

# L'apprentissage artificiel (automatique)

26 août 2020

# Terminologie clé

## Les concepts

Qu'est ce que l'apprentissage artificiel (supervisé) ?

L'apprentissage artificiel apprend à produire à partir de données d'entrée des prédictions utiles sur des données encore jamais vues.

Etiquettes (labels)

Une étiquette est la “chose” que nous essayons de prédire. Par exemple le prix d'une pizza, la race d'un animal sur une photo, l'auteur d'un texte etc

Caractéristiques (features)

Une caractéristique est une variable d'entrée. Par exemple dans un détecteur de spam, les caractéristiques pourraient être les mots du mail, l'adresse de l'expéditeur, etc.

# Terminologie clé

## Les concepts

### Exemples (Examples)

Un exemple est une instance particulière des données, par exemple un mail. On distingue :

- Exemples étiquetés :  $\{\text{caractéristique, étiquette}\} : (x,y)$ . Par exemple des mails déjà étiquetés par l'utilisateur comme spam/non spam.
- Exemples non étiquetés :  $\{\text{caractéristique}\} : (x)$ .

# Terminologie clé

## Les concepts

### Modèle (Model)

Un modèle définit la relation entre les caractéristiques et l'étiquette. Il est composé de deux phases :

- L'entraînement (training). Le modèle apprend les relations entre les caractéristiques et l'étiquette à partir d'exemples.
- L'inférence consiste à appliquer le modèle **entraîné** sur des exemples non étiquetés pour faire des prédictions  $y'$ .

# Terminologie clé

## Les familles d'apprentissage

### Régression et classification

En régression on prédit des valeurs continues. Par exemple :

- Le prix d'une pizza en fonction de sa taille, ses ingrédients etc.
- La consommation d'une voiture en fonction de sa puissance, son poids etc.
- Quelle va être la température en fonction d'un lieu et d'une date etc.

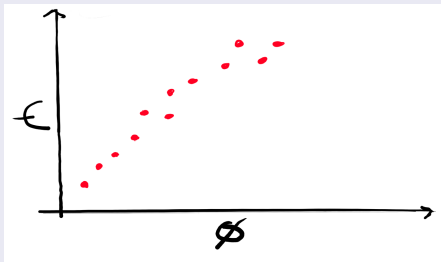
En classification on prédit des valeurs discrètes. Par exemple :

- Est-ce que cet email est un spam ou pas ?
- Quel type d'iris est cette plante ?
- Est-ce que cette image représente un chien ou un chat ?

# Un premier exemple

## La régression linéaire

### Le prix d'une pizza en fonction de sa taille

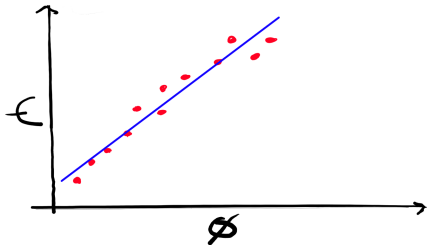


- **Les données** : on a quelques exemples de prix de pizza en fonction de la taille.
- **Le but** : trouver une règle, i.e. un modèle qui permet de lier le prix et la taille d'une pizza.

# Un premier exemple

## La régression linéaire

Le prix d'une pizza en fonction de sa taille



- Une première relation
- Un modèle linéaire.

# Un premier exemple

## La régression linéaire

### Une relation linéaire

$$y = mx + b \text{ avec}$$

- $y$  le prix d'une pizza, ce qu'on cherche à prédire,
- $x$  sa taille, la valeur de notre caractéristique d'entrée,
- $m$  la pente de la droite,
- $b$  l'ordonnée à l'origine.

### Par convention, on note :

$$y' = b + w_1 x_1 \text{ avec}$$

- $y'$  la sortie désirée, une étiquette prédite,
- $b$  le biais, aussi nommé  $w_0$ ,
- $x$  un exemple,
- $w_1$  le poids de la première caractéristique.



# Les principaux concepts

## La régression linéaire

### L'entraînement (training)

L'entraînement consiste à trouver les meilleurs poids et biais possibles à partir des exemples connus.

Dans notre cas, il s'agit de trouver un bon modèle qui permet de prédire le prix d'une pizza en fonction de sa taille, à partir des tailles et prix des pizzas que l'on connaît.

### La perte (loss)

Il existe une infinité de modèles, il faut pouvoir les comparer et mesurer leur efficacité.

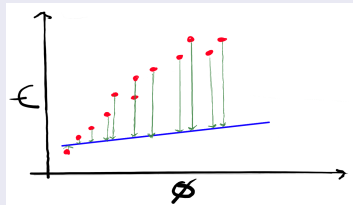
La perte est la pénalité obtenue pour les mauvaises prédictions. Pour un modèle parfait, la perte sera de 0. Un mauvais modèle aura une grande perte.

# Les principaux concepts

## La régression linéaire

### Le prix d'une pizza en fonction de sa taille

- En rouge : la perte.
- En bleu : les prédictions.



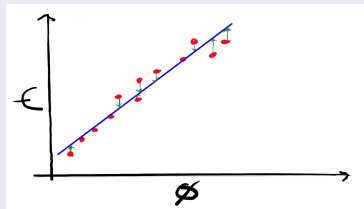
⇒ Grande perte, mauvais modèle.

# Les principaux concepts

## La régression linéaire

### Le prix d'une pizza en fonction de sa taille

- En rouge : la perte.
- En bleu : les prédictions.

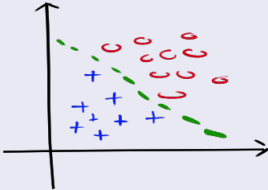


⇒ Faible perte, bon modèle.

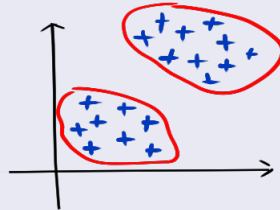
# Les principaux concepts

## Les différentes familles

### Supervisé



### Non-supervisé



# Les principaux concepts

## La perte quadratique

### Perte $L_2$ : une mesure classique

La perte quadratique est une notion connue. Elle est également appelée perte  $L_2$ .

