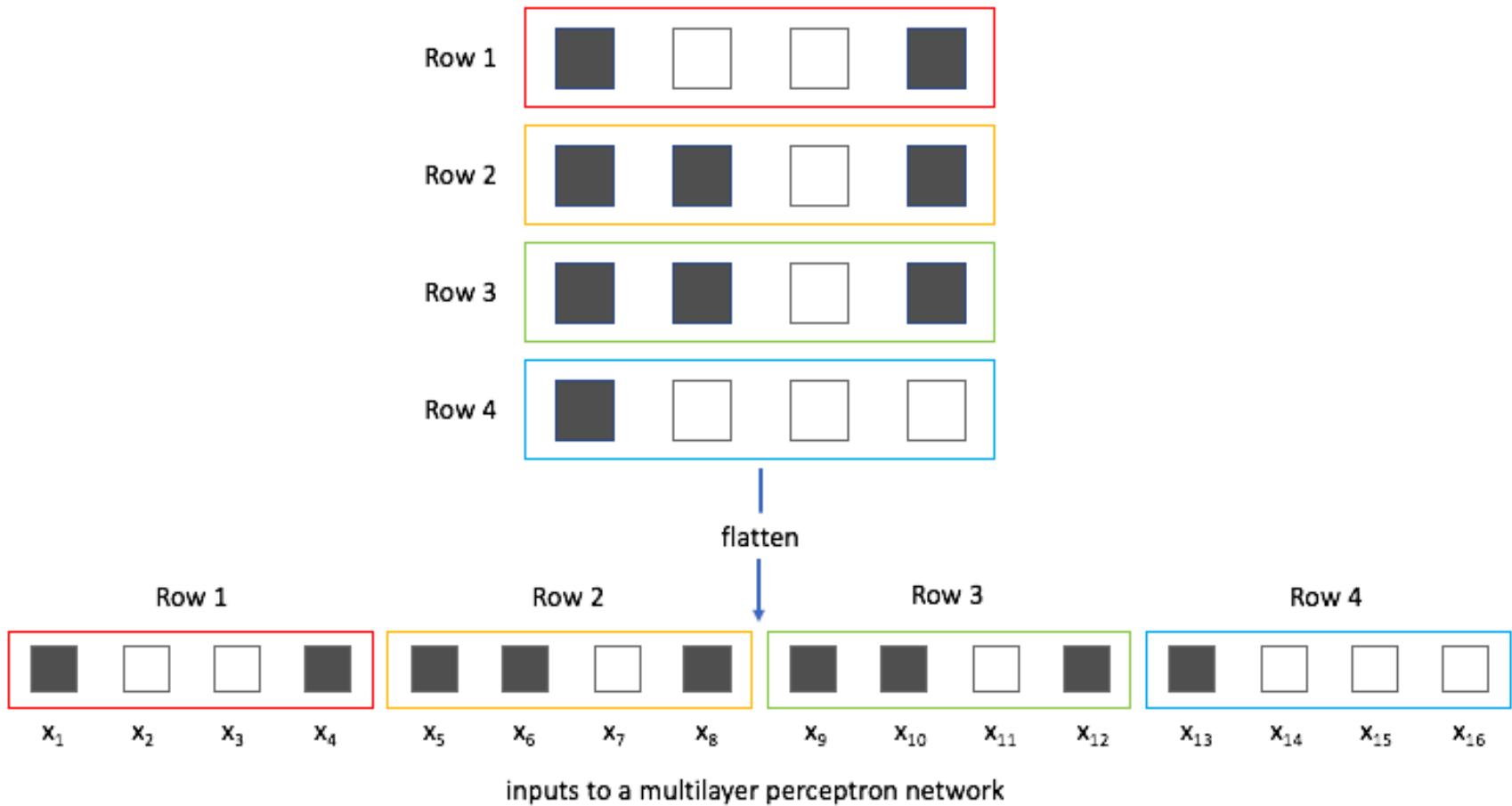
# Multilayer Perceptron (MLP)



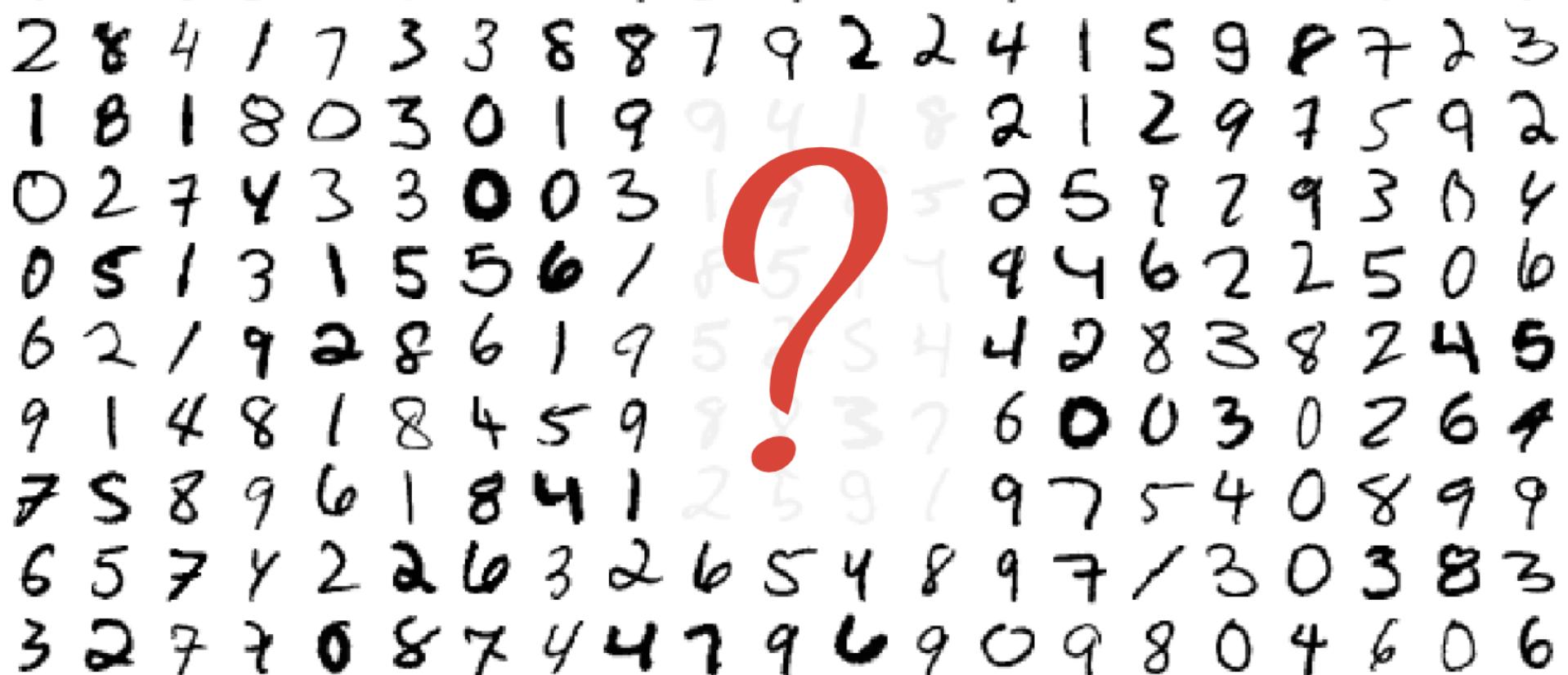
- **MLP** : Couche d'entrée + 1 ou plusieurs couches cachées + couche de sortie
- 1 couche = plusieurs perceptrons
- Deep Learning => beaucoup de couches cachées

# Multilayer Perceptron (MLP)

Une image

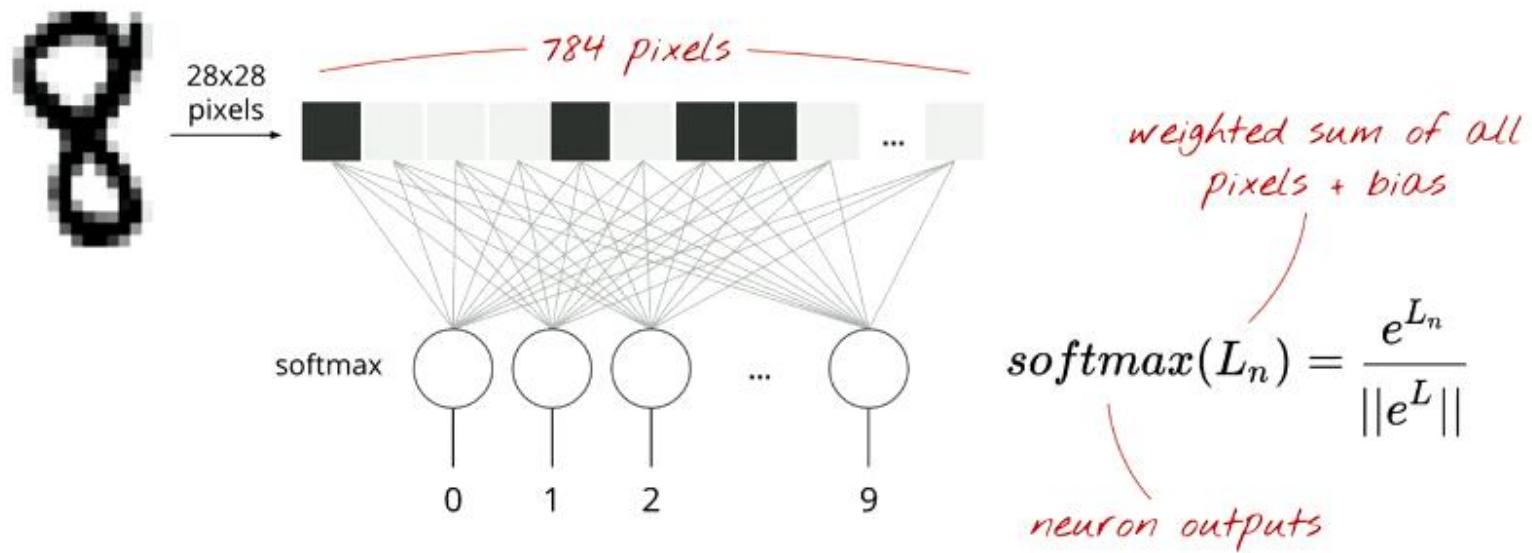


# Prenons un exemple



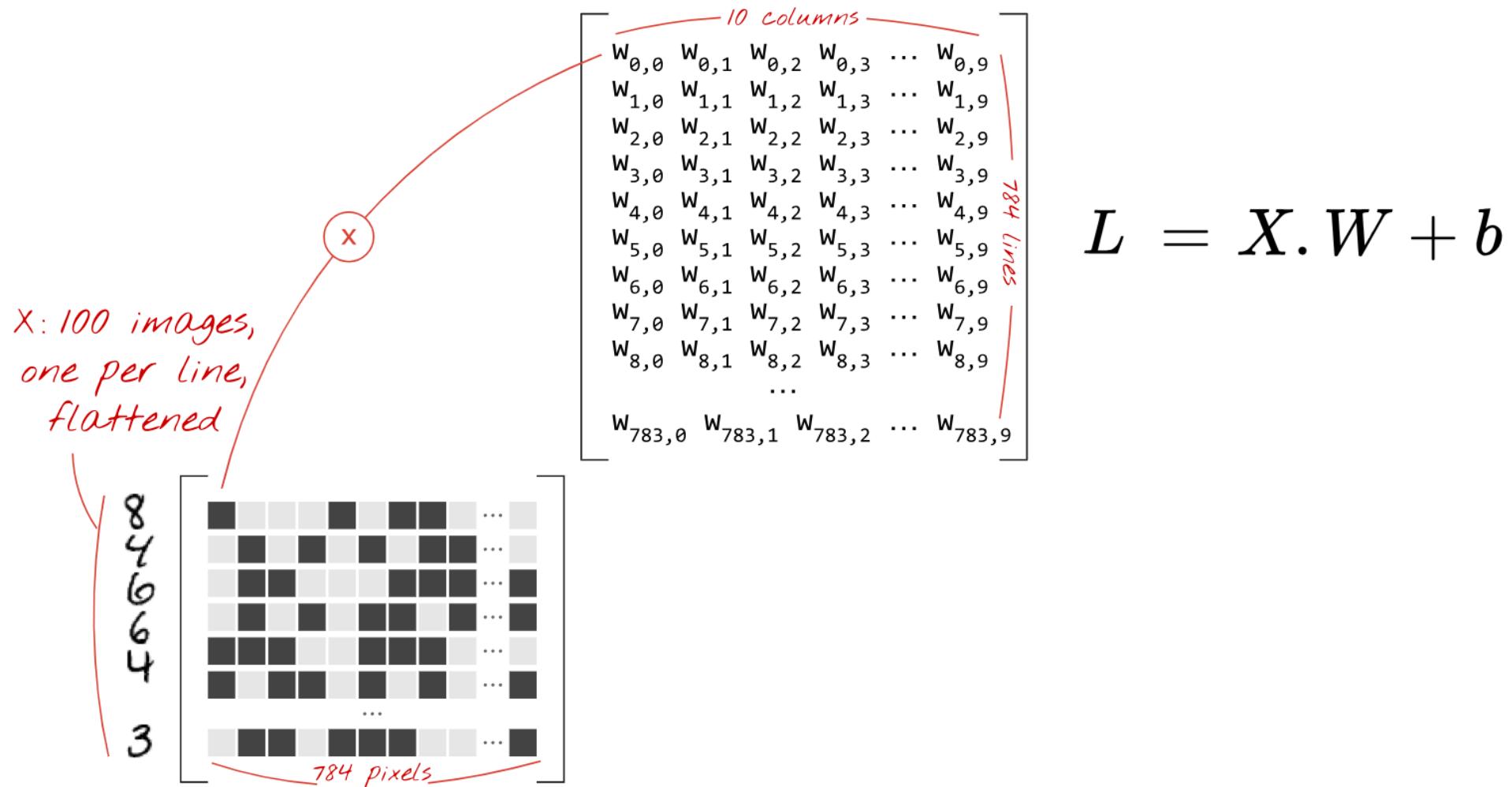
MNIST : Mixed National Institute of Standards and Technology [1]

# Prenons un exemple



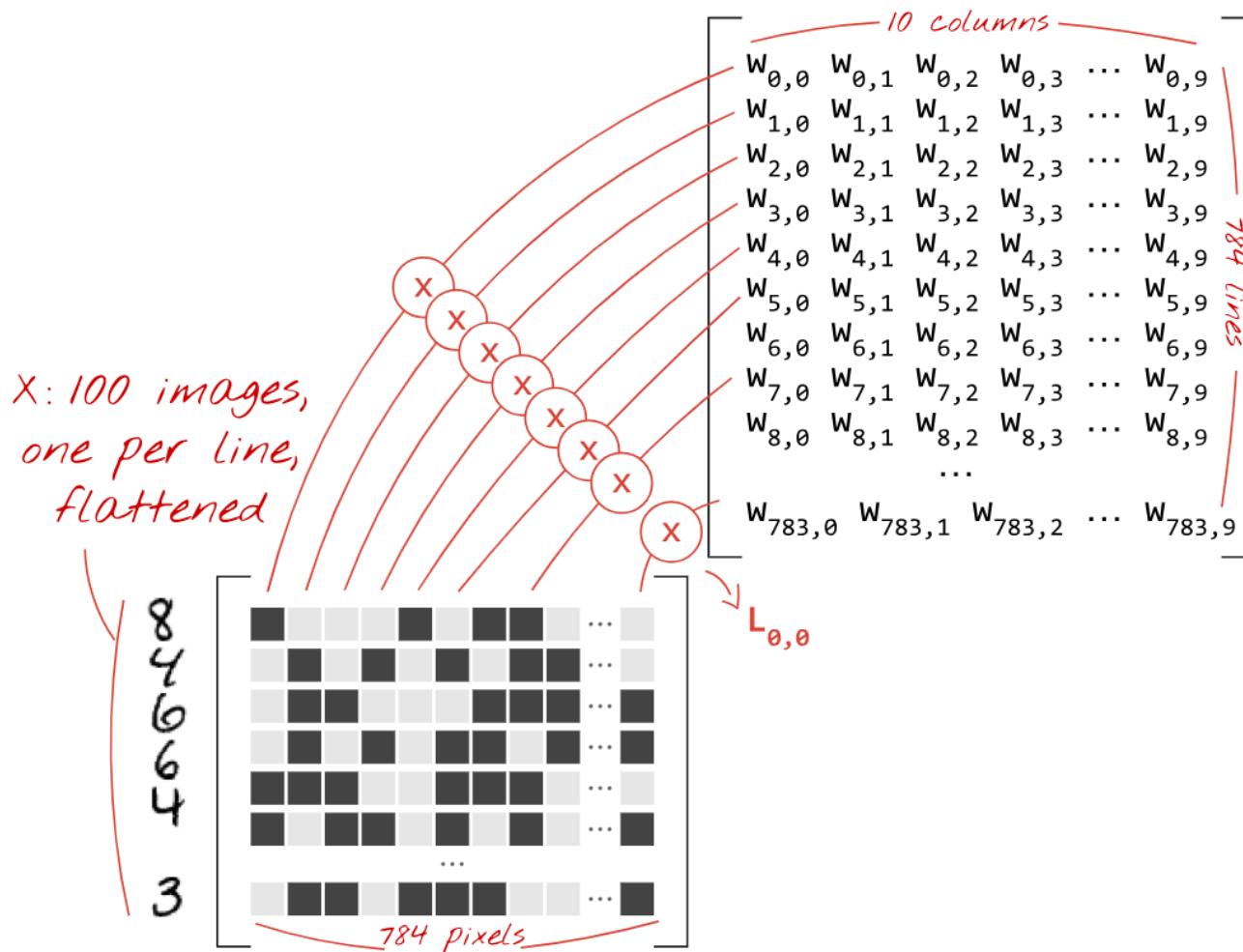
## Modèle simple : classification avec SoftMax [2]

