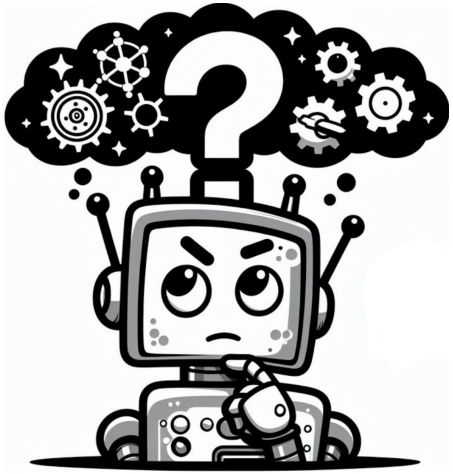
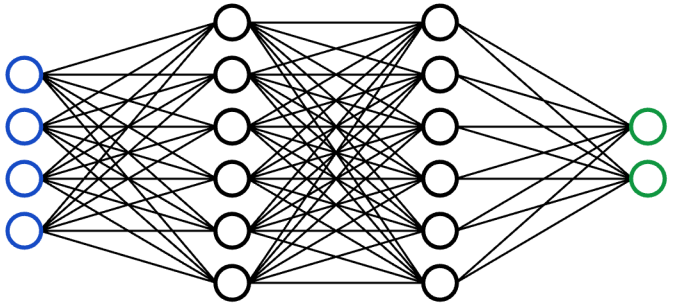
Intelligence Artificielle

Introduction aux Réseaux de Neurones

GDG Le Mans

TensorFlow





Plan

I. Traitement Automatique : dont les réseaux de neurones
II. TensorFlow : présentation
III. TensorFlow : TP
IV. Métriques d’évaluation

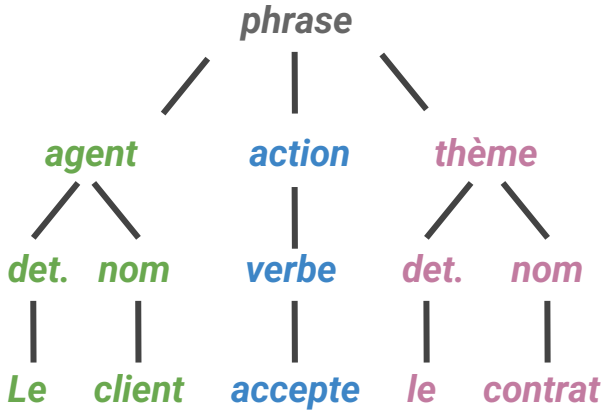
Traitement Automatique

1. Systèmes à base de règles
2. Systèmes d'apprentissage automatique (dont les réseaux de neurones)

Systèmes à base de règles

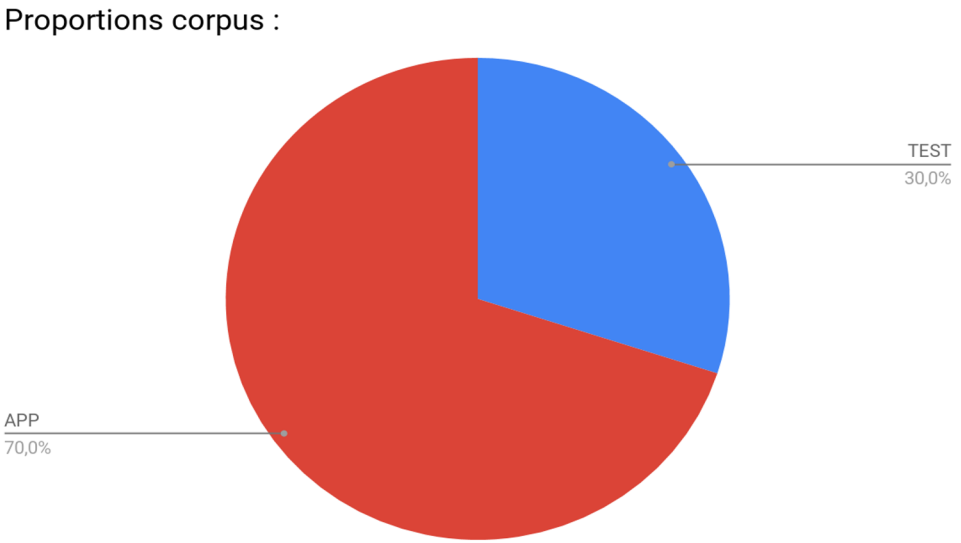
- Tâche :
  - analyse de texte
- Règles (grammaire) :
  - *phrase = agent + action + thème*
  - *agent = déterminant + nom*
  - *action = verbe*
  - *thème = déterminant + nom*
- Phrase donnée :
  - *“Le client accepte le contrat”*