### Erreur quadratique moyenne (MSE)

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

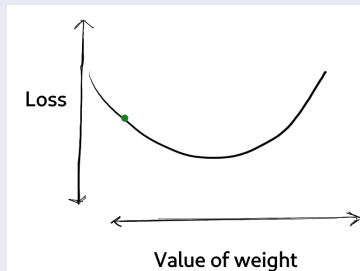
- $(x, y)$  un exemple (caractéristiques, étiquette),
- $prediction(x)$  est le résultat du modèle pour l'entrée  $x$ ,
- $D$  la base contenant l'ensemble des exemples,
- $N$  le nombre d'exemples.

# Les principaux concepts

## Réduire l'erreur quadratique

### La descente de gradient

- **En vert** : point initial.
- Calcul du gradient à partir du point initial, on a :
  - une direction.
  - une magnitude.

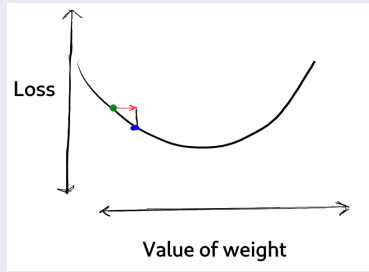


# Les principaux concepts

## Réduire l'erreur quadratique

### La descente de gradient

- **En vert** : point initial.
- **En rouge** : gradient négatif.
- **En bleu** : nouveau point.

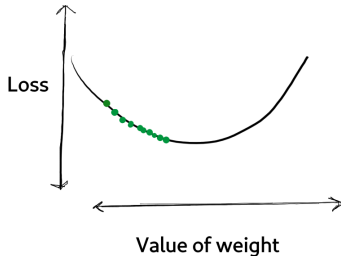


# Les principaux concepts

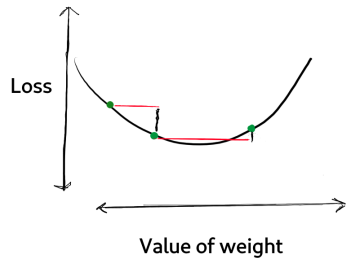
## Réduire l'erreur quadratique

### Le pas d'apprentissage

#### Le pas est trop petit



#### Le pas est trop grand



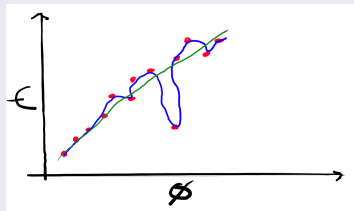


# Les principaux concepts

## Le sur-apprentissage

### Apprentissage par coeur des exemples. Illustration en régression.

- Exemple sur nos pizzas.
- **En rouge** : nos exemples.
- **En bleu** : un modèle sur-appris, on a appris les cas particuliers (par exemple des promotions).
- **En vert** : le modèle attendu.

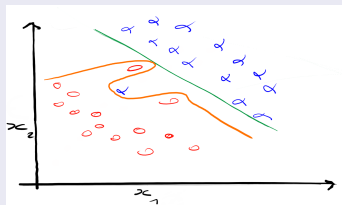


# Les principaux concepts

## Le sur-apprentissage

### Apprentissage par coeur des exemples. Illustration en classification.

- Exemple en classification.  
On cherche à séparer les croix bleues des ronds rouges.
- **En rouge et bleu** : nos exemples.
- **En orange** : un modèle sur-appris, on a appris les cas particuliers.
- **En vert** : le modèle attendu.



# Les principaux concepts

## Le sur-apprentissage

### Le sur-apprentissage

- **Faible erreur** mais **mauvais en prédiction**.
- Le modèle est trop complexe.
- On cherche donc un compromis entre la complexité du modèle et sa performance.

### L'idée

- Séparation des données en deux parties : **apprentissage** et **test**.
- L'ensemble de test **ne doit jamais** être utilisé pour la phase d'apprentissage.
- Les performances sont mesurées sur l'ensemble de test, donc sur des données *nouvelles*.

# Les principaux concepts

## Limite de 2 ensembles

- On a un modèle qui dépend d'un paramètre, comme le pas.
- On souhaite trouver le meilleur pas.
- Un scénario possible :
  - On crée 100 modèles avec 100 pas différents.
  - On apprend ces 100 modèles sur la base d'apprentissage, et on teste ces 100 modèles sur la base de test (généralisation).
  - Le meilleur paramètre/modèle obtient 5% d'erreur en généralisation (sur la base de test).
  - On choisit ce modèle pour la production, mais on s'aperçoit qu'on obtient 15% d'erreur.
  - **Problème** : On a biaisé le modèle sur notre base de test.
- **Solution** : utiliser une base de **validation** en plus.

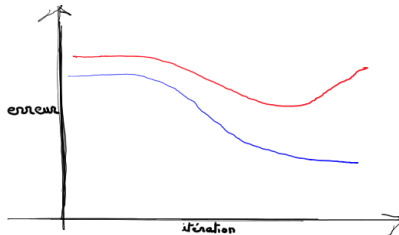
# Les principaux concepts

## La validation croisée (k-fold cross validation)

- La base d'apprentissage est découpée en  $k$  sous-ensembles.
- On sélectionne un des  $k$  sous-ensembles comme ensemble de validation.
- Le modèle est entraîné sur les  $k-1$  sous-ensembles restants.
- Le processus de validation croisée est donc répété  $k$  fois.
- Les  $k$  résultats de validation sont ensuite moyennés.
- **Avantages :**
  - Toutes les observations sont utilisées pour l'apprentissage et la validation.
  - Chaque observation est utilisée qu'une seule fois en validation.

# Les principaux concepts

## Régularisation



- Rouge : En test.
- Bleu : En apprentissage.
- → Sur-apprentissage



- On minimise :  
 $Erreur(Donnees|Modele) + \lambda complexite(modele)$ , avec  $\lambda$  le coefficient de régularisation.
- Classiquement : Régularisation  
$$L_2 = ||w||_2^2 = w_1^2 + w_2^2 + \dots + w_n^2$$

# Manipulation des données

## Représentation

### Les caractéristiques numériques

Rien à faire.

### Les caractéristiques catégorielles

- Création d'un **vocabulaire**.
- Représentation **one-hot encoding** :
  - Création d'un vecteur de 0 / 1.
  - Valeur = 1 si l'élément correspond à l'exemple,
  - 0 sinon.