# Prenons un exemple



Modèle simple : classification avec SoftMax [2]

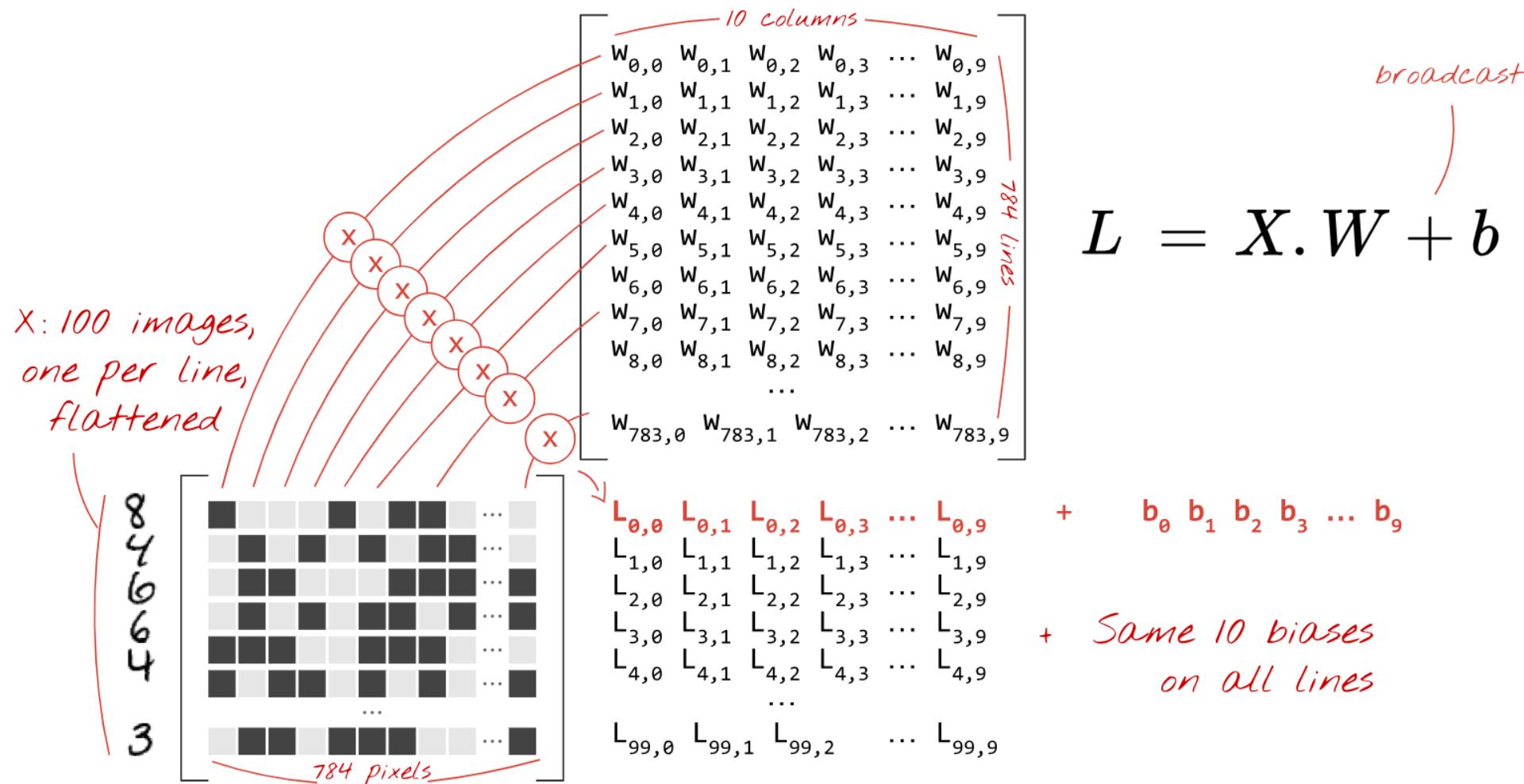
# Prenons un exemple



$$L = X \cdot W + b$$

Modèle simple : classification avec SoftMax [2]

# Prenons un exemple



Modèle simple : classification avec SoftMax [2]

# Prenons un exemple

Predictions

$Y[100, 10]$

$$Y = \text{softmax}(X \cdot W + b)$$

Images

$X[100, 784]$

Weights

$W[784, 10]$

Biases

$b[10]$

applied line  
by line

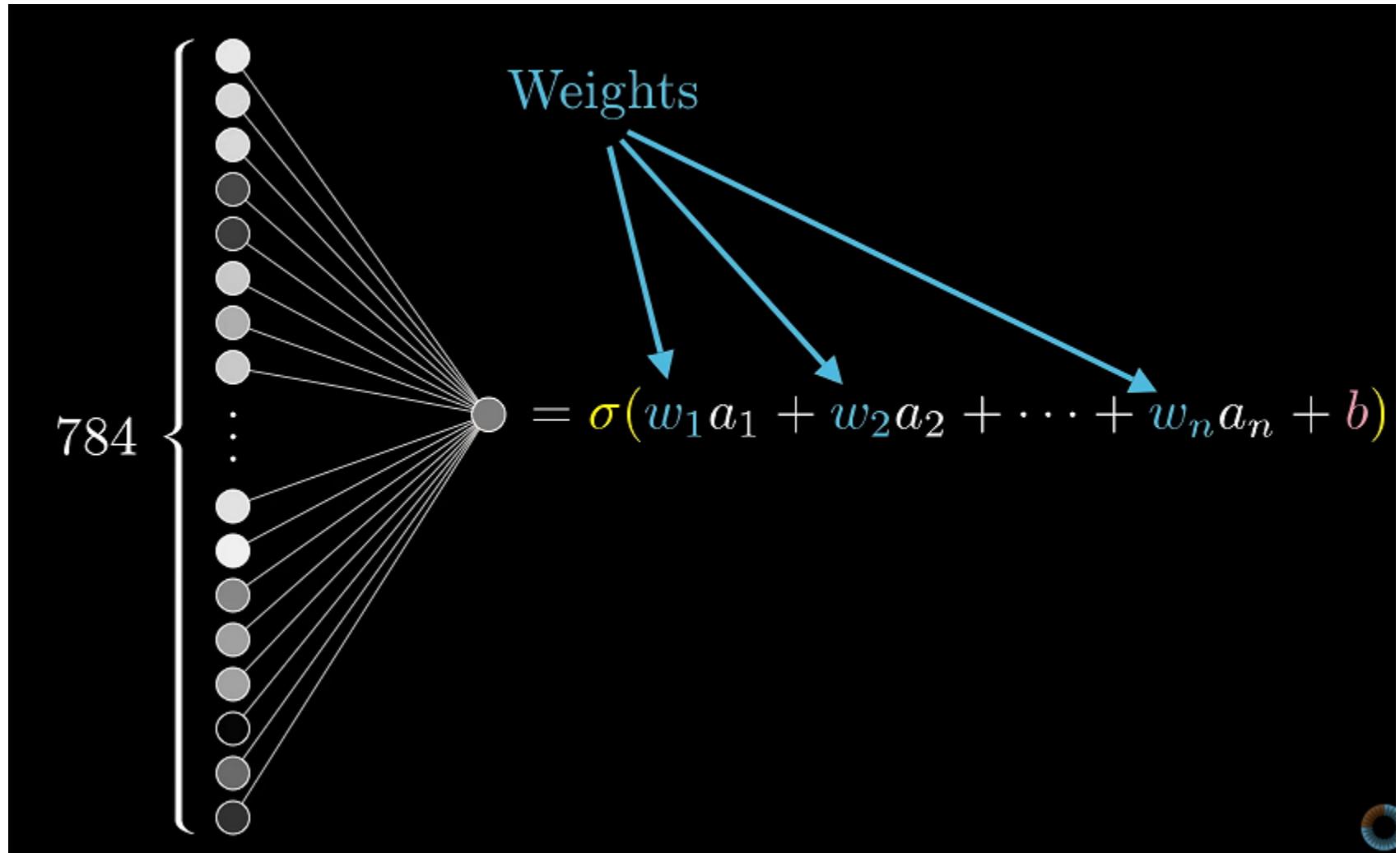
matrix multiply

broadcast  
on all lines

tensor shapes in [ ]

Modèle simple : classification avec SoftMax [2]

# Prenons un exemple



# Exemple de calcul d'erreur

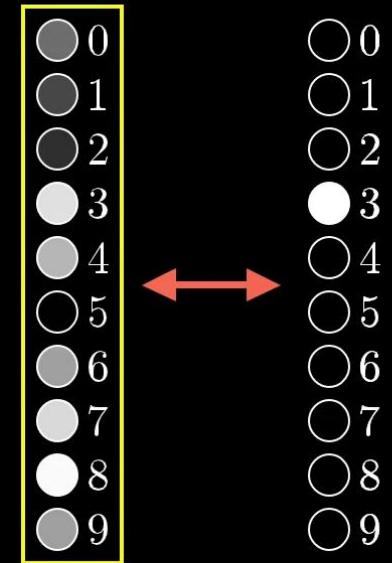
Cost of

3

3.37

$$\left\{ \begin{array}{l} 0.1863 \leftarrow (0.43 - 0.00)^2 + \\ 0.0809 \leftarrow (0.28 - 0.00)^2 + \\ 0.0357 \leftarrow (0.19 - 0.00)^2 + \\ 0.0138 \leftarrow (0.88 - 1.00)^2 + \\ 0.5242 \leftarrow (0.72 - 0.00)^2 + \\ 0.0001 \leftarrow (0.01 - 0.00)^2 + \\ 0.4079 \leftarrow (0.64 - 0.00)^2 + \\ 0.7388 \leftarrow (0.86 - 0.00)^2 + \\ 0.9817 \leftarrow (0.99 - 0.00)^2 + \\ 0.3998 \leftarrow (0.63 - 0.00)^2 \end{array} \right.$$

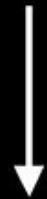
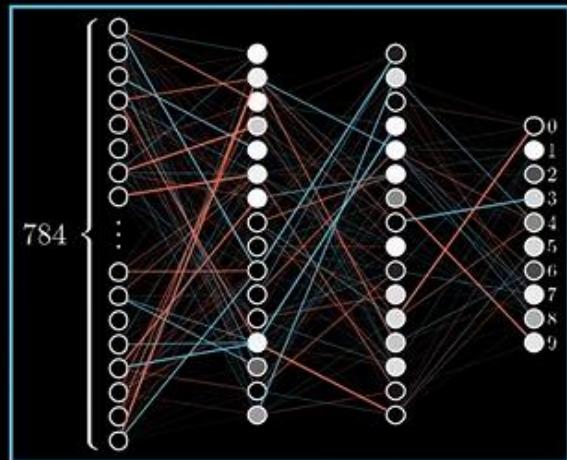
What's the “cost”  
of this difference?



Utter trash

# Exemple de calcul d'erreur

Input



Cost: 5.4

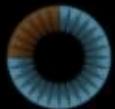
Cost function

Input: 13,002 weights/biases

Output: 1 number (the cost)

Parameters: Many, many, many training examples

$$\left( \boxed{f}, 9 \right)$$



# Descente de gradient et mise à jour des poids

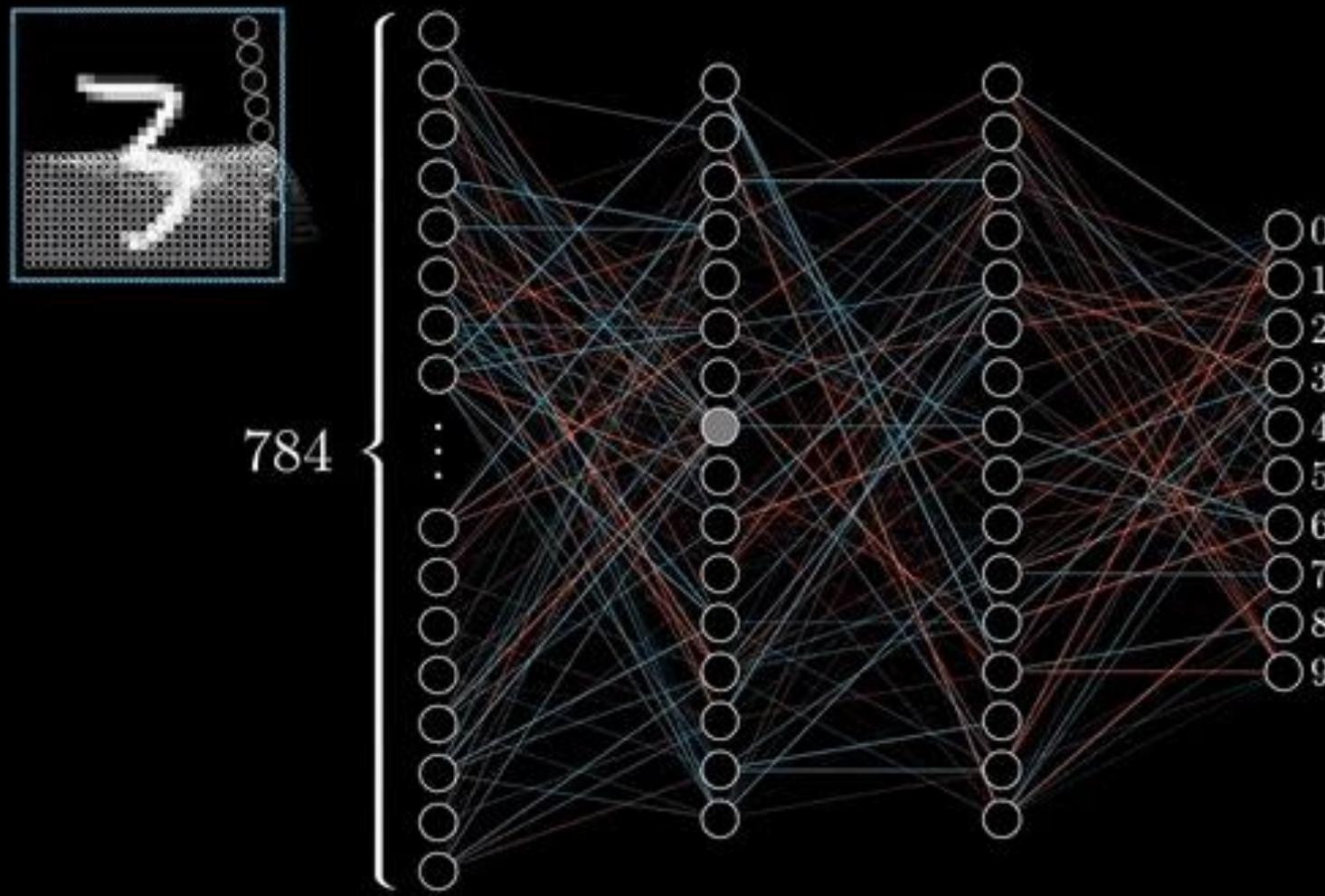
- Problème d'optimisation pouvant être représenté par la formule suivante :

$$\min_{w \in \mathcal{W}} f(w) = \frac{1}{n} \sum_{i=1}^n \ell(h(x_i, w), y_i).$$

- Où n : nombre de données,  
 $X_i$  : données d'entraînement et Y : labels réels
- Les poids d'un réseau de neurones peuvent être mis à jour comme ceci :

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$

# Calcul d'erreur



# Encodage et calcul d'erreur

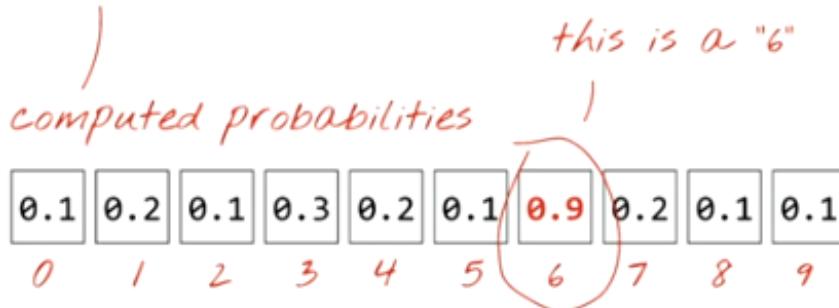
```
labels = to_categorical(labels, num_classes=num_classes)
```