- Traitement :

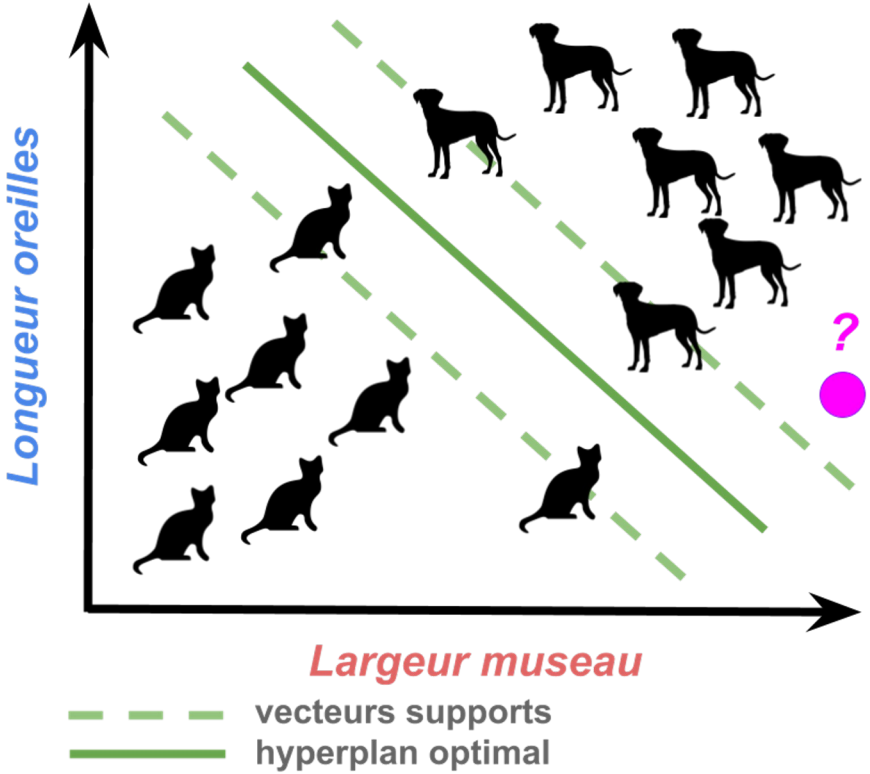


Systèmes d’apprentissage automatique (supervisé)

- Pas de règles
- Apprentissage par l’exemple
- Nécessite de disposer d’un corpus suffisamment large d’associations correctes
- Une partie du corpus est dédiée à l’apprentissage
- Une partie du corpus est dédiée au test (performances réelles)

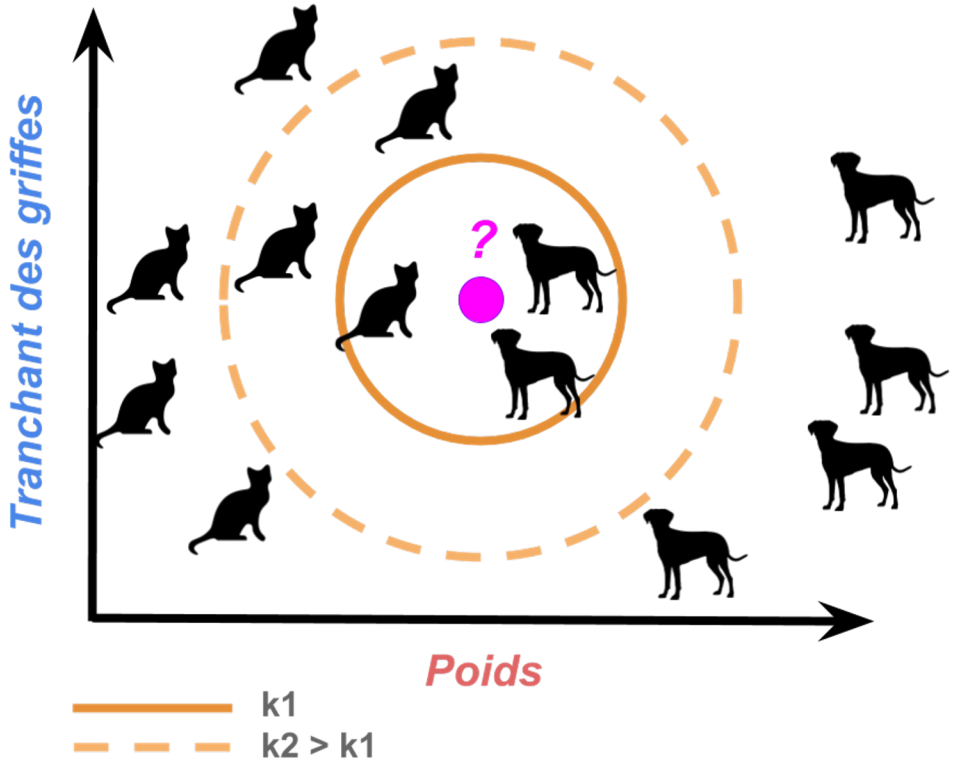


Machines à vecteur support



- Détermine un plan de séparation à partir des données connues
- Peut ensuite catégoriser un élément inédit