# Manipulation des données

## Représentation

### Les bons réflexes

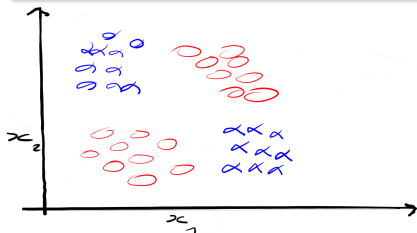
- Mise à l'échelle des données.
- Traitement des données aberrantes :
  - Valeurs de binning : transformation de numériques en  $k$  caractéristiques booléennes différentes.
  - Corrections des erreurs en supprimant les exemples non fiables.  
Un exemple peut être non fiable pour plusieurs raisons :
    - Valeurs omises,
    - Doublons,
    - Etiquettes erronées,
    - Valeurs de caractéristiques erronées.



# Croisement de caractéristiques

## Le problème du XOR

- Problème non linéaire.
- Aucune séparatrice linéaire  $y = b + w_1x_1 + w_2x_2$



## Solution

- On définit  $x_3 = x_1x_2$
- $y =$   
 $b + w_1x_1 + w_2x_2 + w_3x_3$

# Classification : mesure de performance

## La précision ?

- On a deux classes **A** et **B**.
- Notre base d'apprentissage contient 9000 exemples étiquetés **A** et 1000 étiquetés **B**.
- Notre classifieur obtient 90% de précision.
- Comment interpréter ce résultat ?
- **La précision seule n'est en général pas un bon critère de performance.**

# Classification : mesure de performance

Vrai, faux, positif vs négatif

## Le garçon qui criait au loup !

Il était une fois un jeune berger qui gardait tous les moutons des habitants de son village[...]

Un jour qu'il s'ennuyait particulièrement [...] il hurla : "Au loup!"[...] les villageois bondirent hors de leurs maisons et grimpèrent sur la colline pour chasser le loup. Mais ils ne trouvèrent que le jeune garçon qui riait[...]

Environ une semaine plus tard, le jeune homme [...] se remit à crier : "Au loup! Un loup dévore le troupeau!"

Une nouvelle fois, les villageois se précipitèrent pour le secourir. Mais point de loup[...]

Enfin, un soir d'hiver, alors que le berger rassemblait son troupeau pour le ramener à la bergerie, un vrai loup approcha des moutons[...]

Il se précipita sur la colline et hurla : "Au loup!"[...]

Mais pas un villageois ne bougea[...]

# Classification : mesure de performance

Vrai, faux, positif vs négatif

Supposons :

- “Loup” est une classe positive.
- “Pas de loup” est une classe négative.

	Vrais positifs (VP)	Faux positifs (FP)
Réalité	Un loup est présent	Aucun loup n'est présent
Le garçon	crie au loup	crie au loup
Récompense	correct	incorrect
	Faux négatifs (FN)	Vrais négatifs (VN)
Réalité	Un loup est présent	Aucun loup n'est présent
Le garçon	ne dit rien	ne dit rien
Récompense	incorrect	correct

- Un vrai positif prédit correctement une classe positive. Un vrai négatif prédit correctement une classe négative.
- Un faux positif prédit incorrectement une classe positive. Un faux négatif prédit incorrectement une classe négative.

# Classification : mesure de performance

## Accuracy : prédictions correctes sur le total

- Terme français “justesse” mais peu utilisé.
- $accuracy = \frac{VP+VN}{VP+VN+FP+FN}$

## La précision

- $precision = \frac{VP}{VP+FP}$

## Le rappel

- $rappel = \frac{VP}{VP+FN}$

# Classification : mesure de performance

## Exercice

On cherche à prédire si une transaction bancaire est frauduleuse ou non.

- On a une base d'apprentissage contenant 10000 exemples : 9000 transactions correctes et 1000 fraudes.
- On a une base de test contenant 1000 exemples : 900 transactions correctes et 100 fraudes.
- On a appris notre modèle, et voici les résultats en test :
  - 890 correctes qui l'étaient.
  - 80 correctes qui étaient des fraudes.
  - 10 fraudes qui étaient correctes.
  - 20 fraudes qui étaient des fraudes.
- **Questions :**
  - Quelle répartition de VP/FP/FN/VN ?
  - Calculez l'accuracy.
  - Calculez la précision.
  - Calculez le rappel.

# Classification : mesure de performance

## Correction de l'exercice

VP = 20      FP=10

FN=80      VN=890

- $accuracy = \frac{890+20}{1000} = 91\%$
- $precision = \frac{20}{20+10} = 66\%$
- $rappel = \frac{20}{20+80} = 20\%$

# Mesure de performance

## Score $F_1$

Afin de comparer deux classifieurs il est pratique de n'avoir qu'un seul score qui combine précision et rappel.

Le score  $F_1$  correspond à la moyenne harmonique :

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{rappel}} = 2 \times \frac{precision \times rappel}{precision + rappel} = \frac{VP}{VP + \frac{FN+FP}{2}}$$

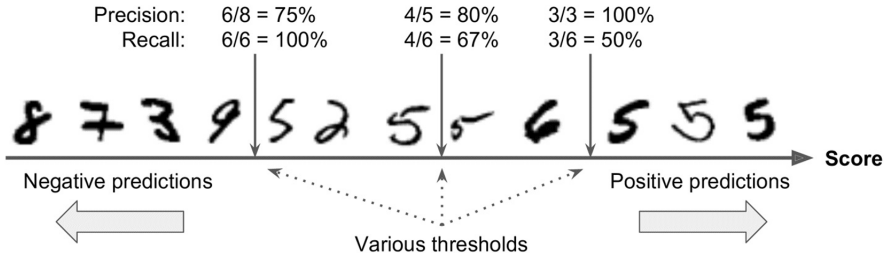
Ce score favorise des classifieurs qui ont une précision et un recall similaires.

Pour certains problèmes il est plus pertinent de s'intéresser à une mesure plutôt qu'à une autre.