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	0	0	0

actual probabilities, "one-hot" encoded

$$L_2 Loss = \sum_{(x,y) \in D} (y - \text{prédiction}(x))^2$$

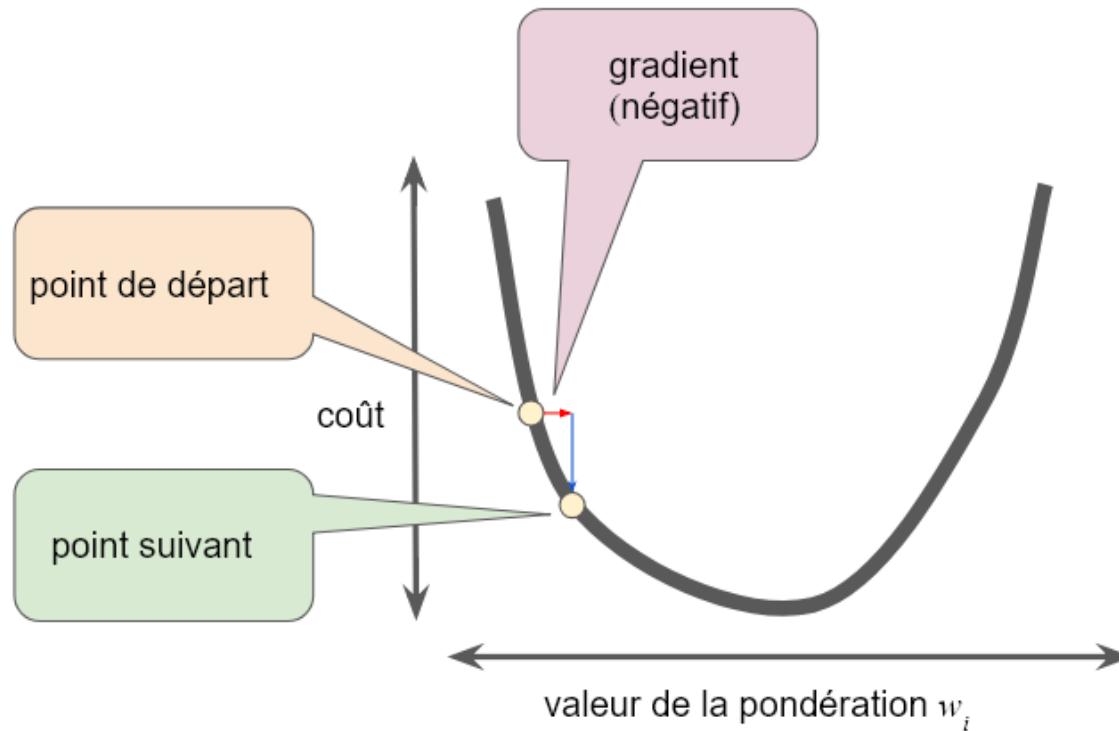
## Erreur



$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - prediction(x))^2$$

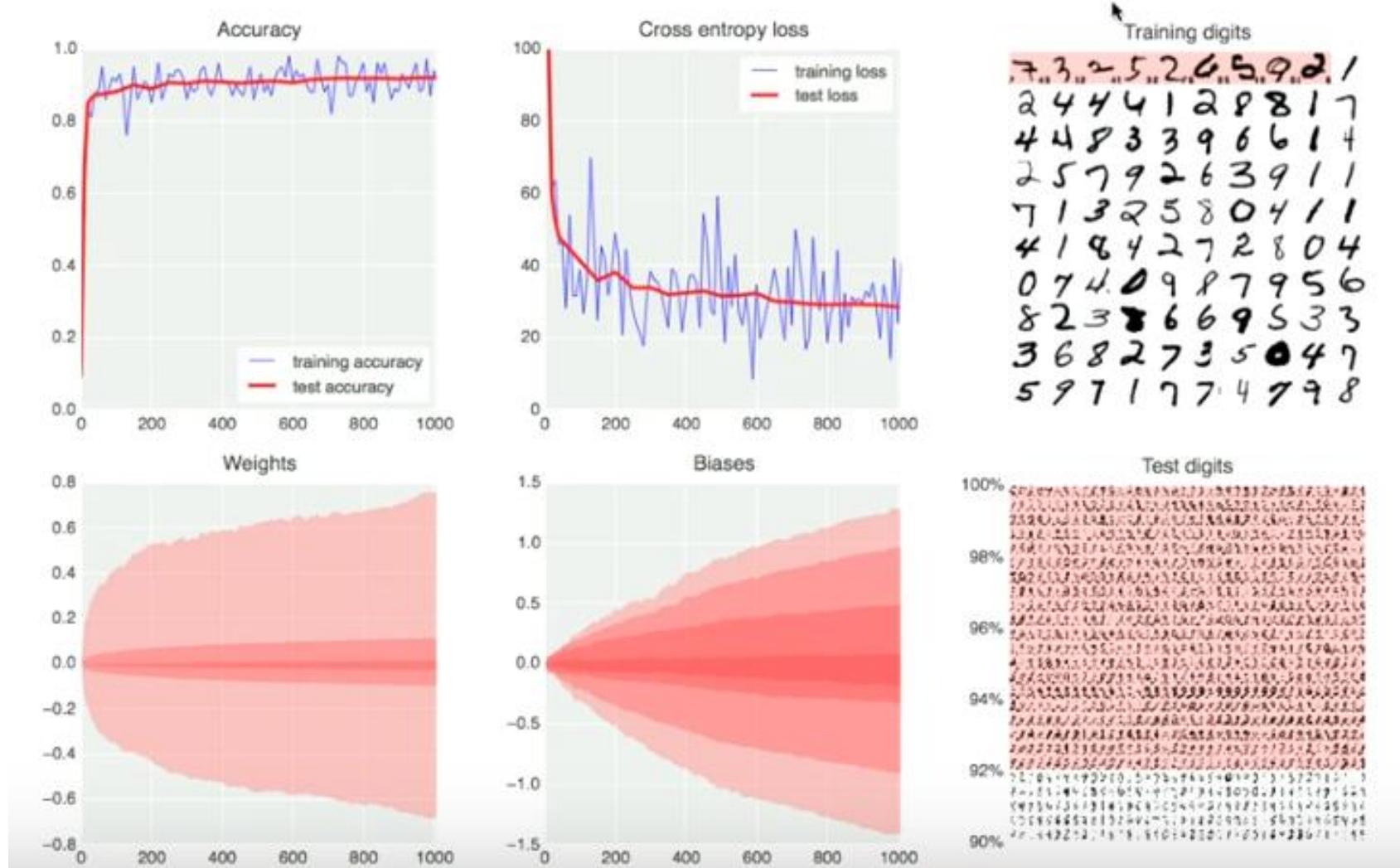
One hot encoding

# La descente du gradient



- Pour déterminer le point suivant, l'algorithme de descente de gradient ajoute une fraction de la magnitude du gradient au point de départ
- Processus répété jusqu'à l'arrivée au minimum

# Premier résultat

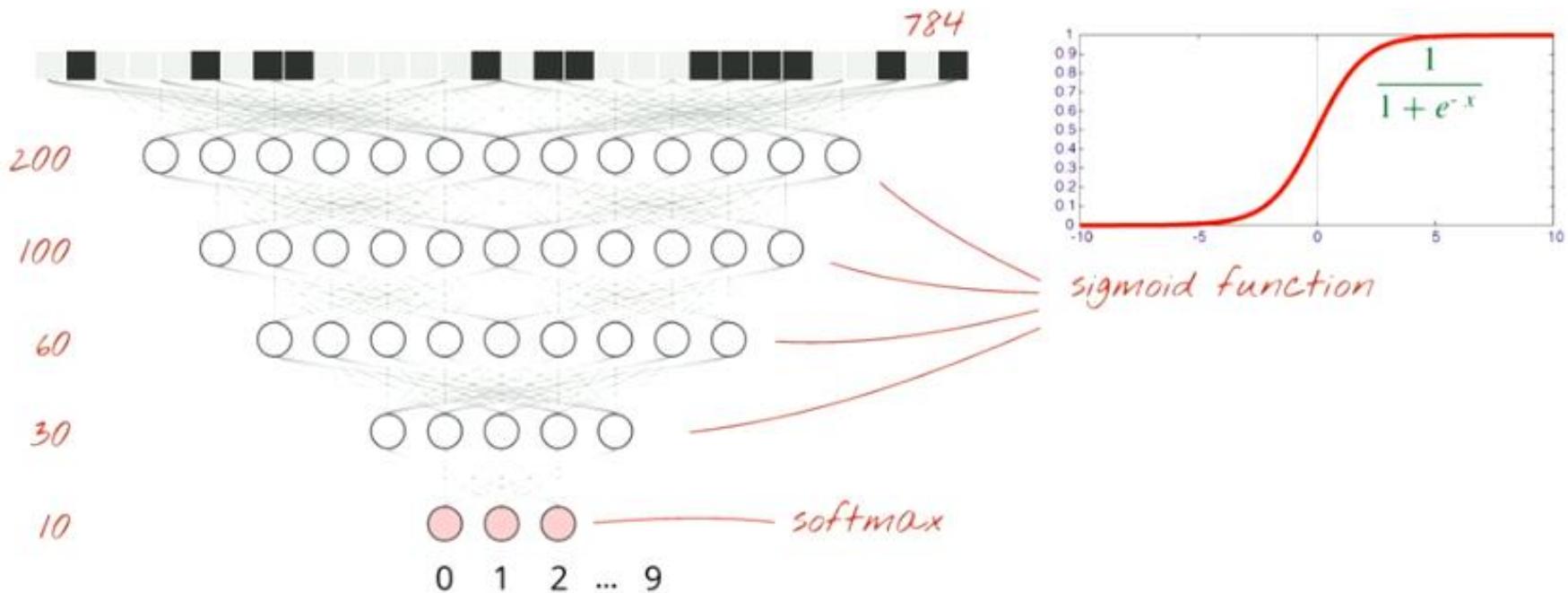


<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/tree/master/tensorflow-mnist-tutorial>

# Analyse

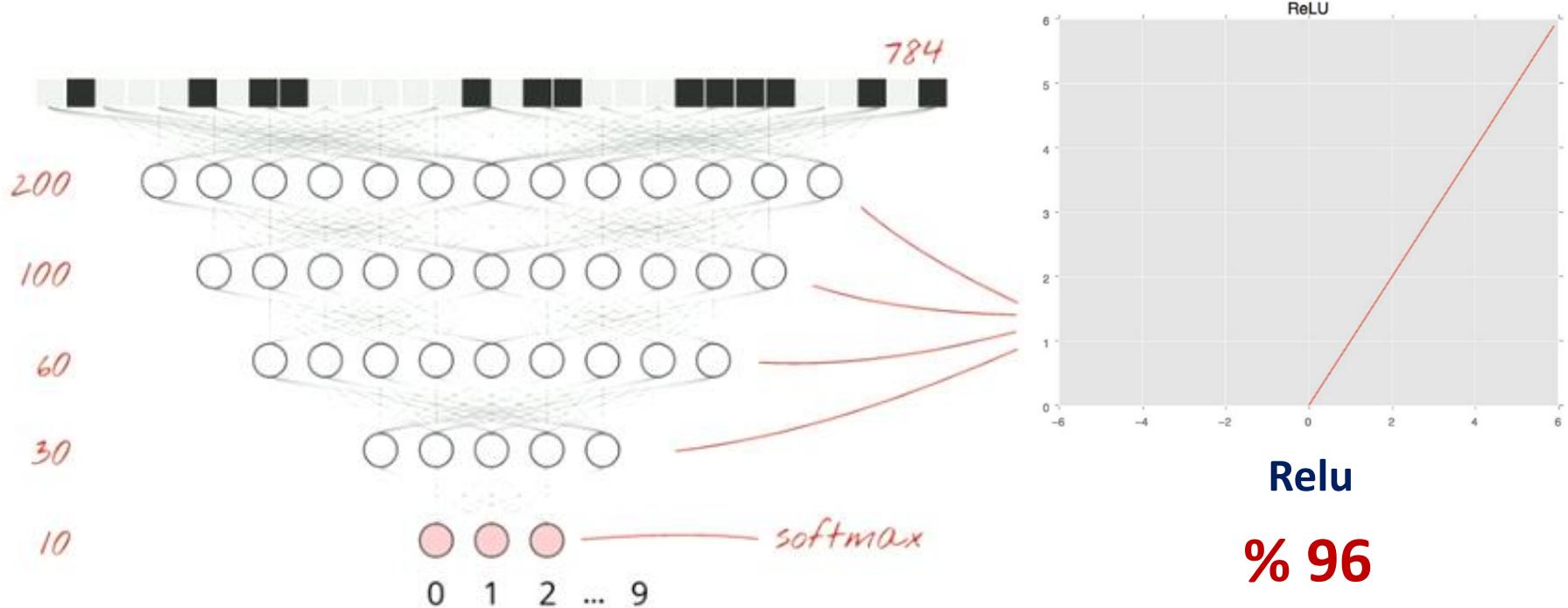
- Peut-on améliorer le score ?
- Si oui, **comment ?**

# Amélioration N° 01



- Meilleur score mais avec un démarrage d'entraînement plus lent
- Pourquoi ?

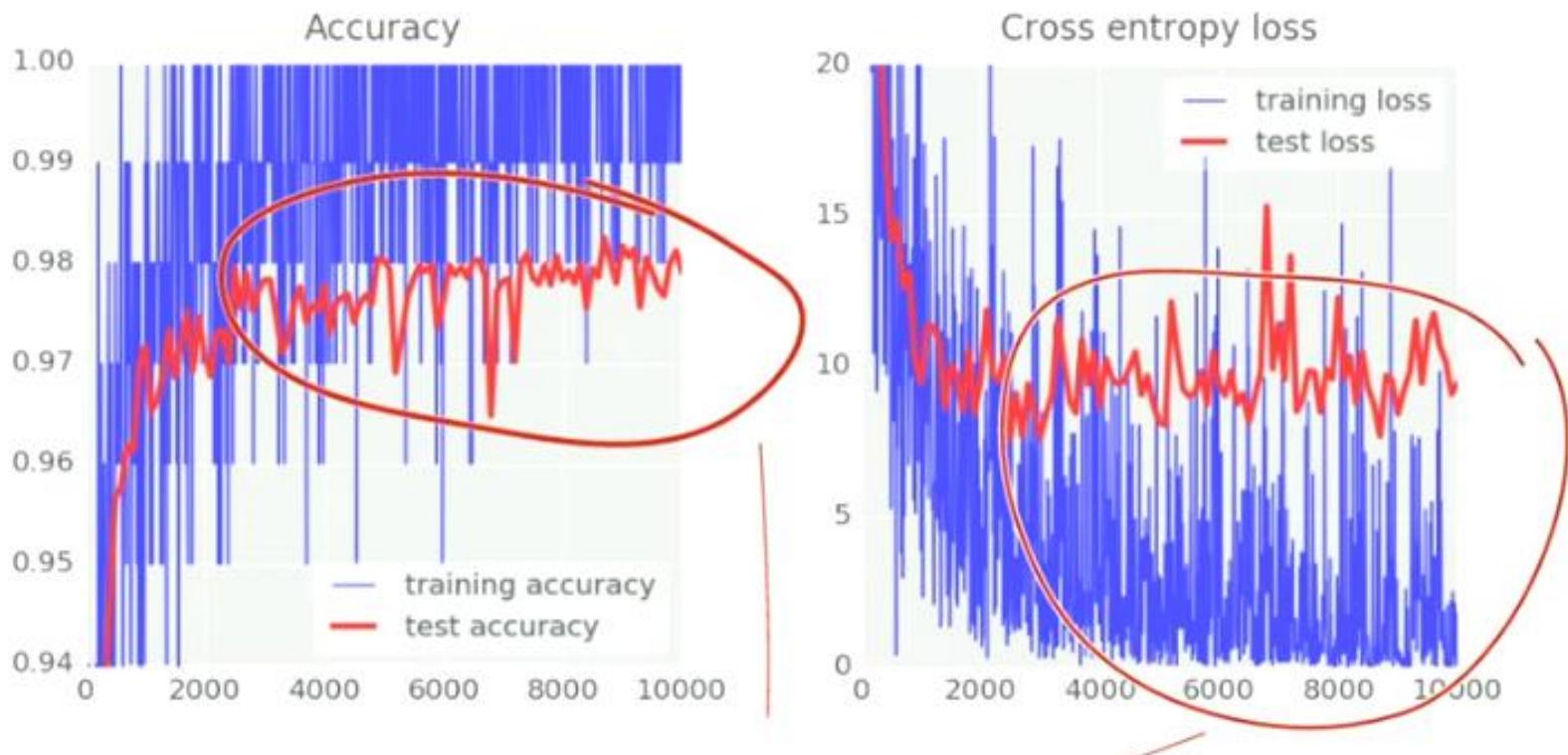
# Amélioration N° 02



% 96

- Remplacement de la fonction Sigmoid par la fonction Relu
- Meilleur score : 96 % au lieu de 92 %