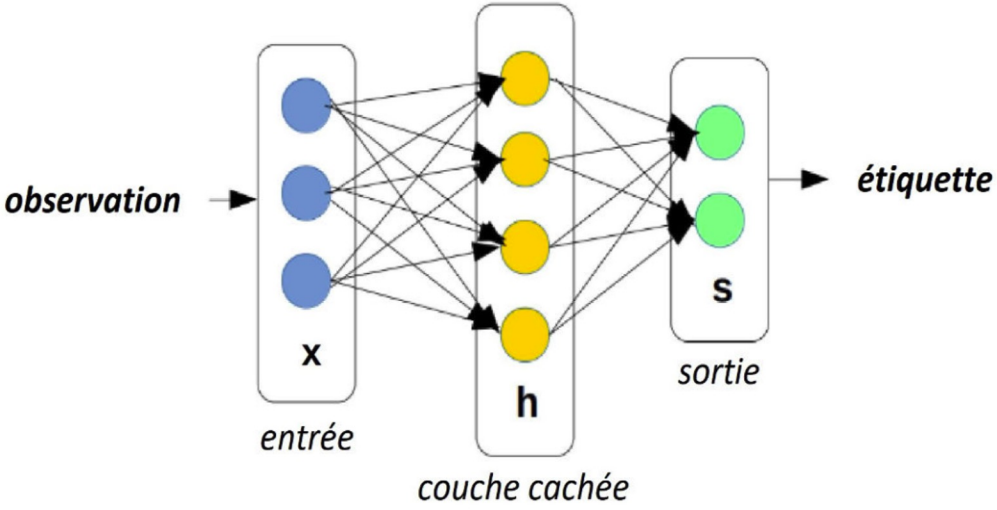
Méthode des *k* plus proches voisins



- Un objet d'entrée est classifié selon la majorité de ses ***k*** plus proches voisins
- La taille de ***k*** peut changer les résultats

Réseaux de neurones : présentation

- Transformation d'une observation en un vecteur numérique
- L'information est véhiculée à travers le réseau par des connexions pondérées
- La couche de sortie donne le score (probabilité) d'appartenance de l'entrée pour chacune des catégories possibles
- Les couches cachées retiennent l'information
- Les paramètres sont optimisés sur les exemples d'apprentissage (époques)
- Attention au sur-apprentissage



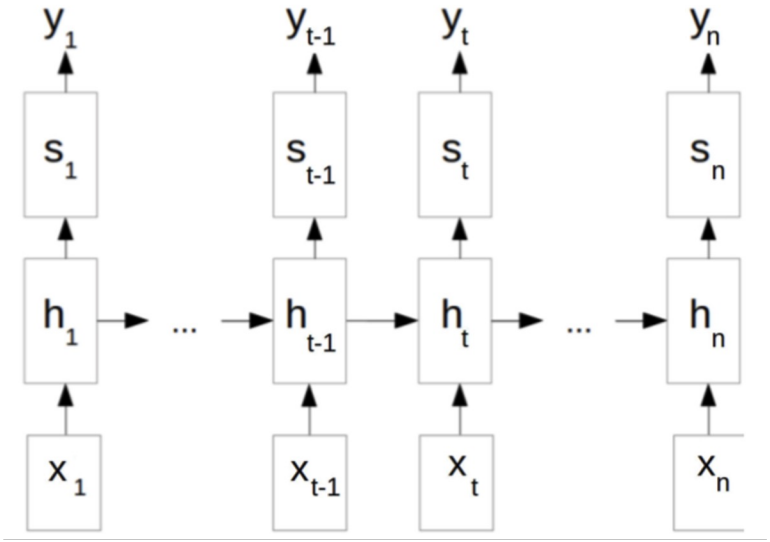
Déterminer ce que le système produit

- Le système apprend à faire lui-même les associations qu'on lui a montré
- Image vers classe (reconnaissance)
- Mot vers mot (traduction)
- Mot/Phrase vers étiquette/concept (compréhension/intention)

Mots	*je*	*veux*	*un*	*billet*	*pour*	*Paris*	*jeudi*	*3*	*octobre*
Étiquettes	O	O	O	O	O	**ville-arrivée**	**date**	**date**	**date**
Concepts	O	O	O	O	O	**ville-arrivée**	**date**		