# Mesure de performance

Score  $F_1$



# La courbe ROC

## ROC (Receiver Operating Characteristic Curve)

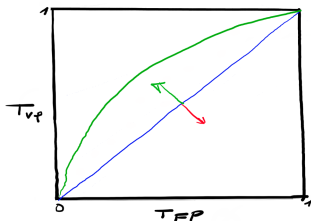
Il s'agit d'un graphe qui montre la performance d'un modèle pour tous les seuils. Il s'agit du taux des vrais positifs en fonction de celui des faux positifs :

- $taux_{vp} = \frac{VP}{VP+FN}$

- $taux_{fp} = \frac{FP}{FP+VN}$

La courbe ROC montre ces taux pour différents seuils.

# LA courbe ROC



- La droite en bleu représente un classificateur aléatoire.
- La courbe en verte un classifieur correct.

- (0, 0) : toujours *négatif*, donc aucun FP ni aucun VP.
- (1, 1) : toujours *positif*, donc aucun VN ni aucun FN.
- (0, 1) : aucun FP ni aucun FN, il ne se trompe jamais.
- (1, 0) : aucun VN ni aucun VP, il se trompe toujours (il faut l'inverser).
- **L'aire sous la courbe représente la probabilité que le modèle classe un exemple aléatoire positif au dessus d'un exemple aléatoire négatif.**

# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron

### Régression linéaire, erreur quadratique moyenne

- Règle de décision :  $y = W'X$

*Si c'est supérieur à un seuil c'est une classe, sinon c'est une autre.*

- Fonction de coût :  $L(W, y_i, X_i) = \frac{1}{2}(y_i - W'X_i)^2$

*On minimise la différence entre la sortie désirée et la sortie du système.*

- Gradient du coût :  $\frac{\partial L(W, y_i, X_i)}{\partial W} = -(y_i - W(t)'X_i)X_i$

*Si la sortie est plus petite que la sortie désirée alors on rajoute au vecteur de poids le vecteur d'entrées multiplié par l'erreur.*

- Mise à jour :  $W(t+1) = W(t) + \eta(t)(y_i - W(t)'X_i)X_i$

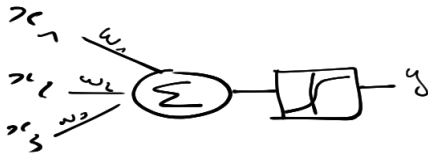
*On soustrait au poids le gradient du coût.*



Convergence : minimisation de la moyenne de la fonction coût sur tous les exemples d'apprentissage.

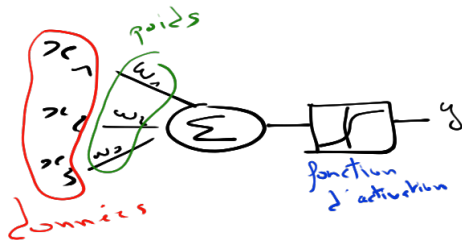
# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron



# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron



# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron

### Machine linéaire, le perceptron

- Règle de décision :  $y = F(W'X)$

*F est la fonction seuil.*

- Fonction de coût :  $L(W, y_i, X_i) = (F(W'X_i) - y_i)W'X_i$

*On minimise la différence entre la sortie désirée et la sortie du système.*

- Gradient du coût :  $\frac{\partial L(W, y_i, X_i)}{\partial W} = -(y_i - F(W(t)'X_i))X_i$

*Si la sortie est plus petite que la sortie désirée alors on rajoute au vecteur de poids le vecteur d'entrées multiplié par l'erreur.*

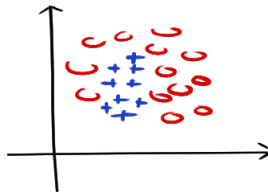
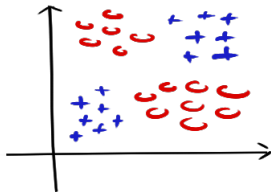
- Mise à jour :  $W(t+1) = W(t) + \eta(t)(y_i - F(W(t)'X_i))X_i$

*On soustrait au poids le gradient du coût.*

(Sans oublier le biais !)

# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron

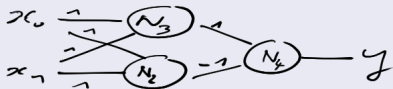




# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron

### Le XOR



Activation, fonction seuil :  
Si  $> \text{seuil}$  alors 1 sinon 0

$$\Leftrightarrow \sum_{i=1}^d w_i x_i - \text{seuil}$$

# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron

### Le XOR



$Seuil_{N_3} = 0, Seuil_{N_2} = 1$

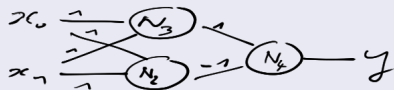
Activation, fonction seuil :  
Si  $> seuil$  alors 1 sinon 0

$$\Leftrightarrow \sum_{i=1}^d w_i x_i - seuil$$

# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron

### Le XOR



$Seuil_{N_3} = 0, Seuil_{N_2} = 1$

Activation, fonction seuil :  
Si  $> seuil$  alors 1 sinon 0

$$\Leftrightarrow \sum_{i=1}^d w_i x_i - seuil$$

$x_0$	$x_1$	$\sigma_3$	$\sigma_2$	$y_3$	$y_2$	$y$
0	0	0	0	0	0	0
0	1	1	1	1	0	1
1	0	1	1	1	0	1
1	1	2	2	1	1	0

# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron

### Le XOR

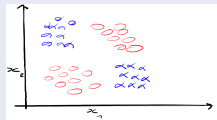


$$Seuil_{N_3} = 0, Seuil_{N_2} = 1$$

$$\begin{cases} y_3 = w_{0,3}x_0 + w_{1,3}x_1 - seuil_{N_3} \\ y_2 = w_{0,2}x_0 + w_{1,2}x_1 - seuil_{N_2} \\ y = w_{3,4}y_3 + w_{2,4}y_2 \end{cases}$$

Activation, fonction seuil :  
Si  $> seuil$  alors 1 sinon 0

$$\Leftrightarrow \sum_{i=1}^d w_i x_i - seuil$$



# Rappel sur les réseaux de neurones

## Rappel et limite du perceptron

### Le XOR

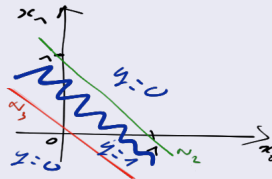


$$Seuil_{N_3} = 0, Seuil_{N_2} = 1$$

$$\begin{cases} y_3 = x_0 + x_1 \\ y_2 = x_0 + x_1 - 1 \\ y = y_3 - y_2 \end{cases}$$