# 98 % mais ?



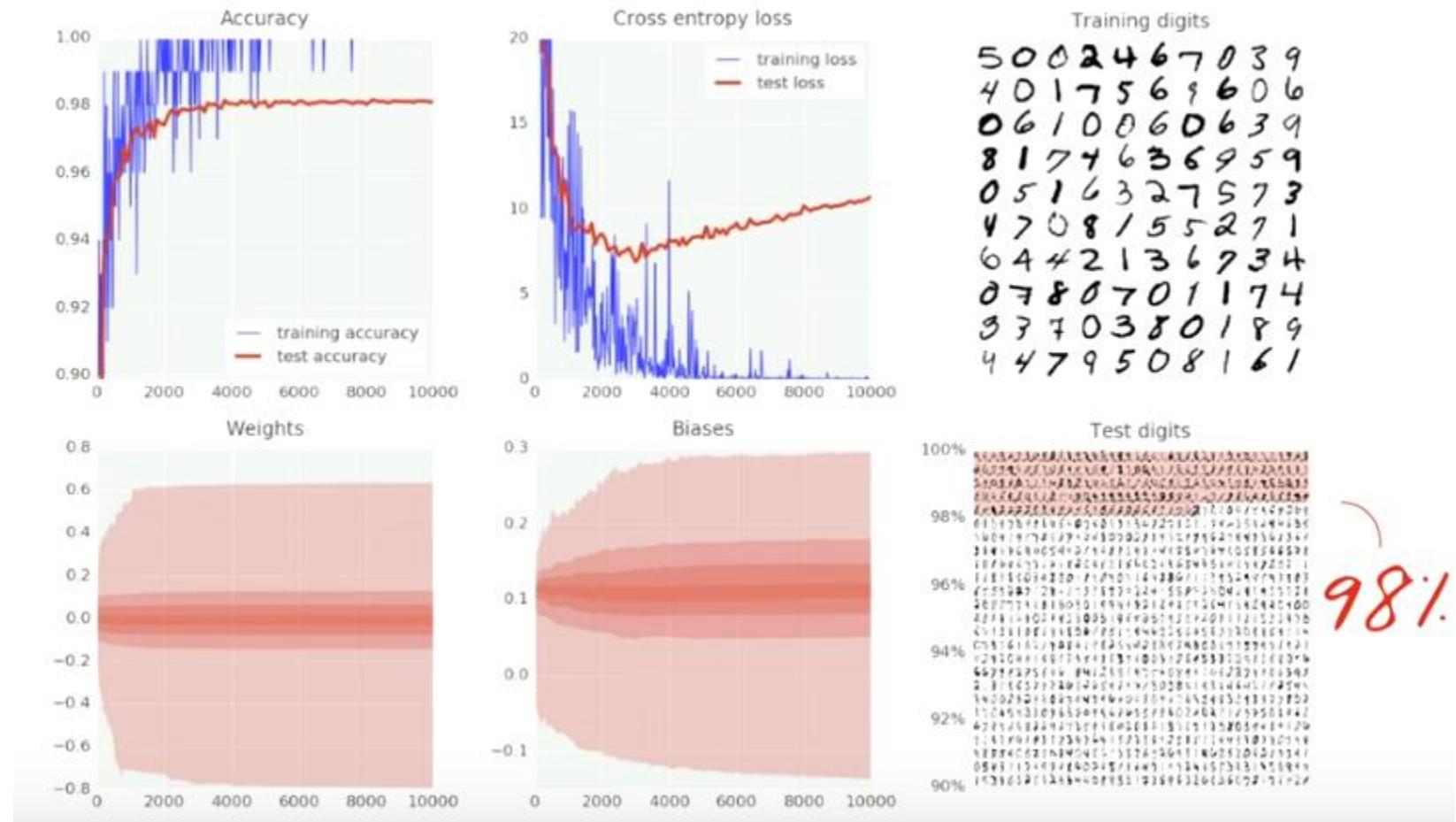
Courbe de précision “Accuracy” bruitée



Réduction progressive du taux d'apprentissage (0,003 à 0,0001)

<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/tree/master/tensorflow-mnist-tutorial>

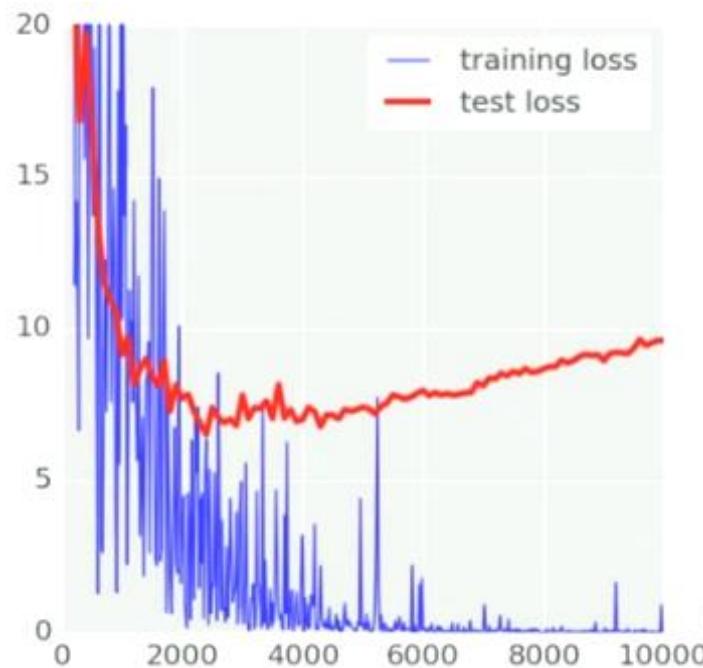
# Amélioration N° 03 (rate decay)



<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/tree/master/tensorflow-mnist-tutorial>

# Analysons de nouveau

- Voyez-vous un problème dans les courbes ?
- Si oui, lequel ?



Overtfitting  
(sur apprentissage)

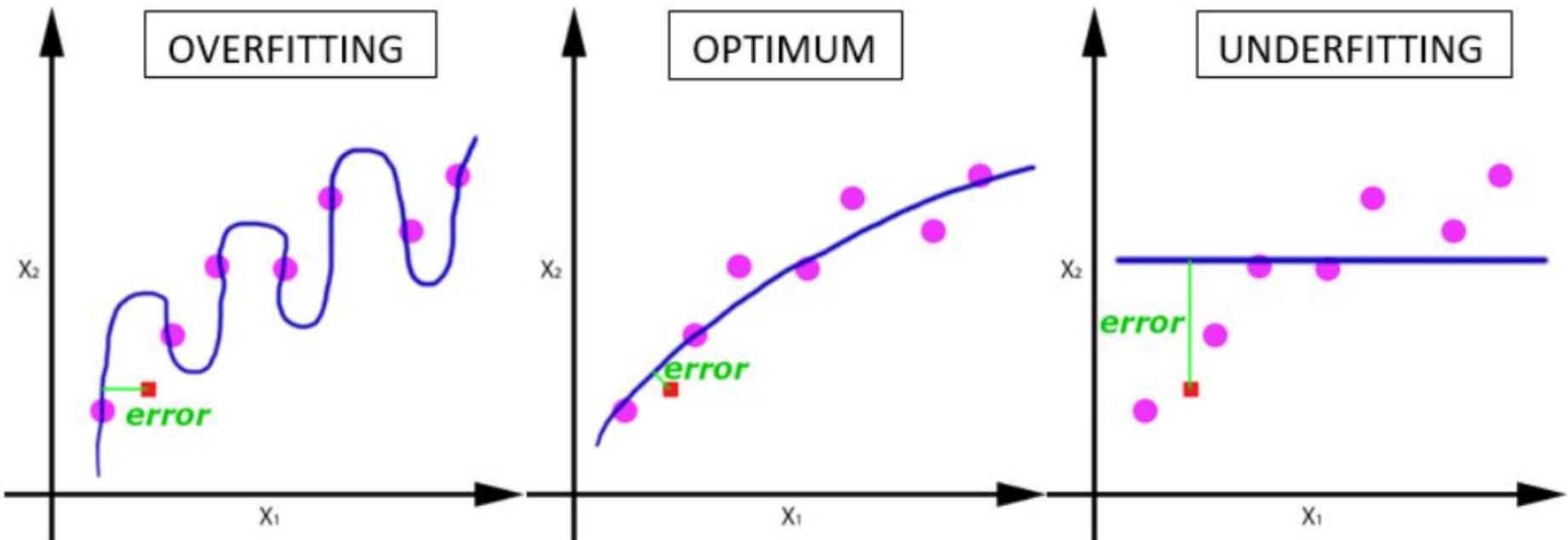
<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/tree/master/tensorflow-mnist-tutorial>

# Overfitting



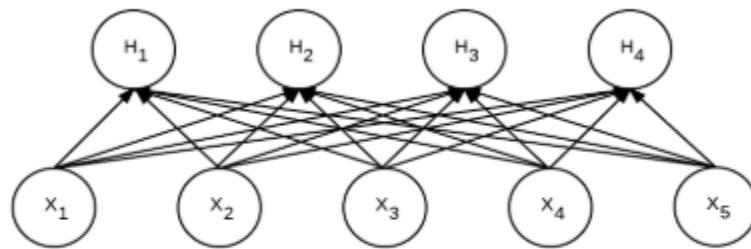
<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/tree/master/tensorflow-mnist-tutorial>

# Overfitting

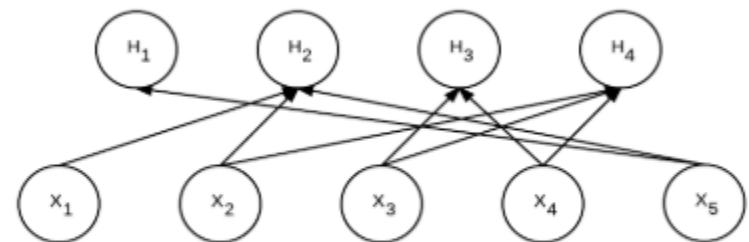


Le modèle apprend les données d'entraînement “par cœur”, et ne peut donc généraliser sur d'autres données.

# Dropout



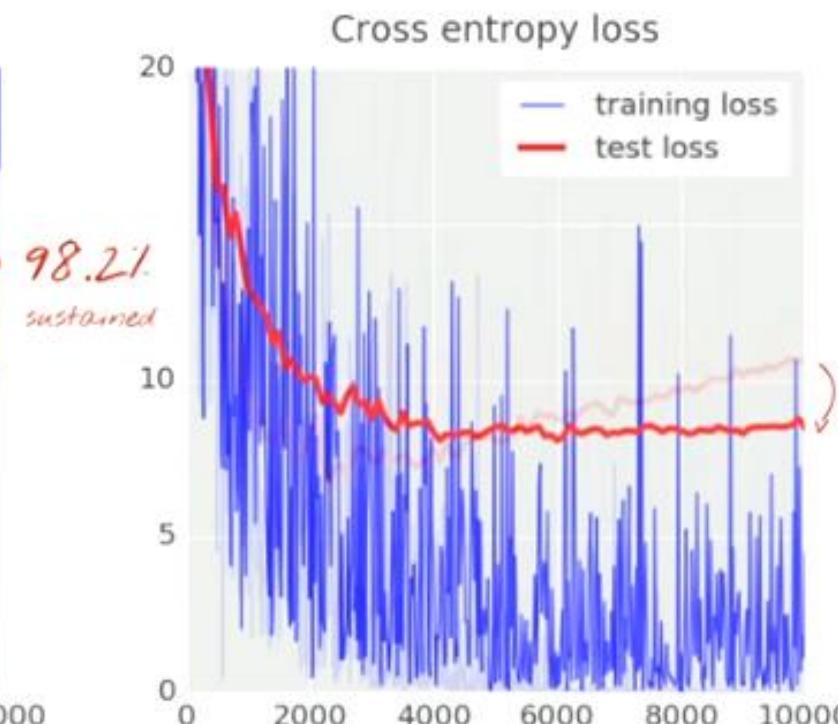
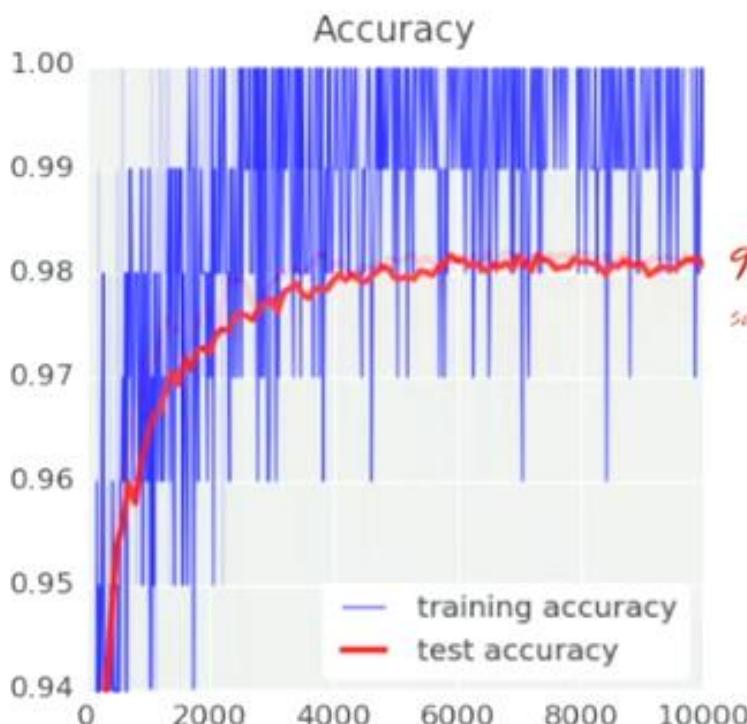
FC



FC with 50% dropout

Utilisé pour éviter l'Overfitting en **supprimant aléatoirement** des connexions entre deux couches pendant l'entraînement

# Amélioration N° 04



%98,2

05 couches + Relu + réduction progressive du taux d'apprentissage + Dropout (75%)

<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/tree/master/tensorflow-mnist-tutorial>

# PLAN

## Introduction

I. Le Perceptron

II. Types de réseaux de neurones profonds (ANN, MLP, CNN, RNN, etc.)

III. Paramètres d'entraînement, syntaxe et évaluation de modèles

IV. Réseaux de neurones convolutionnels (CNNs)

V. Outils de développement et matériel de calcul

## Conclusion

# Paramètres d'entraînement

## Epochs

- 1 époque : l'ensemble entier des données est passé dans le réseau 1 fois

## Batch\_size

- Nombre de données d'entraînement présents dans un batch
- Données divisées en plusieurs batchs

## Itérations

- Nombre de batchs par époques : tailles des données/batch\_size

# Paramètres d'entraînement « Torch Lightning »

## trainer.fit

```
modell = MNISTModel(mlp1)

# Initialize a trainer
trainer = pl.Trainer(
    accelerator=ACCELERATOR,
    max_epochs=EPOCHS,
    logger=csv_logger,
    callbacks=[checkpoint_callback]
)
# Train the model ⚡
trainer.fit(modell, train_loader, val_loader)
```

# Préparation et division des données

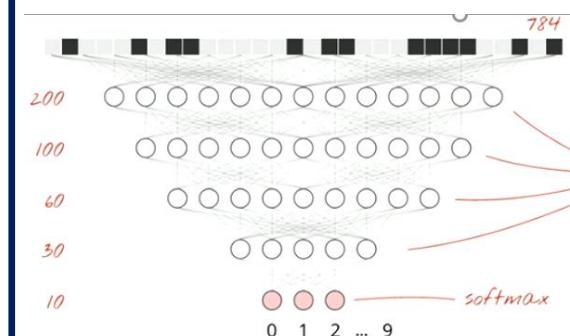


# Préparation, division des données et entraînement

## Training Set

```
class MLP2(nn.Module):
    def __init__(self):
        super(MLP2, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 200)
        self.fc2 = nn.Linear(200, 100)
        self.fc3 = nn.Linear(100, 60)
        self.fc4 = nn.Linear(60, 30)
        self.fc5 = nn.Linear(30, 10)

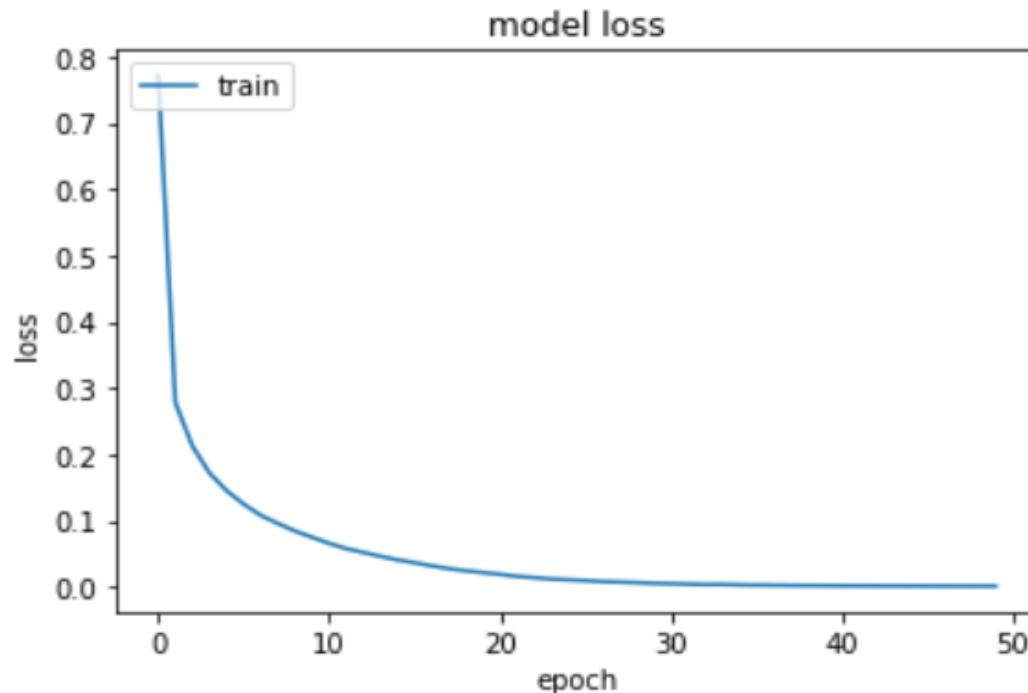
    def forward(self, x):
        # Flatten the input (28x28 -> 784)
        x = x.view(x.size(0), -1)
        # Hidden layers with sigmoid activations
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        x = torch.sigmoid(self.fc4(x))
        x = self.fc5(x)
        return x
```



# Préparation, division des données et entraînement

## Training Set

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

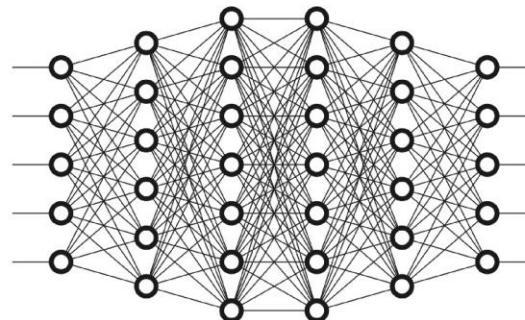


- Peut-on évaluer ce modèle ?