Activation, fonction seuil :  
Si  $> seuil$  alors 1 sinon 0

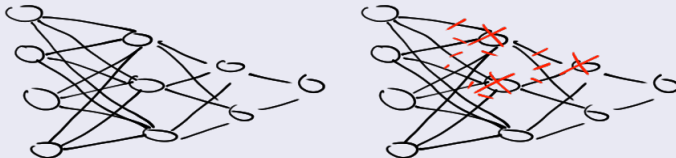
$$\Leftrightarrow \sum_{i=1}^d w_i x_i - seuil$$



# Les réseaux de neurones

## Le dropout

Le dropout : suppression de neurone sur un pas



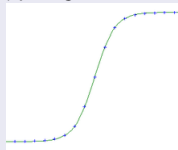
# Rappel sur les réseaux de neurones

Les principales fonctions d'activation [\[images wikipedia\]](#)

## La sigmoid

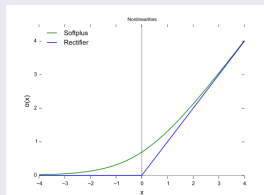
$$F(x) = \frac{1}{1 + e^{-\lambda x}}$$

$\lambda = 5$



## La ReLU (rectified linear unit)

$$F(x) = \max(0, x)$$



# Rappel sur les réseaux de neurones

## Réseaux multi-classes

Par exemple, on veut apprendre à distinguer une pomme, une orange et une banane.

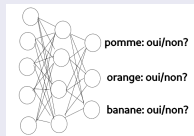
### Un contre tous (one vs all)

Pour  $N$  classes, on va créer  $N$  classifieurs binaires distincts.

- Pomme contre non pomme,
- orange contre non orange,
- banane contre non banane.

### Plus efficacement

$N$  neurones de sortie. Chacun de ces neurones représente une classe.





# La bibliothèque tensorflow

## TensorFlow

- Bibliothèque d'apprentissage artificiel.
- Développée par Google, 2015.
- Utilisée en recherche et en industrie.
- Interface en plusieurs langages. Dans ce cours nous nous limiterons à Python.

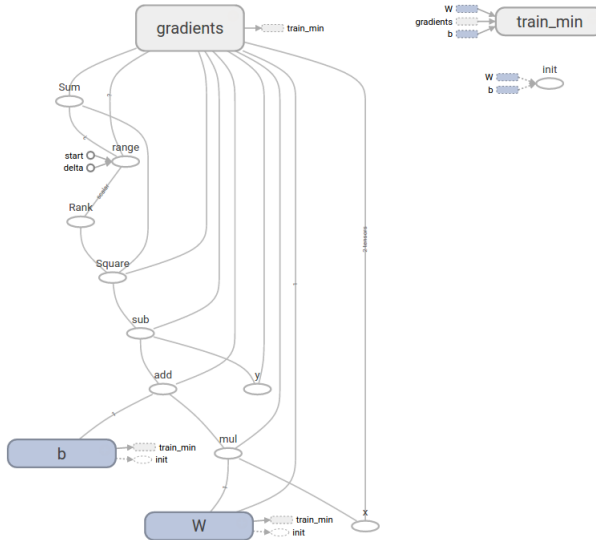
## Mais aussi ...

- mxnet,
- torch,
- pytorch,
- ...

# La bibliothèque TensorFlow

- <https://www.tensorflow.org/>
- Très bien documentée.
- Tutorial à python disponible :  
<http://www-lisic.univ-littoral.fr/~teytaud/files/Cours/Apprentissage/tutoPython.pdf>.
- Graphe de nœuds TensorFlow : série d'opérations.
- Pour faire tourner un programme il faut :
  - Construire le graphe.
  - Exécuter le graphe.

# Exemple de graphe [Doc tensorflow]



# Exemple de graphe [\[Doc tensorflow\]](#)

Code :

```
import tensorflow as tf
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = tf.add(a,b)
sess = tf.Session()
print(sess.run(adder_node, {a:3, b:4.5}))
print(sess.run(adder_node, {a:[1,3], b:[2,4]}))
```

Sortie :

```
7.5
[ 3.  7.]
```

[\[Doc tensorflow\]](#)

## Analysons le code suivant [\[Doc tensorflow\]](#)

```
# Model parameters
```

```
W = tf.Variable([.3], dtype=tf.float32)
```

```
b = tf.Variable([-0.3], dtype=tf.float32)
```

```
# Model input and output
```

```
x = tf.placeholder(tf.float32)
```

```
linear_model = W * x + b
```

```
y = tf.placeholder(tf.float32)
```

```
# loss (sum of the squares)
```

```
loss = tf.reduce_sum(tf.square(linear_model - y))
```

```
# optimizer
```

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
```

```
train = optimizer.minimize(loss)
```

# Analysons le code suivant [Doc tensorflow]

```
# training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(1000):
    sess.run(train, {x: x_train, y: y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss =
    sess.run([W, b, loss], {x: x_train, y: y_train})
print("W: %s b: %s loss: %s"
      %(curr_W, curr_b, curr_loss))
```

## Analysons le code suivant [Doc tensorflow]

```
W: [-0.99999969] b: [ 0.99999082] loss: 5.69997e-11
```

# Un premier exemple avec Keras

## Classification d'images

### Keras

- Une API haut niveau permettant de créer des réseaux de neurones.
- Plus simple à utiliser que tensorflow.
- Écrite en python.
- Supporte les réseaux de neurones classiques et profonds.
- Facilité d'exécution CPU/GPU.

### La base MNIST

- Reconnaissance de chiffres manuscrits.
- 60k images d'apprentissage, 10k de test.
- Normalisées, centrées,  $28 \times 28$  pixels.
- <http://yann.lecun.com/exdb/mnist>



# Un premier exemple avec Keras

## Les librairies

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
```

# Un premier exemple avec Keras

## Chargement des données

```
mnist = keras.datasets.mnist  
(train_images, train_labels),  
 (test_images, test_labels) =  
 mnist.load_data()
```

# Un premier exemple avec Keras

## Visualisation des données

### Structures des données

```
print(train_images.shape)
print(train_labels.shape)
print(test_images.shape)
print(test_labels.shape)
```

### Résultat

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

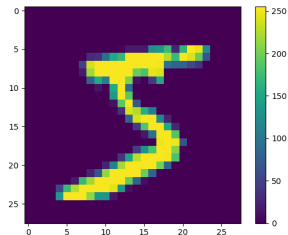
# Un premier exemple avec Keras

## Visualisation des données

### Un exemple

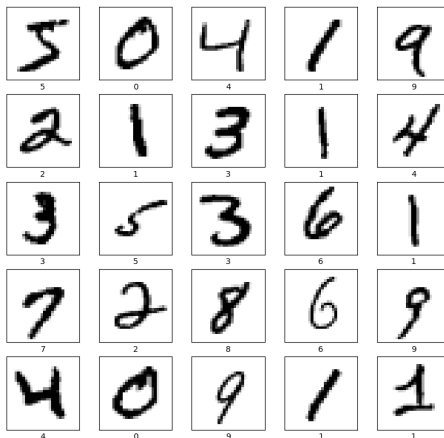
```
plt.figure()  
plt.imshow(train_images[0])  
plt.colorbar()  
plt.grid(False)  
plt.show()
```

### Résultat



# Un premier exemple avec Keras

Affichage des 25 premiers exemples et des étiquettes correspondantes



# Un premier exemple avec Keras

## Construction du modèle

```
model = keras.Sequential([  
    # Le modèle Sequential est un ensemble linéaire de couches  
    keras.layers.Flatten(input_shape=(28,28)),  
    # Transforme une matrice 28x28 en un tableau de 784  
    keras.layers.Dense(128, activation=tf.nn.relu),  
    # Couche entièrement connectée de 128 neurones  
    keras.layers.Dense(10, activation=tf.nn.softmax)  
    # Couche entièrement connectée de 10 neurones:  
    # 10 probabilités de sortie  
])
```

# Un premier exemple avec Keras

## Construction du modèle

```
model.compile(optimizer='sgd',  
# On choisit la descente de gradient  
# stochastique comme optimisation  
              loss='sparse_categorical_crossentropy',  
# Définition de la mesure de perte  
# Ici l'entropie croisée  
              metrics=['accuracy'])  
# Définition de la mesure de performance  
# que l'on souhaite utiliser. Ici la accuracy
```

# Un premier exemple avec Keras

## Entrainement du modèle

```
model.fit(train_images, train_labels, epochs=5)
```

### Affichage

```
Epoch 1/5  
60000/60000 [=====] -  
      3s 45us/step - loss: 0.6562 - acc: 0.8352  
Epoch 2/5  
60000/60000 [=====] -  
      3s 49us/step - loss: 0.3378 - acc: 0.9061  
Epoch 3/5  
60000/60000 [=====] -  
      3s 48us/step - loss: 0.2889 - acc: 0.9182  
Epoch 4/5  
60000/60000 [=====] -  
      2s 37us/step - loss: 0.2583 - acc: 0.9284  
Epoch 5/5  
60000/60000 [=====] -  
      2s 35us/step - loss: 0.2349 - acc: 0.9339
```



# Un premier exemple avec Keras

## Entrainement du modèle









































































































```
test_loss, test_acc =  
    model.evaluate(test_images, test_labels)  
print("perte: {}, accuracy: {}".format(  
    test_loss, test_acc))
```

### Affichage

```
perte: 0.2213208372950554, accuracy: 0.9381
```

# Un premier exemple avec Keras

## Les 105 premiers exemples de test

 7 100% (7)	 2 96% (2)	 1 98% (1)	 0 100% (0)	 4 94% (4)	 1 99% (1)	 4 97% (4)	 9 95% (9)	 5 97% (5)	 9 92% (9)	 0 98% (0)	 6 77% (6)	 9 95% (9)	 0 100% (0)	 1 100% (1)
 5 94% (5)	 9 94% (9)	 7 99% (7)	 3 84% (3)	 4 98% (4)	 9 89% (9)	 6 98% (6)	 6 95% (6)	 5 99% (5)	 4 87% (4)	 0 100% (0)	 7 97% (7)	 4 98% (4)	 0 100% (0)	 1 96% (1)
 3 100% (3)	 1 97% (1)	 3 99% (3)	 4 39% (4)	 7 99% (7)	 2 99% (2)	 7 98% (7)	 1 99% (1)	 2 48% (2)	 1 98% (1)	 1 95% (1)	 7 95% (7)	 4 93% (4)	 2 95% (2)	 3 92% (3)
 5 95% (5)	 1 70% (1)	 2 95% (2)	 4 99% (4)	 4 98% (4)	 6 99% (6)	 3 100% (3)	 5 94% (5)	 5 90% (5)	 6 97% (6)	 0 99% (0)	 4 100% (4)	 1 97% (1)	 9 96% (9)	 5 80% (5)
 7 99% (7)	 8 96% (8)	 9 98% (9)	 3 54% (3)	 7 97% (7)	 4 83% (4)	 6 49% (6)	 4 99% (4)	 3 100% (3)	 0 99% (0)	 7 100% (7)	 0 100% (0)	 2 96% (2)	 9 55% (9)	 1 98% (1)
 7 99% (7)	 3 96% (3)	 2 59% (2)	 9 99% (9)	 7 100% (7)	 7 80% (7)	 6 98% (6)	 2 100% (2)	 7 96% (7)	 8 96% (8)	 4 100% (4)	 7 100% (7)	 3 80% (3)	 6 100% (6)	 1 99% (1)
 3 100% (3)	 6 100% (6)	 9 47% (9)	 3 100% (3)	 1 99% (1)	 4 100% (4)	 1 81% (1)	 7 93% (7)	 6 97% (6)	 9 100% (9)	 6 95% (6)	 0 99% (0)	 5 100% (5)	 4 100% (4)	 9 88% (9)