# Préparation, division des données et entraînement

Training Set

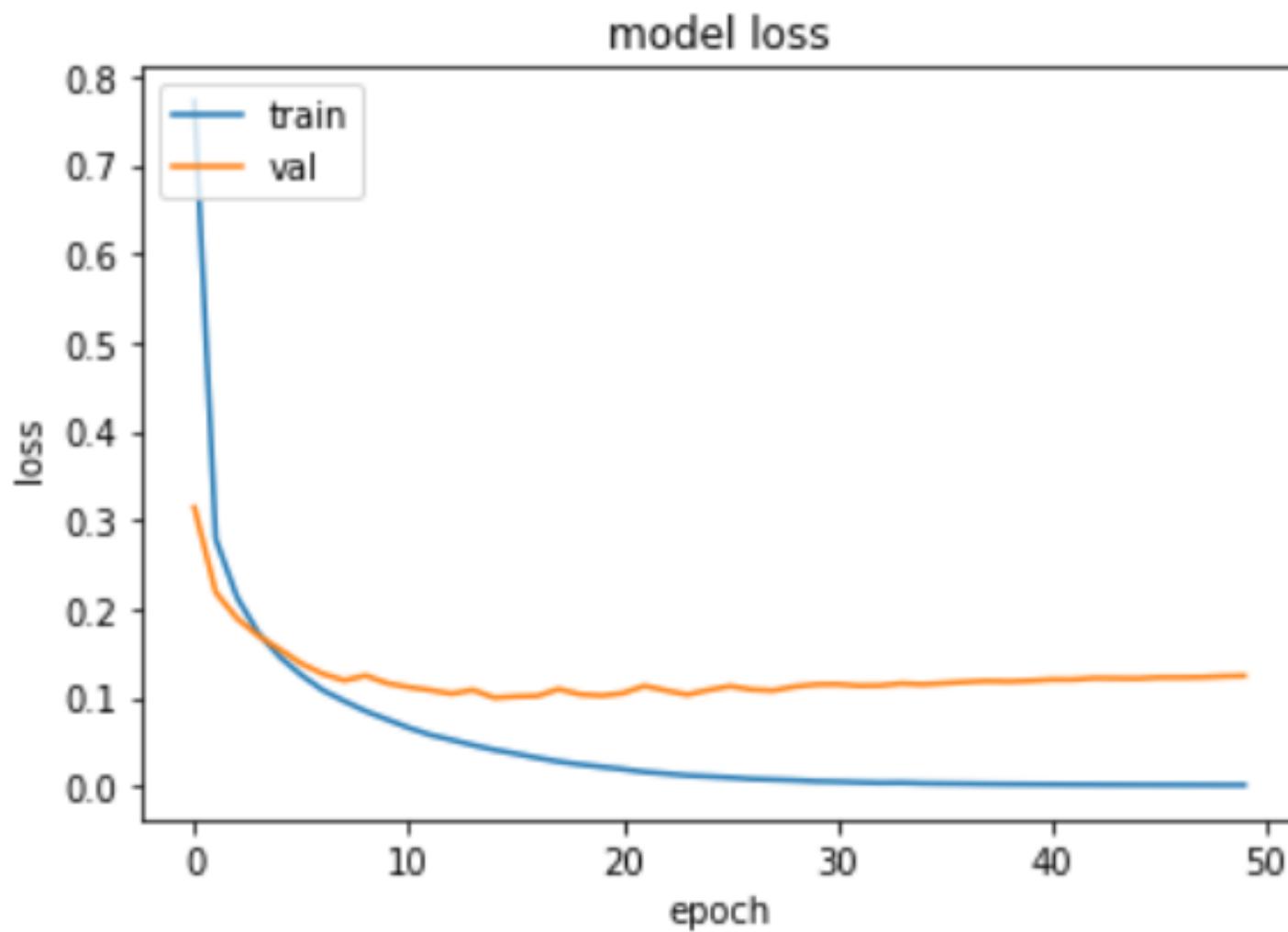
Validation Set



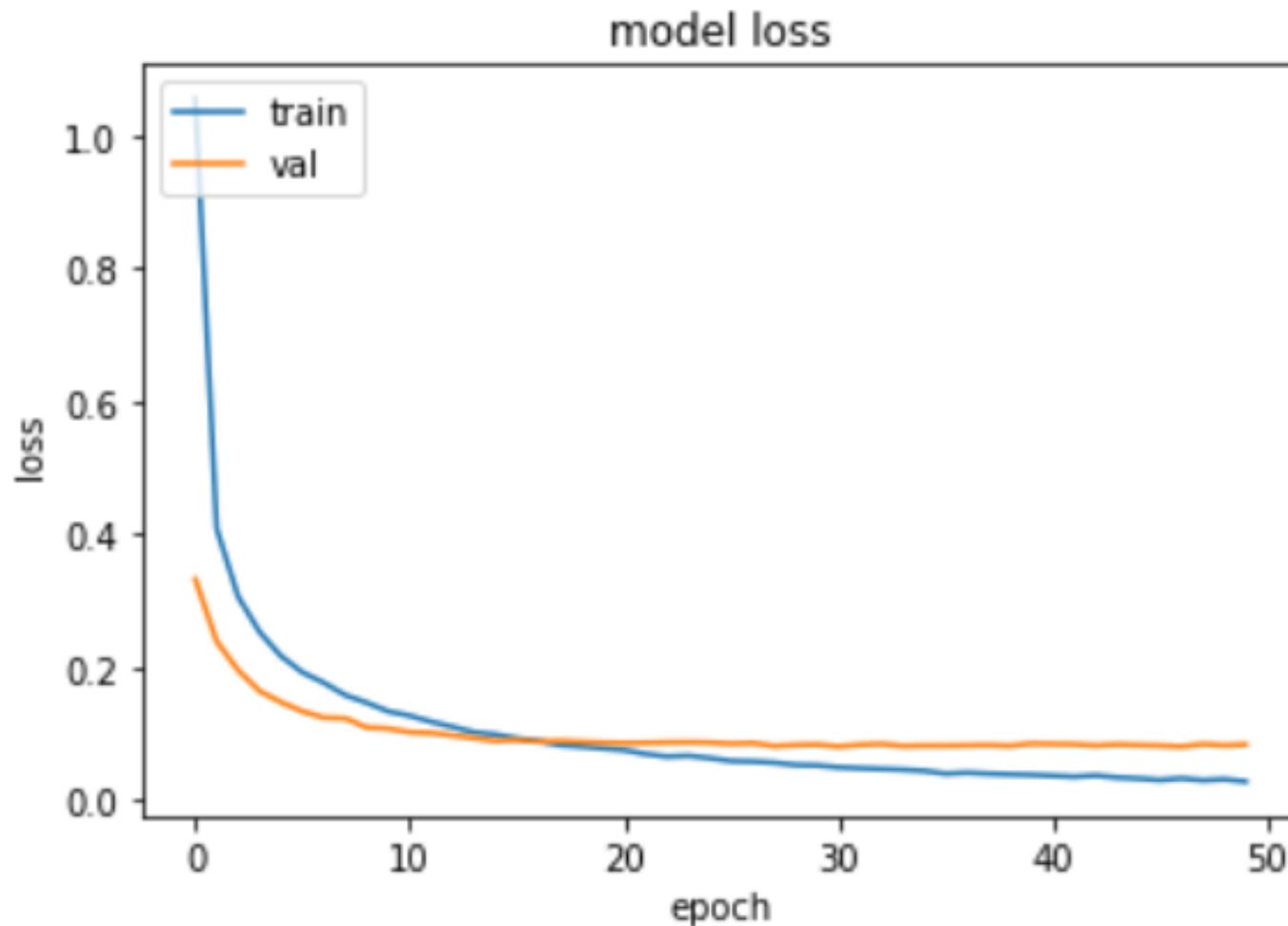
```
trainer.fit(model, train_loader,val_loader)
```

- Où se trouve la différence ? Est-ce suffisant ?

# Préparation, division des données et entraînement



# Avec le Dropout



# Préparation, division des données et entraînement

Training Set

Validation Set

Test Set

- Données de test: soit téléchargés au début
- Données de test : split des données d'entraînement

```
trainX, testX, trainY, testY=train_test_split(x_train, y_train, test_size=0.25, random_state=42)
```

# Préparation, division des données et entraînement

Training Set

Validation Set

Test Set

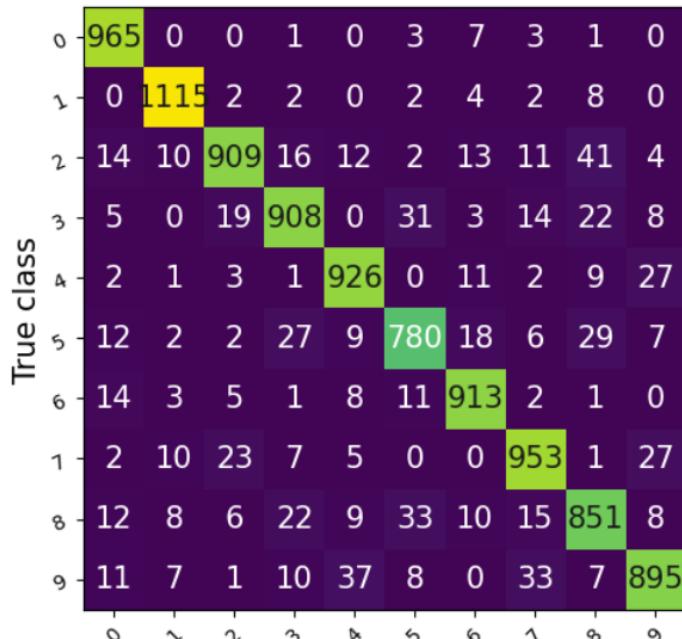
Entrainement

Après entraînement

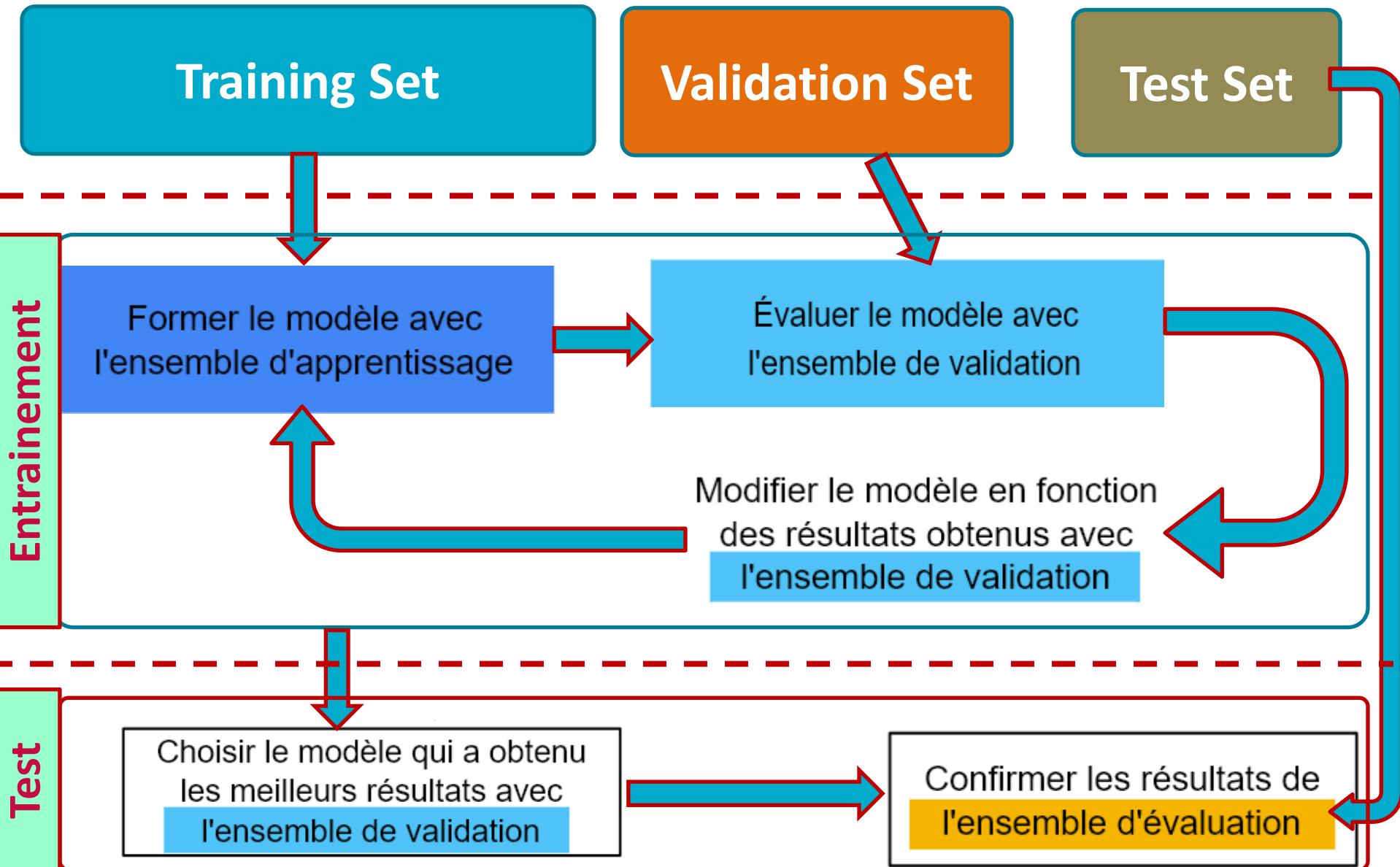
```
trainer.test(dataloaders=train_loader, ckpt_path="./mlp1.ckpt")  
  
trainer.test(dataloaders=val_loader, ckpt_path="./mlp1.ckpt")  
  
trainer.test(dataloaders=test_loader, ckpt_path="./mlp1.ckpt")
```

Test metric	DataLoader 0
test_acc	0.92150027179718
test_loss	0.2809731364250183

[{'test\_loss': 0.2809731364250183, 'test\_acc': 0.92150027179718}



# Processus d'entraînement



# PLAN

Introduction

I. Le Perceptron

II. Types de réseaux de neurones profonds (ANN, MLP, CNN, RNN, etc.)

III. Analyse des données et évaluation de modèles

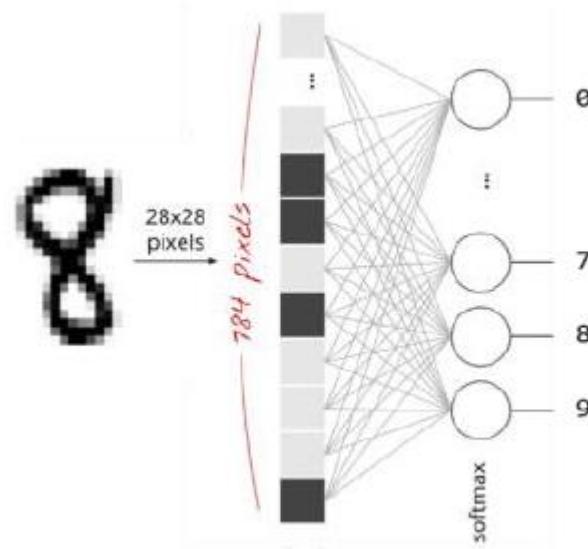
IV. Réseaux de neurones convolutionnels (CNNs)

V. Outils de développement et matériel de calcul

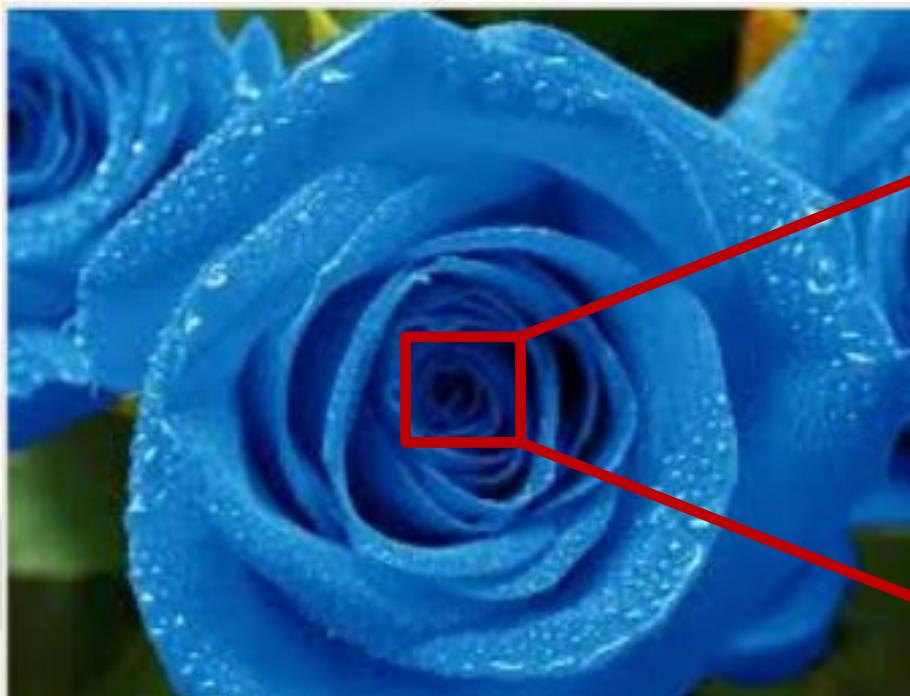
Conclusion

# Encore

- Peut-on aller plus loin ?
- Pourquoi ?