# Un premier exemple avec Keras

## Et avec un peu plus de temps



# A vous de jouer !

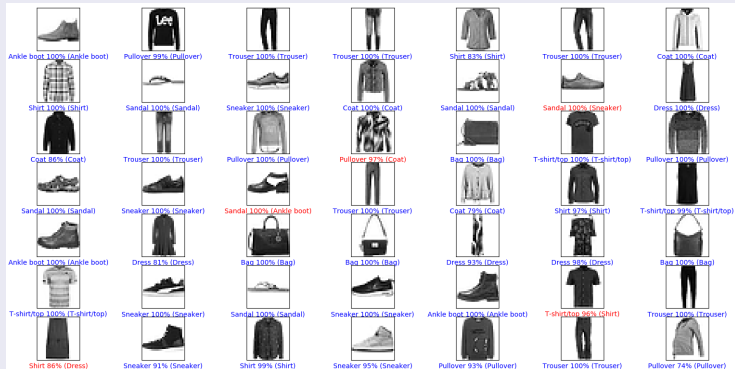
Base : `keras.datasets.fashion_mnist`



# A vous de jouer !

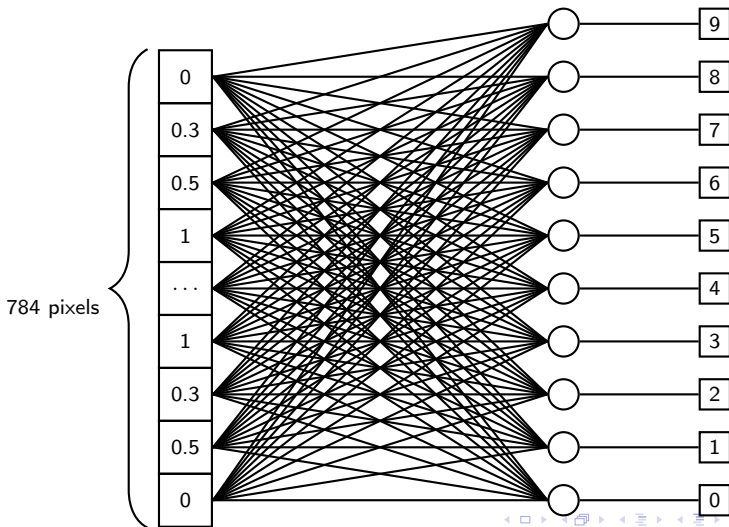
## Résultats attendus

perte: 0.3492206485748291, accuracy: 0.8947



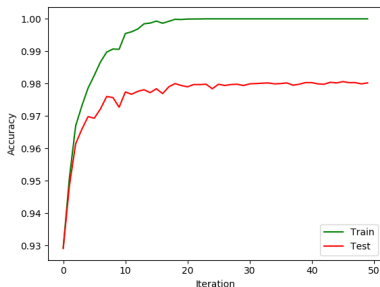
# Amélioration de notre réseau

## Réseau actuel



# Amélioration de notre réseau

## Réseau actuel



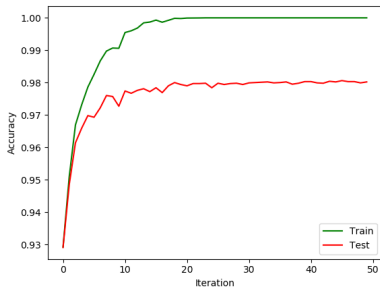
## Problèmes

- Sur-apprentissage.
- Perte de la 2D.

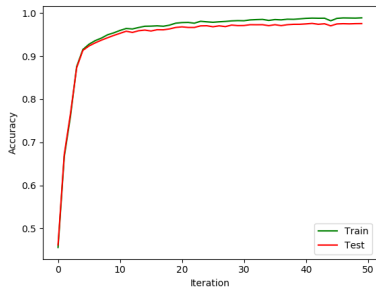
# Amélioration de notre réseau

## Dropout à 0.5

Sans dropout



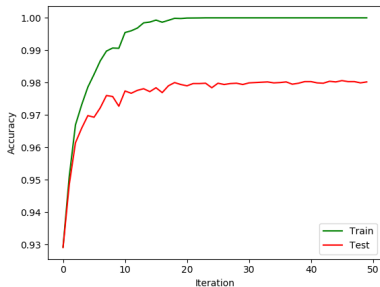
Avec dropout



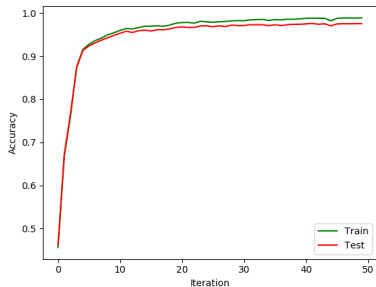


# Amélioration de notre réseau Dropout à 0.5

Sans dropout



Avec dropout



# Les réseaux convolutifs

## Vision globale

- Création de réseaux avec de nombreuses couches :
  - Les premières couches extraient des caractéristiques *bas niveau*.
  - Ces caractéristiques sont similaires partout dans l'image.
  - Les dernières couches reconnaissent des formes *haut niveau*.
  - Seule la dernière couche exploite les informations de sortie.

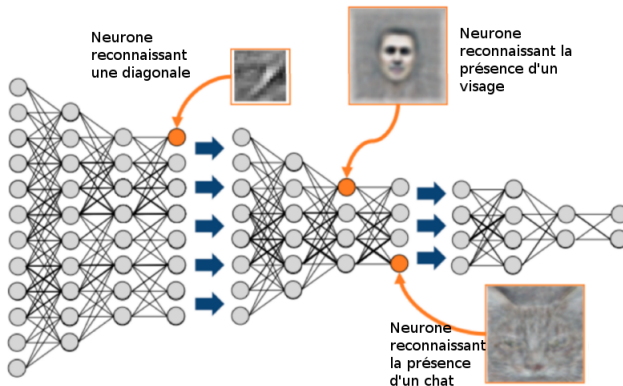
# Les réseaux convolutifs

## Vision globale

- Pavage : découpage de l'image en petites zones
- Chaque tuile correspond à un neurone.
- Correspond à une convolution (même traitement pour chaque receptr) :
  - Réduit l'empreinte mémoire.
  - Invariance par translation.
- Subsampling (pooling, mise en commun) : on garde le max de chaque zone (idée : ce que le neurone a le mieux appris à reconnaître).