# Petit rappel : une image



Une image



Une région de l'image

# Extraction de caractéristiques et filtrage d'image convolutions

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

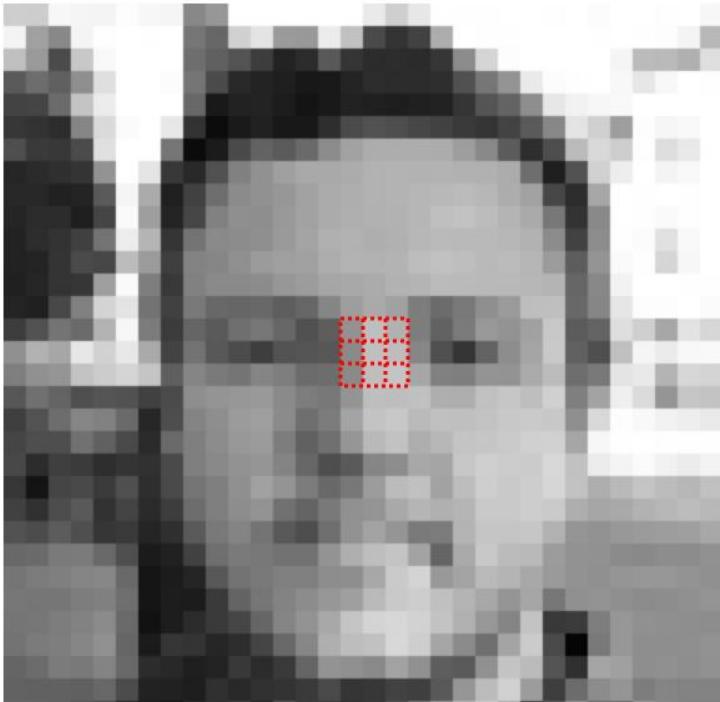
Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

## Exemple de convolution

# Extraction de caractéristiques et filtrage d'image convolutions

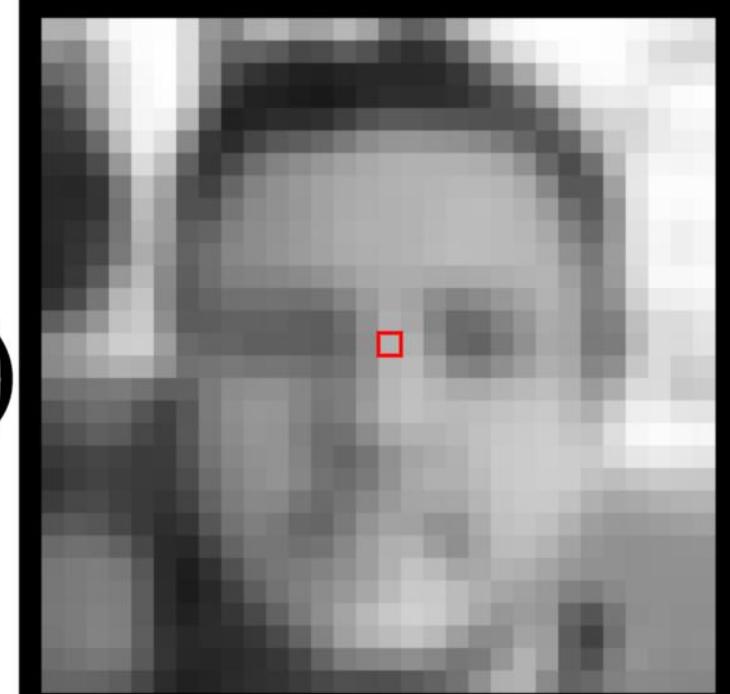


input image

$$\begin{aligned} & \left( \begin{array}{ccc} 147 & + & 191 & + & 178 \\ \times 0.0625 & & \times 0.125 & & \times 0.0625 \\ \\ & + & 139 & + & 192 & + & 190 \\ & \times 0.125 & & \times 0.25 & & \times 0.125 \\ \\ & + & 139 & + & 191 & + & 197 \\ & \times 0.0625 & & \times 0.125 & & \times 0.0625 \end{array} \right) \\ \\ & = 178 \end{aligned}$$

kernel:

blur

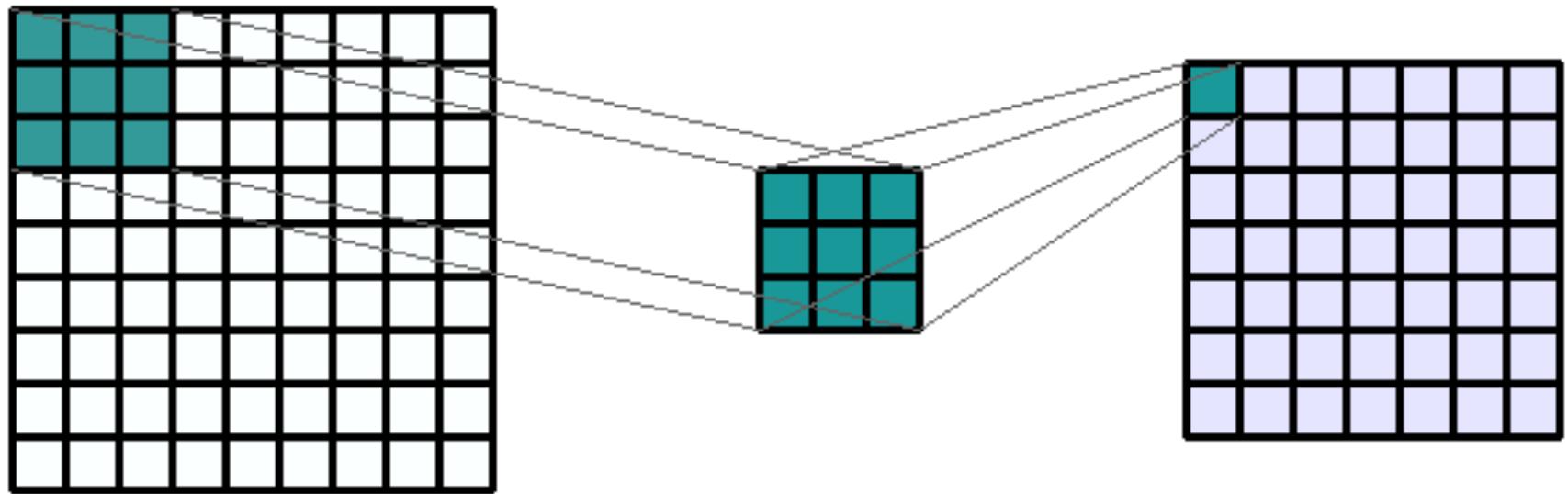


output image

<https://setosa.io/ev/image-kernels/>

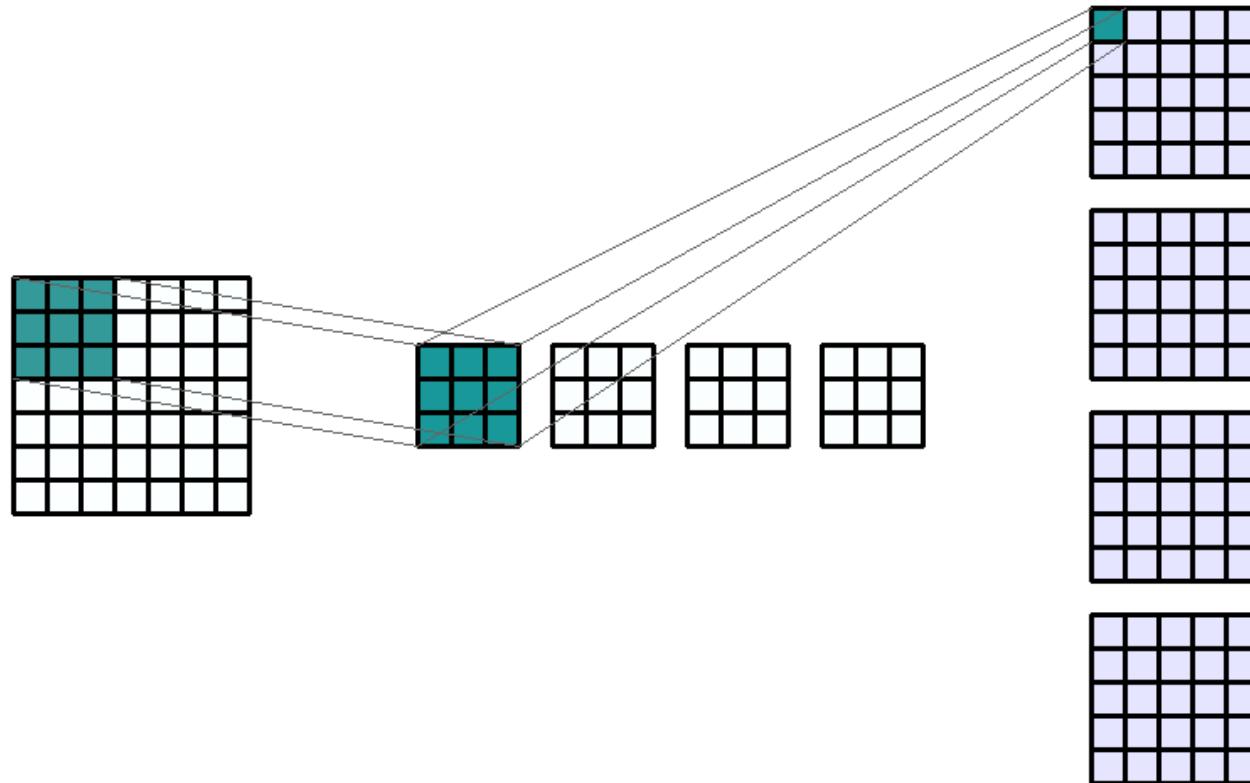
Exemple de convolution

# Extraction de caractéristiques et filtrage d'image convolutions



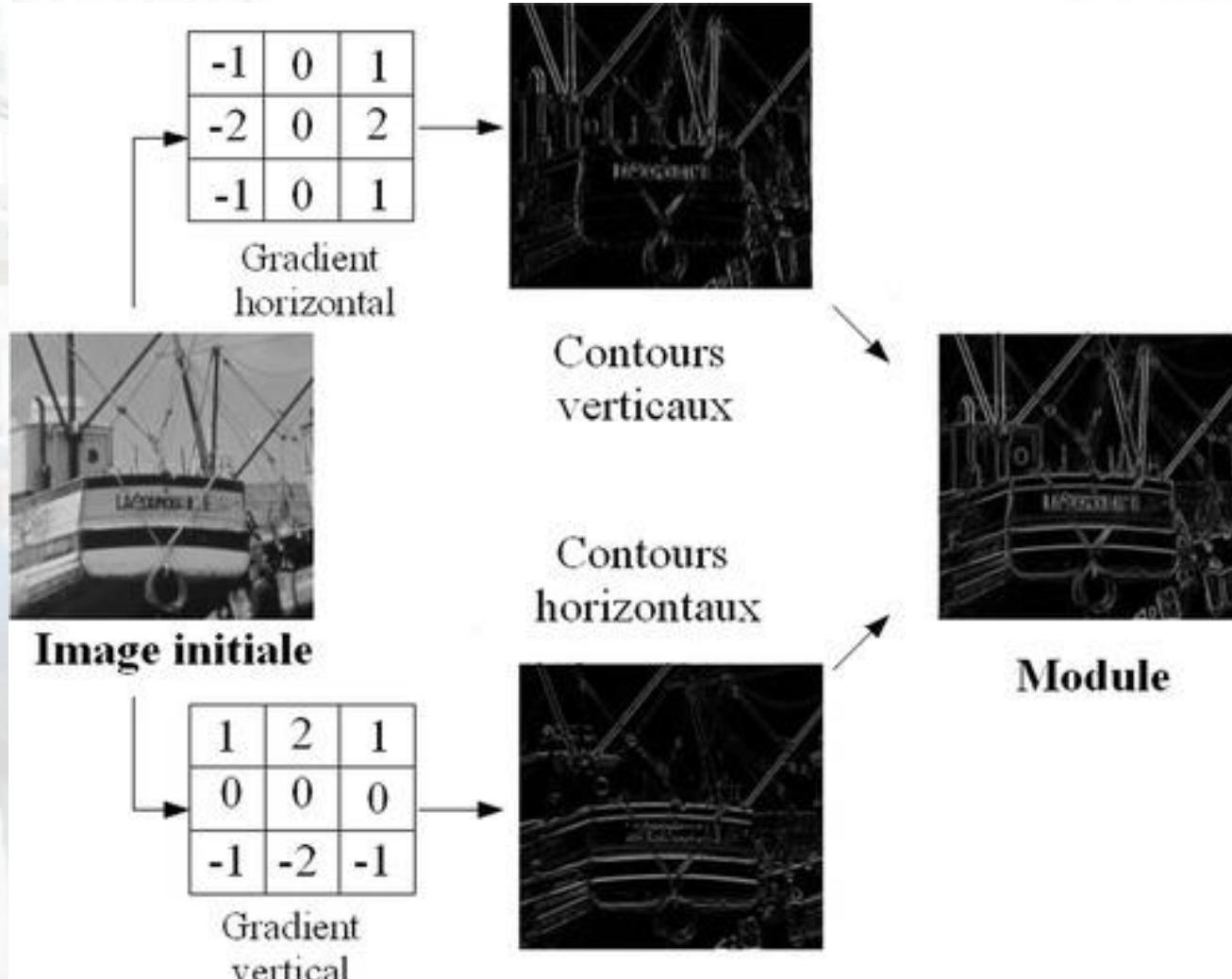
Un filtre de convolution

# Extraction de caractéristiques et filtrage d'image convolutions



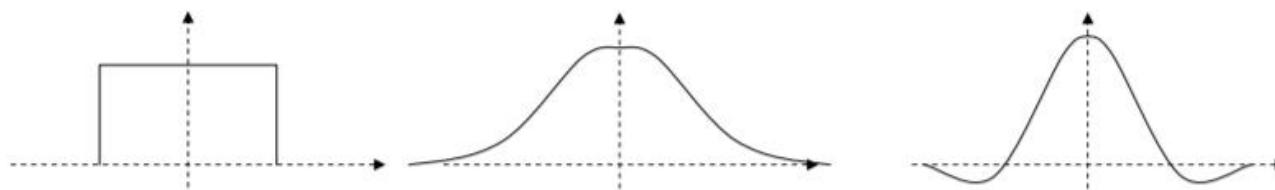
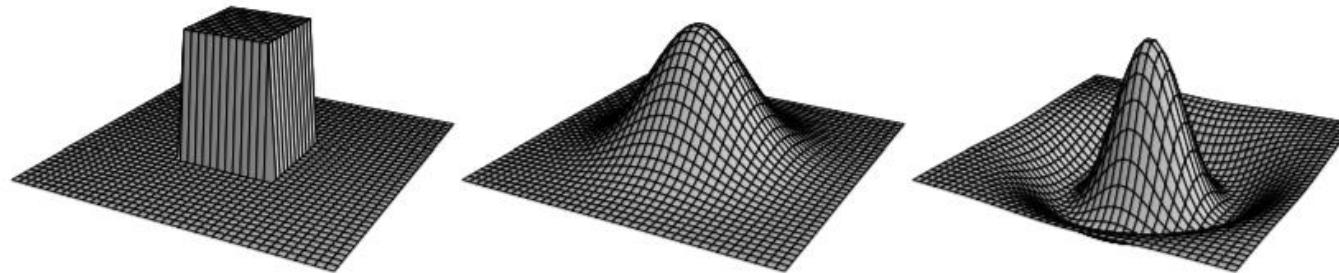
4 filtres de convolution

# Extraction de caractéristiques et filtrage d'image convolutions



## Exemples de filtres

# Extraction de caractéristiques et filtrage d'image convolutions



0	0	0	0	0	0
0	1	1	1	0	
0	1	1	1	0	
0	1	1	1	0	
0	0	0	0	0	

(a)

0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

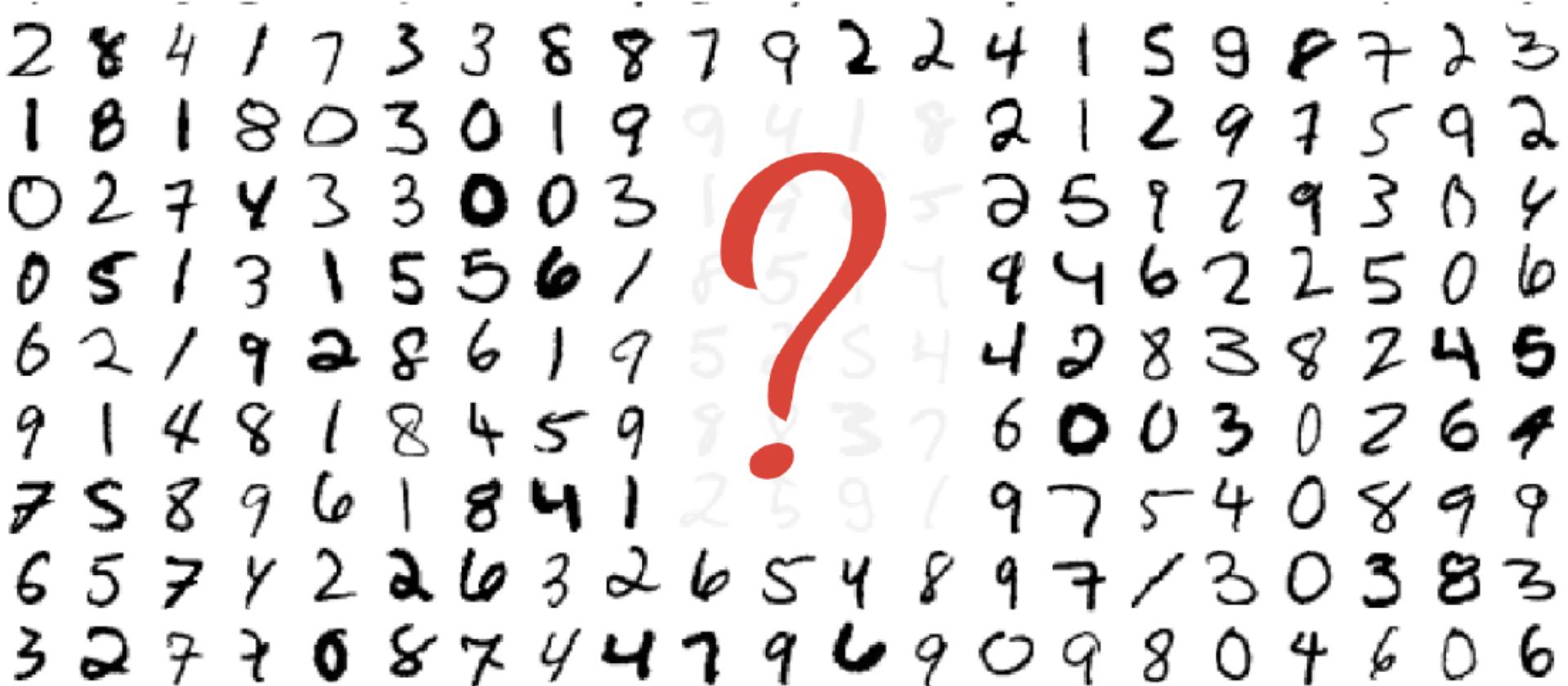
(b)

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

(c)

## Exemples de convolution

# Premier classifieur d'images



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

# Réseaux de neurones convolutionnels (CNNs)

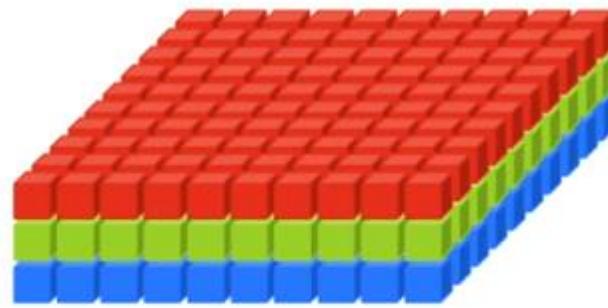
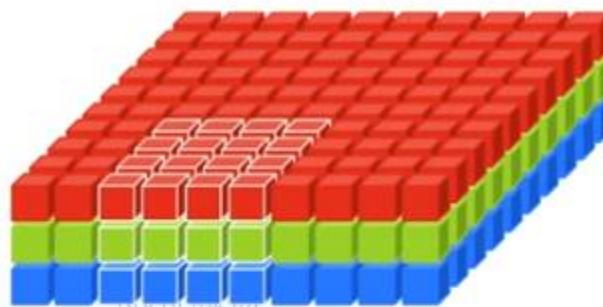


Image 2D

# Réseaux de neurones convolutionnels (CNNs)



$$\begin{array}{|c|c|c|} \hline -1 & 0 & +1 \\ \hline -2 & 0 & +2 \\ \hline -1 & 0 & +1 \\ \hline \end{array}$$

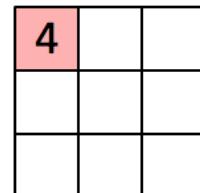
Gx

$$\begin{array}{|c|c|c|} \hline +1 & +2 & +1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

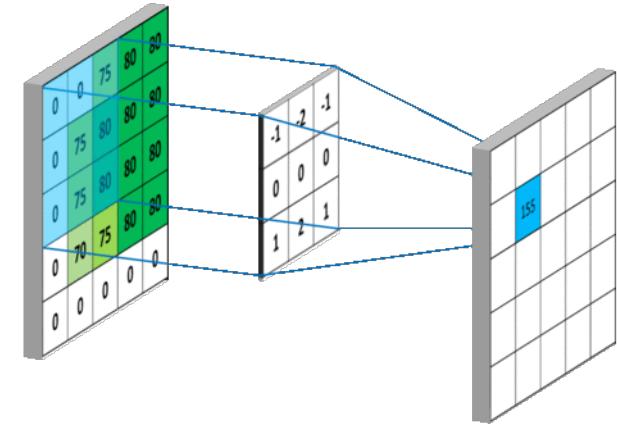
Gy

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

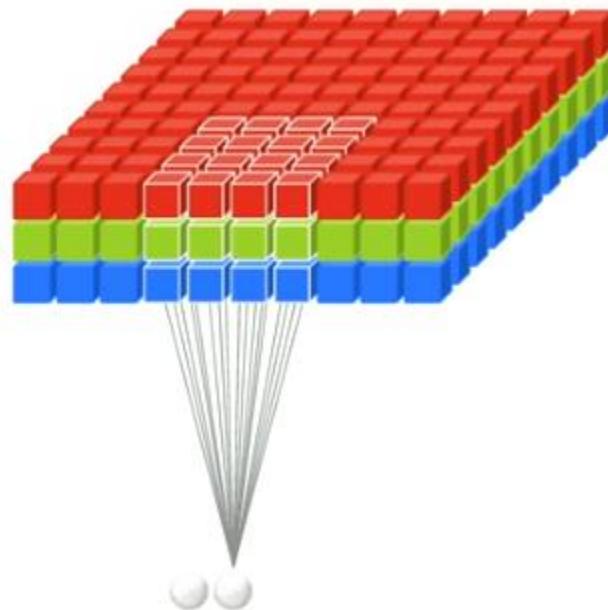
Image



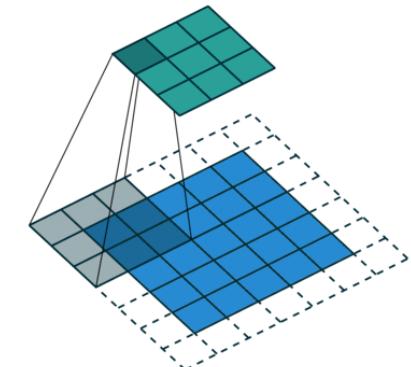
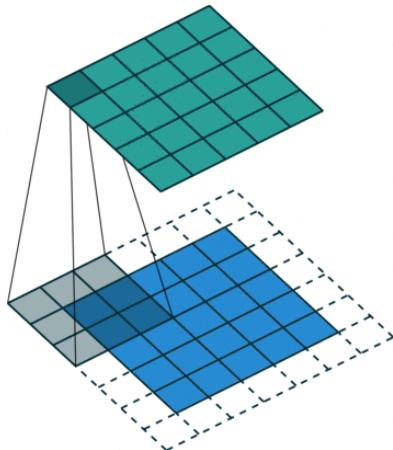
Convolved  
Feature



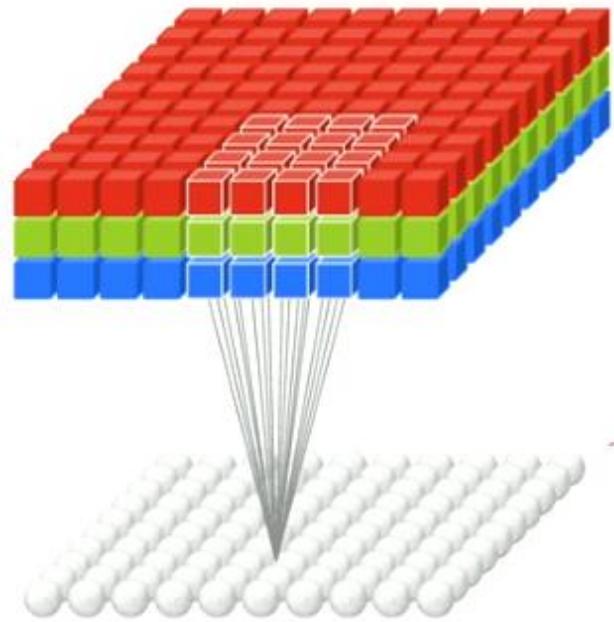
# Réseaux de neurones convolutionnels (CNNs)



**Stride = 2**



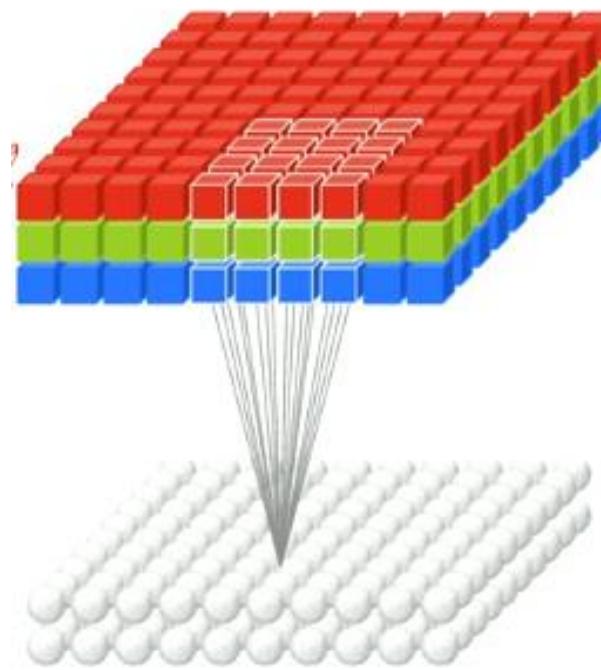
# Réseaux de neurones convolutionnels (CNNs)