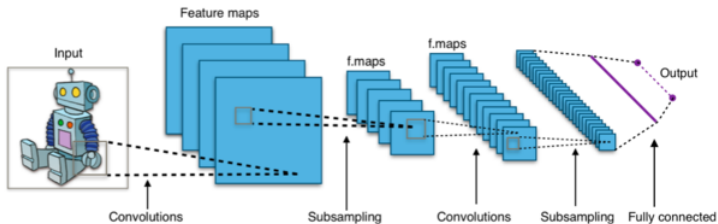
# Les réseaux convolutifs

## Vision globale



# Les réseaux convolutifs

## Vision globale



[wikipedia]

# Les réseaux convolutifs

## Le fonctionnement plus en détail

### Idée

- Les réseaux convolutifs comparent les images fragment par fragment. L'idée est d'extraire des motifs (caractéristiques) similaires entre les images, et ceci indépendamment de la position du noyau de convolution (motif dans l'image).
- Chaque motif est comme une mini-image (une matrice).

# Les réseaux convolutifs

## Le fonctionnement plus en détail

### Convolution

- Le réseau cherche donc la présence d'un motif dans une image, mathématiquement le processus utilisé pour réaliser cette tâche s'appelle une convolution.
- Soit une image  $I$ , un motif  $K$  de taille  $h \times w$ , l'image calculée est le recouvrement de l'image avec le motif de toutes les

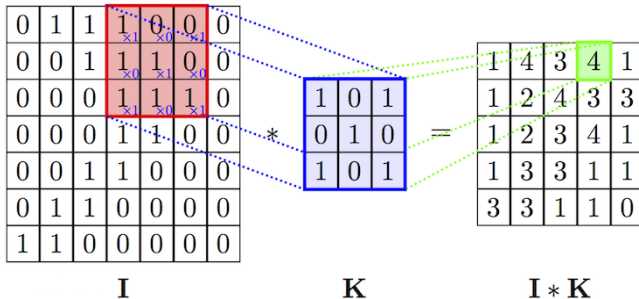
façons possibles :

$$(I \times K)_{xy} = \sum_{i=1}^h \sum_{j=1}^w K_{ij} \cdot I_{x+i-1, y+j-1}$$

# Les réseaux convolutifs

## Le fonctionnement plus en détail

### La convolution [\[cambridgespark\]](#)



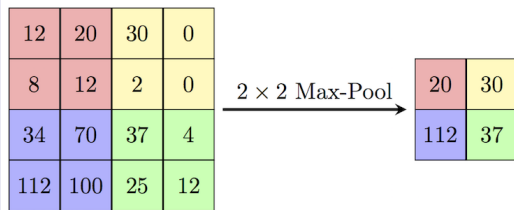


# Les réseaux convolutifs

## Le fonctionnement plus en détail

### Pooling

- Réduction de la taille de l'image tout en préservant les informations importantes.
- On définit une sous zone (généralement une matrice  $2 \times 2$  ou  $3 \times 3$ ) et on garde le max (information importante) pour chaque zone.



# Les réseaux convolutifs

## La convolution avec Keras

```
k1 = [  
    [0, 1, 0],  
    [1, -4, 1],  
    [0, 1, 0]  
]  
  
k_rand = -1.0 + 1.0 *  
    np.random.rand(3, 3)
```

```
k_vertical = [  
    [-1, 0, 1],  
    [-2, 0, 2],  
    [-1, 0, 1]  
]  
  
plt.figure()  
plt.subplot(2, 2, 1)  
plt.imshow(c, cmap='gray')  
plt.axis('off')  
plt.title('original image')
```

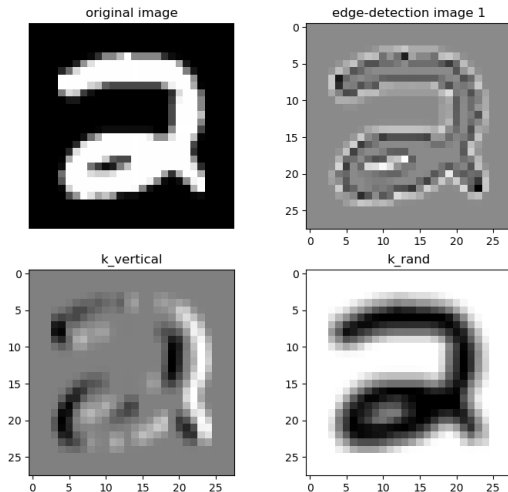
# Les réseaux convolutifs

## La convolution avec Keras

```
plt.subplot(2, 2, 2)
c_digit = signal.convolve2d(c,
                             k1, boundary='symm', mode='same')
plt.imshow(c_digit, cmap='gray')
plt.title('edge-detection')
plt.subplot(2, 2, 3)
c_digit = signal.convolve2d(c,
                             k_vertical, boundary='symm', mode='same')
plt.imshow(c_digit, cmap='gray')
plt.title('k_vertical')
plt.subplot(2, 2, 4)
c_digit = signal.convolve2d(c,
                             k_rand, boundary='symm', mode='same')
plt.imshow(c_digit, cmap='gray')
plt.title('k_rand')
plt.show()
```

# Les réseaux convolutifs

## La convolution avec Keras



# Les réseaux convolutifs

## Un réseau convolutif avec Keras

```
# input dimensions
num_train, img_rows, img_cols = x_train.shape
depth = 1
x_train = x_train.reshape(x_train.shape[0],
                           img_rows, img_cols, depth)
x_test = x_test.reshape(x_test.shape[0],
                         img_rows, img_cols, depth)
input_shape = (img_rows, img_cols, depth)
# number of convolutional filters to use
nb_filters = 32
# pooling size
pool_size = (2, 2)
# convolution kernel size
kernel_size = (3, 3)
```

# Les réseaux convolutifs

## Un réseau convolutif avec Keras

```
# Create a simple model with pooling and dropout
model = Sequential()
model.add(Conv2D(nb_filters, kernel_size=kernel_size,
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(nb_classes, activation='softmax'))
model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
model.summary()
```

# Les réseaux convolutifs

## Un réseau convolutif avec Keras

```
model.fit(x_train, y_train, batch_size=batch_size,  
          epochs=nb_epoch, verbose=1,  
          validation_data=(x_test, y_test))  
score = model.evaluate(x_test, y_test, verbose=0)  
print('Test loss: ', score[0])  
print('Test accuracy: ', score[1])
```

### Sur 20 itérations

Test loss: 0.04844768892182037

Test accuracy: 0.9844

### Et en forçant un peu ...

Test loss: 0.022147728689857444

Test accuracy: 0.9949