Padding =1

Matrices de poids  
 $W [4, 4, 3]$

# Réseaux de neurones convolutionnels (CNNs)



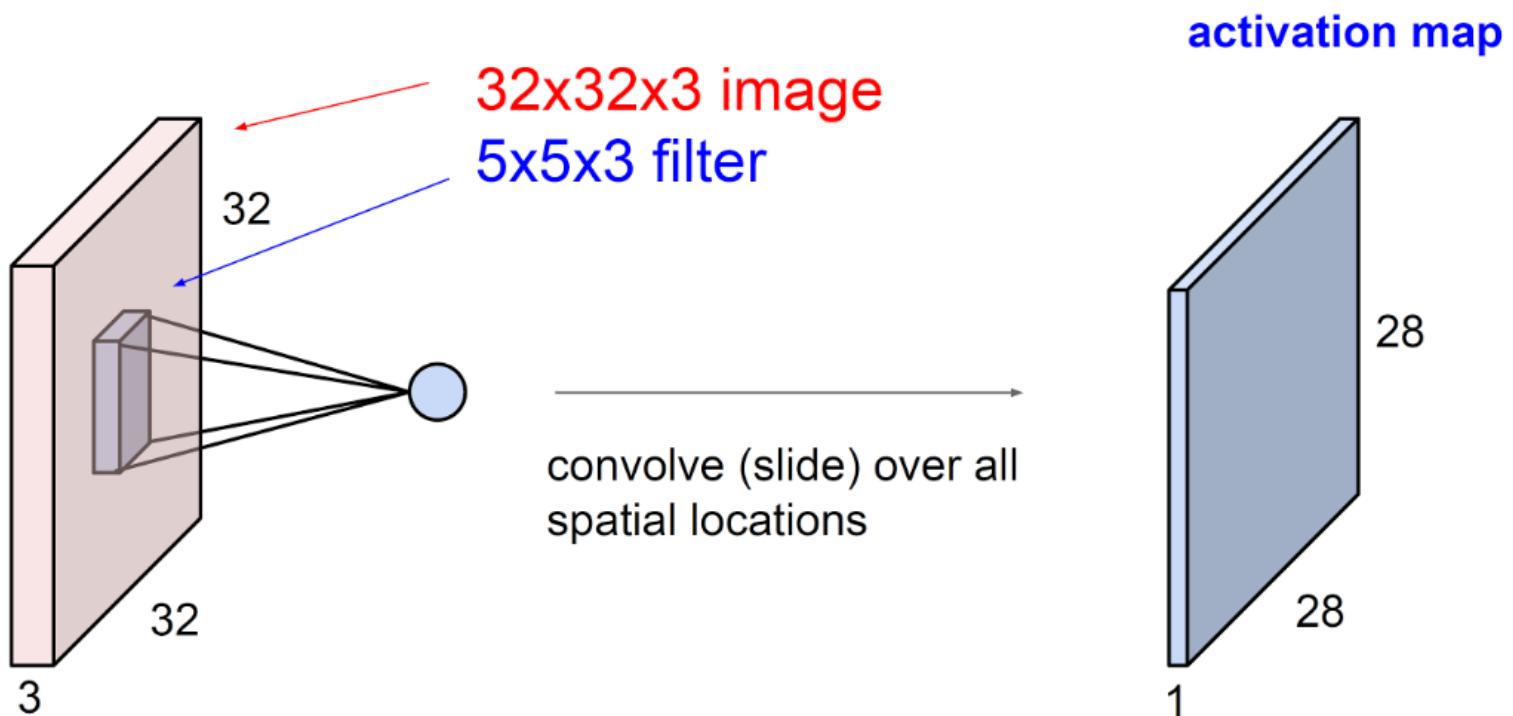
Matrices de poids

$W [4, 4, 3]$

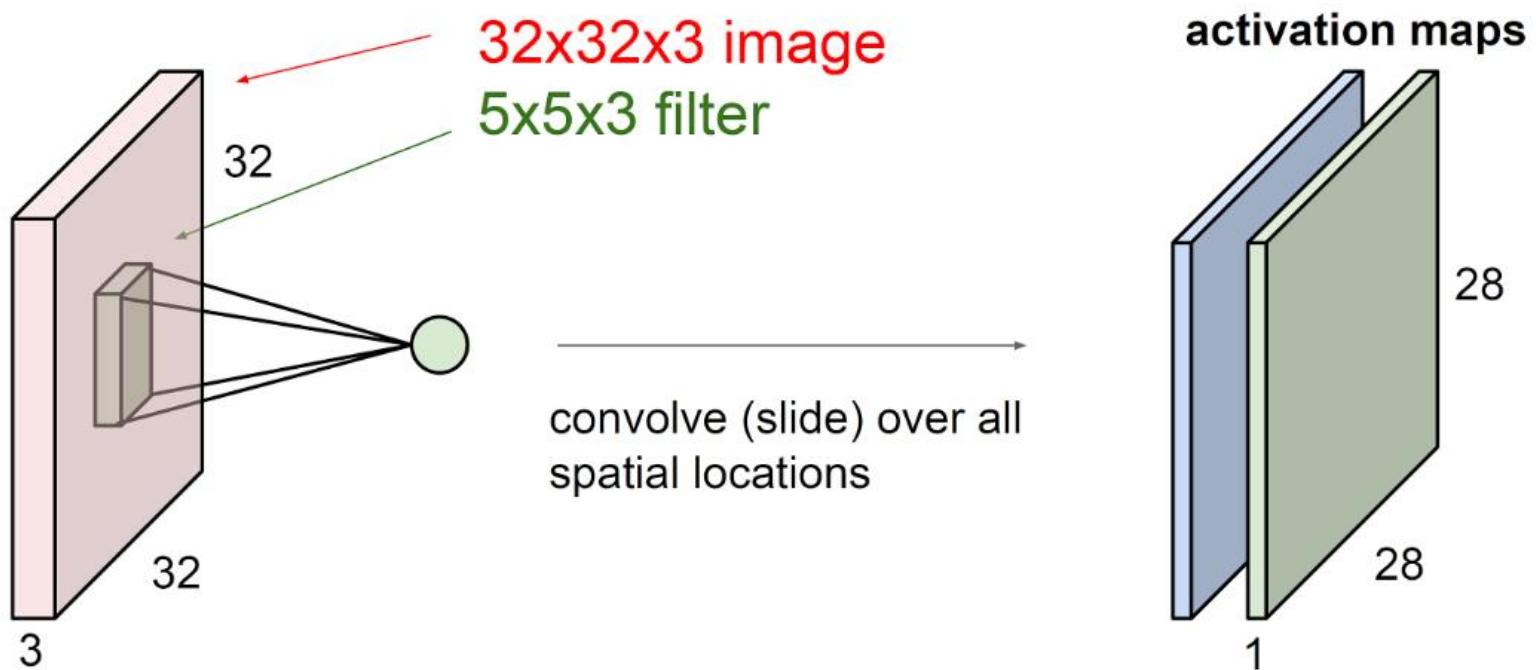
$W [4, 4, 3]$

$W [4, 4, 3, 2]$

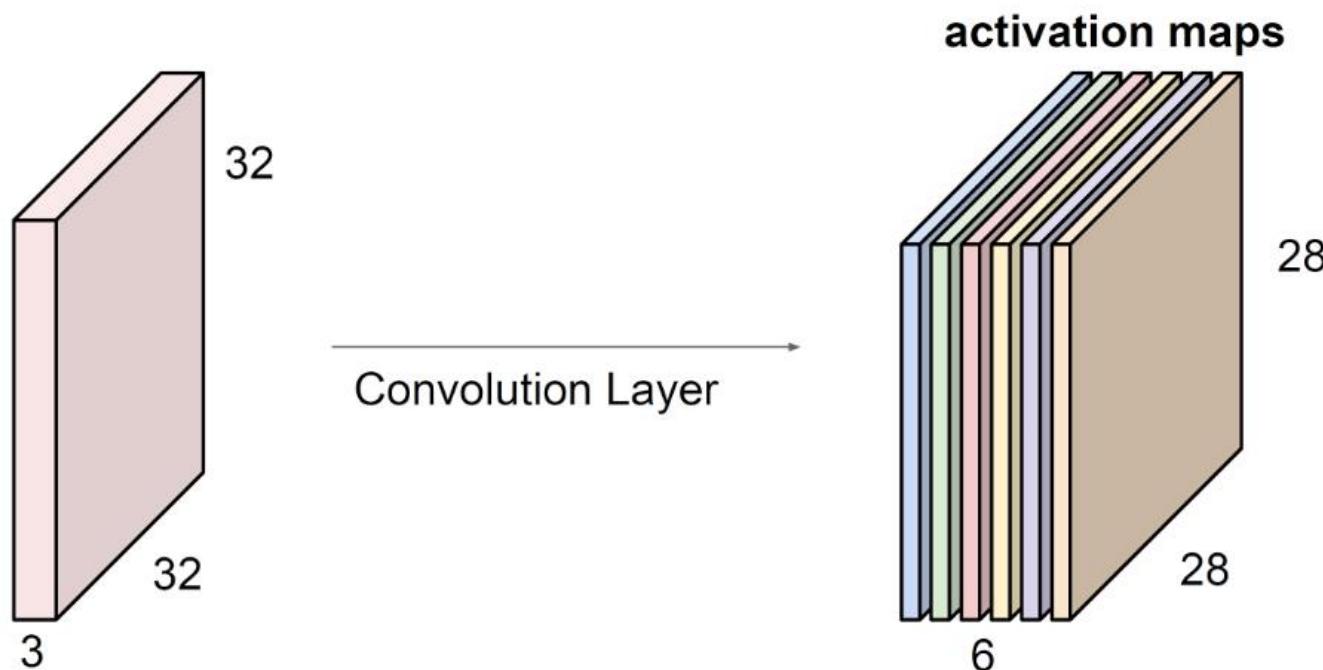
# CNN : couches de convolution



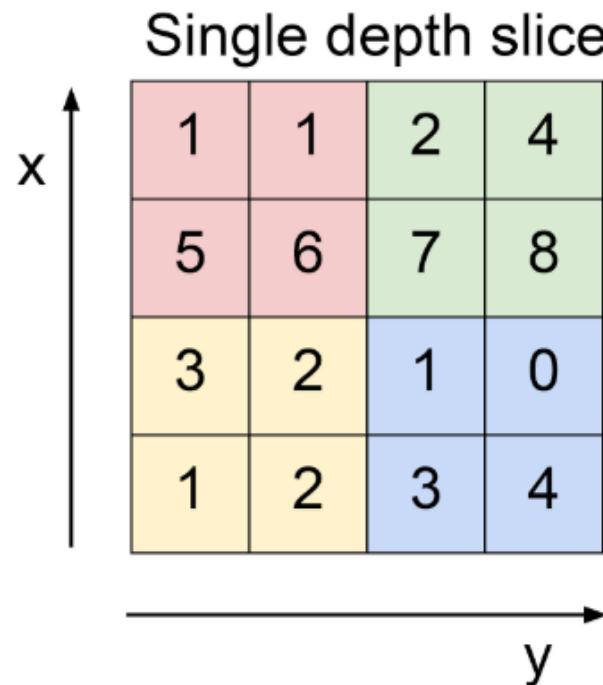
# CNN : couches de convolution



# CNN : couches de convolution



# CNN : pooling



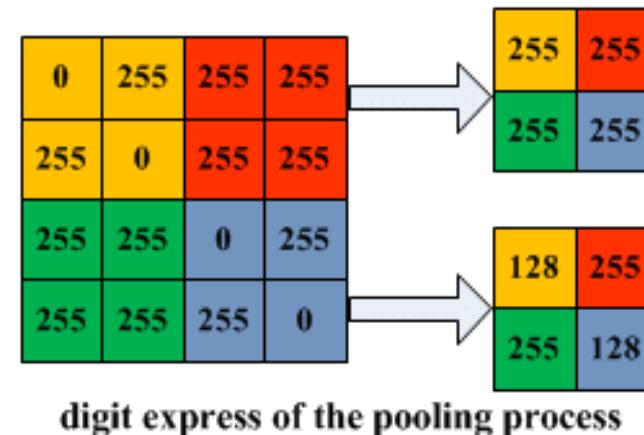
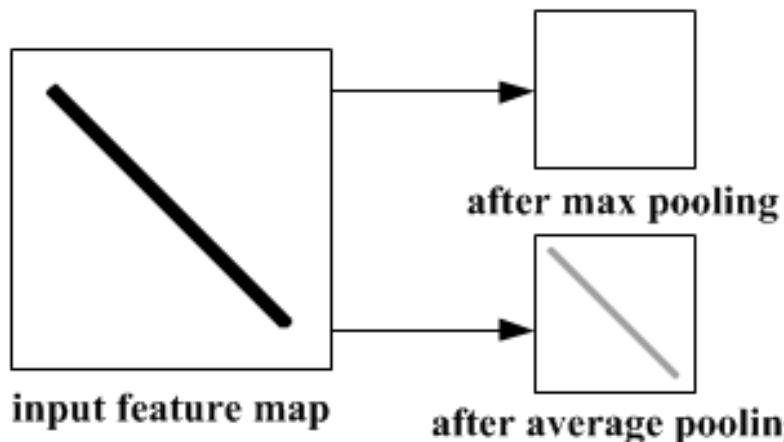
max pool with 2x2 filters  
and stride 2

The result of max pooling is a 2x2 output slice. The top-left cell contains the maximum value from the top-left 2x2 receptive field (5, 6, 1, 2), which is 6 (pink). The top-right cell contains the maximum value from the top-right 2x2 receptive field (1, 2, 3, 4), which is 4 (light green). The bottom-left cell contains the maximum value from the bottom-left 2x2 receptive field (3, 2, 1, 0), which is 3 (yellow). The bottom-right cell contains the maximum value from the bottom-right 2x2 receptive field (1, 2, 3, 4), which is 4 (blue).

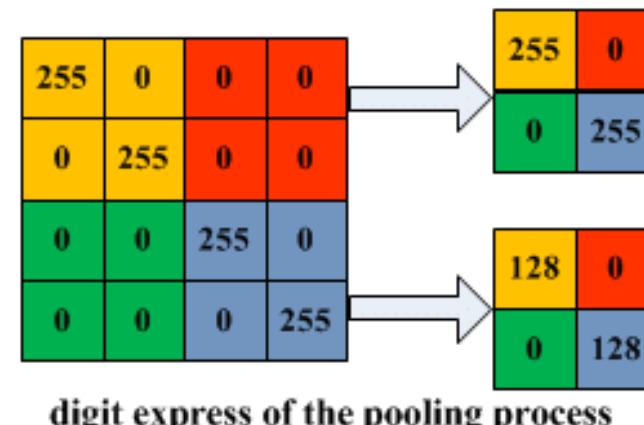
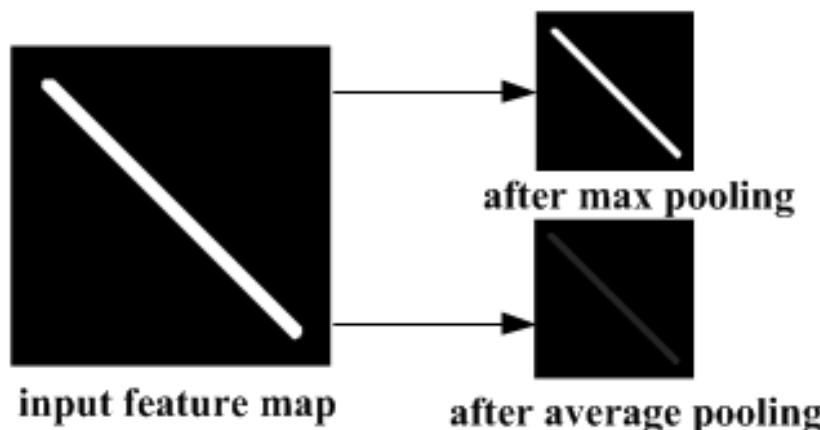
6	8
3	4

- Inséré entre deux convolutions successives
- Réduit la taille spatiale progressivement ainsi que le nombre de paramètres
- Aider au contrôle du sur apprentissage (overfitting)

# CNN : pooling

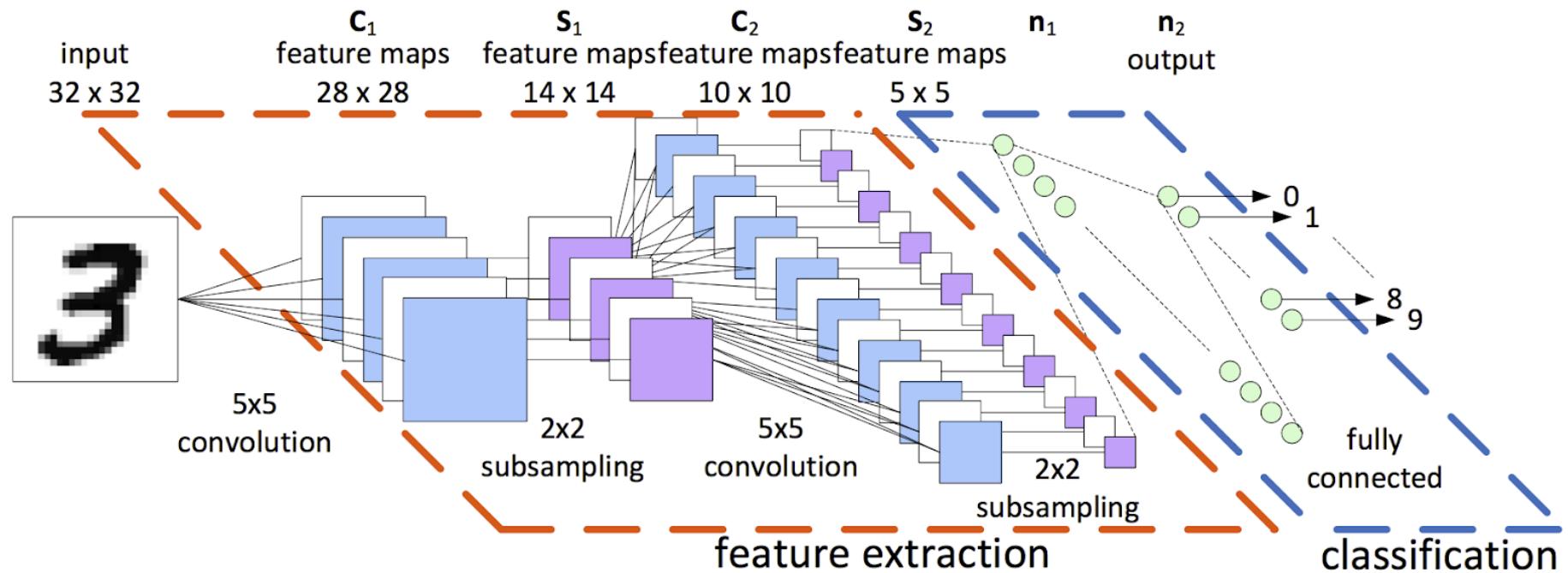


(a) Illustration of max pooling drawback

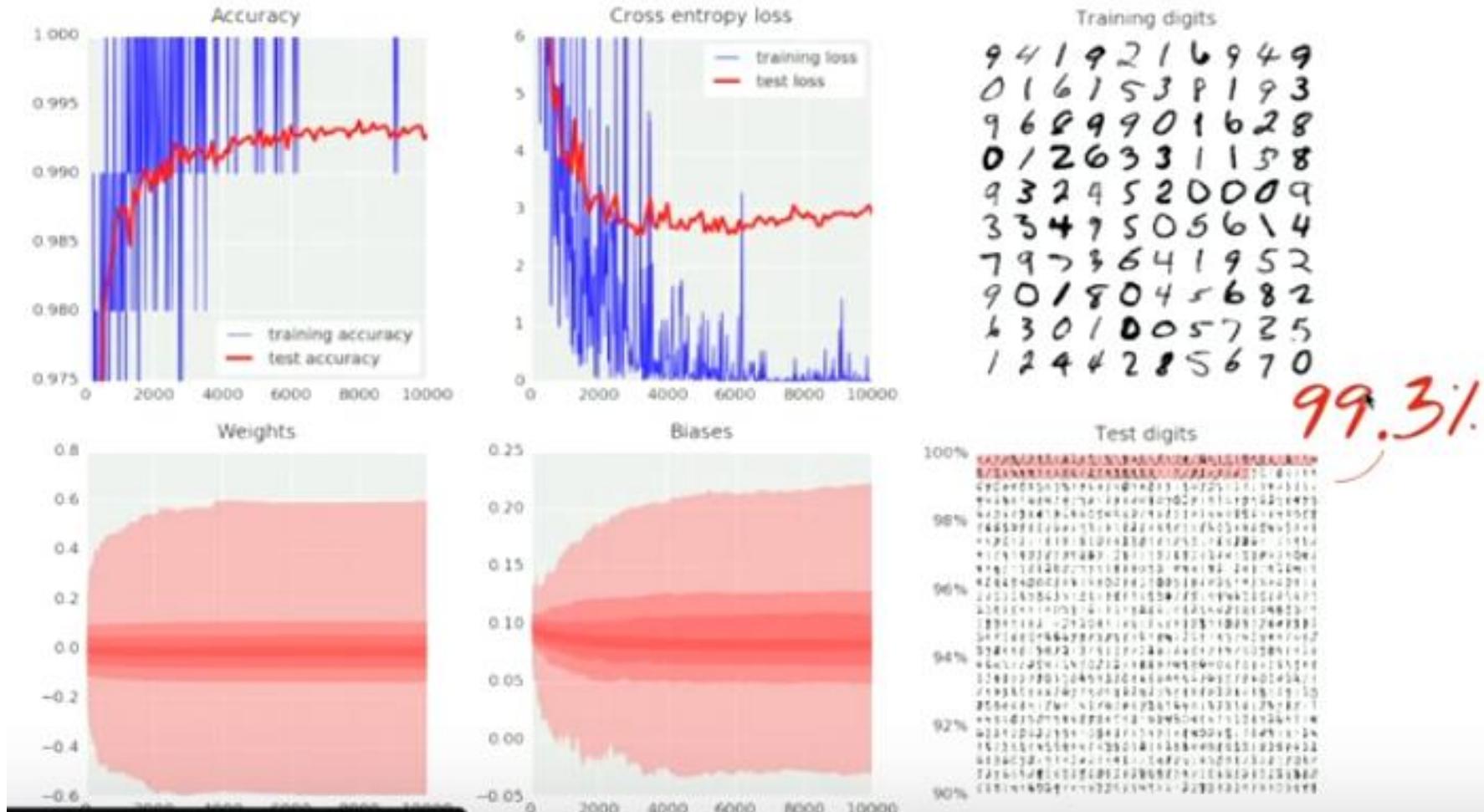


(b) Illustration of average pooling drawback

# Réseaux de neurones convolutionnels (CNNs)



# Réseaux de neurones convolutionnels (CNNs)



<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/tree/master/tensorflow-mnist-tutorial>

# Initiation aux outils : Torch Lightning

## Définition du modèle

```
class MyModel(L.LightningModule):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_size, 128),
            nn.ReLU(),
            nn.Linear(128, output_size)
        )
        self.loss_fn = nn.CrossEntropyLoss()

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = self.loss_fn(y_hat, y)
        self.log("train_loss", loss)
        return loss

    def configure_optimizers(self):
        return optim.Adam(self.parameters(), lr=1e-3)
```

# Initiation aux outils : Torch Lightning

## Préparation/chargement des données

```
class MyDataModule(L.LightningDataModule):  
    def __init__(self, batch_size=32):  
        super().__init__()  
        self.batch_size = batch_size  
  
    def prepare_data(self):  
        X = torch.randn(1000, 10)  
        y = torch.randint(0, 2, (1000,))  
        dataset = TensorDataset(X, y)  
        self.train_data, self.val_data = random_split(dataset, [800, 200])  
  
    def train_dataloader(self):  
        return DataLoader(self.train_data, batch_size=self.batch_size, shuffle=True)  
  
    def val_dataloader(self):  
        return DataLoader(self.val_data, batch_size=self.batch_size)
```

# MERCI