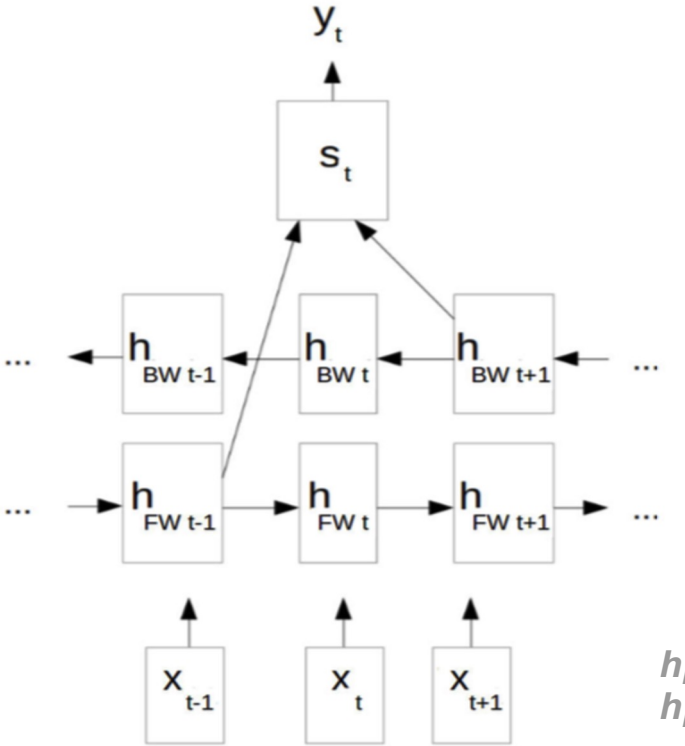
Réseaux de neurones : architectures

- Précédente architecture la plus simple : *feed forward*
- Autres architectures possibles : récurrence avant/arrière *(forward/backward)*



x_t = couche d'entrée (étape t)
h_t = couche cachée (étape t)
s_t = couche de sortie (étape t)
y_t = étiquette de sortie (étape t)

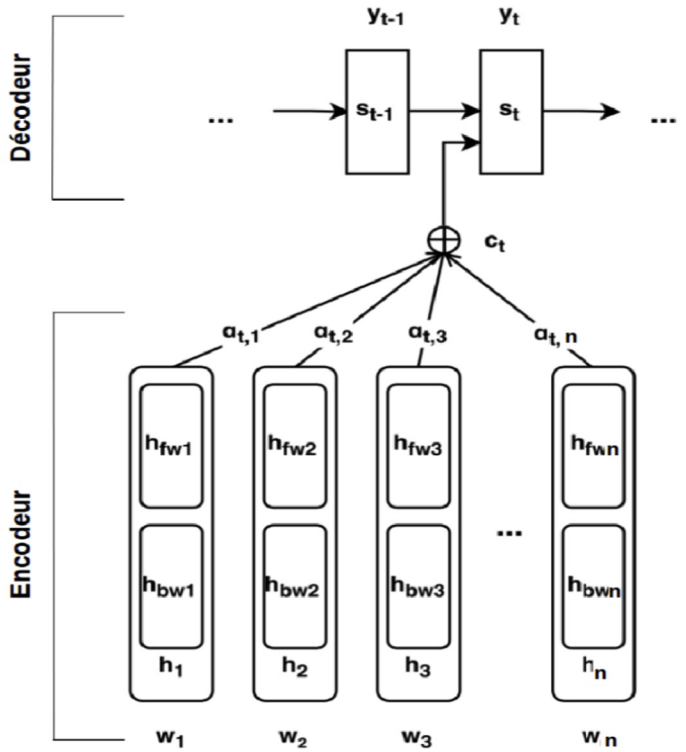
Réseaux de neurones : architectures



- Autres architectures possibles : bidirectionnel

$h_{FW\,t}$ = couche cachée récurrente avant (*forward,* étape *t*)
$h_{BW\,t}$ = couche cachée récurrente arrière (*backward,* étape *t*)

Réseaux de neurones : architectures



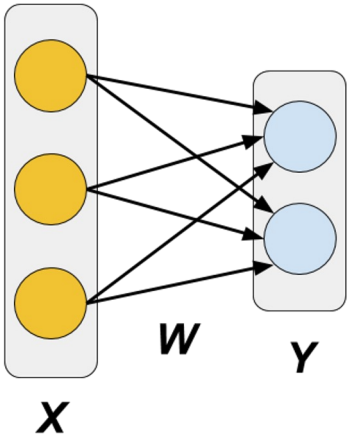
- Autres architectures possibles : encodeur-décodeur avec mécanisme d'attention

$a_{t,j}$ = poids de l'élément *j* pour le traitement de l'élément à l'étape *t*
C_t = vecteur de contexte encodé

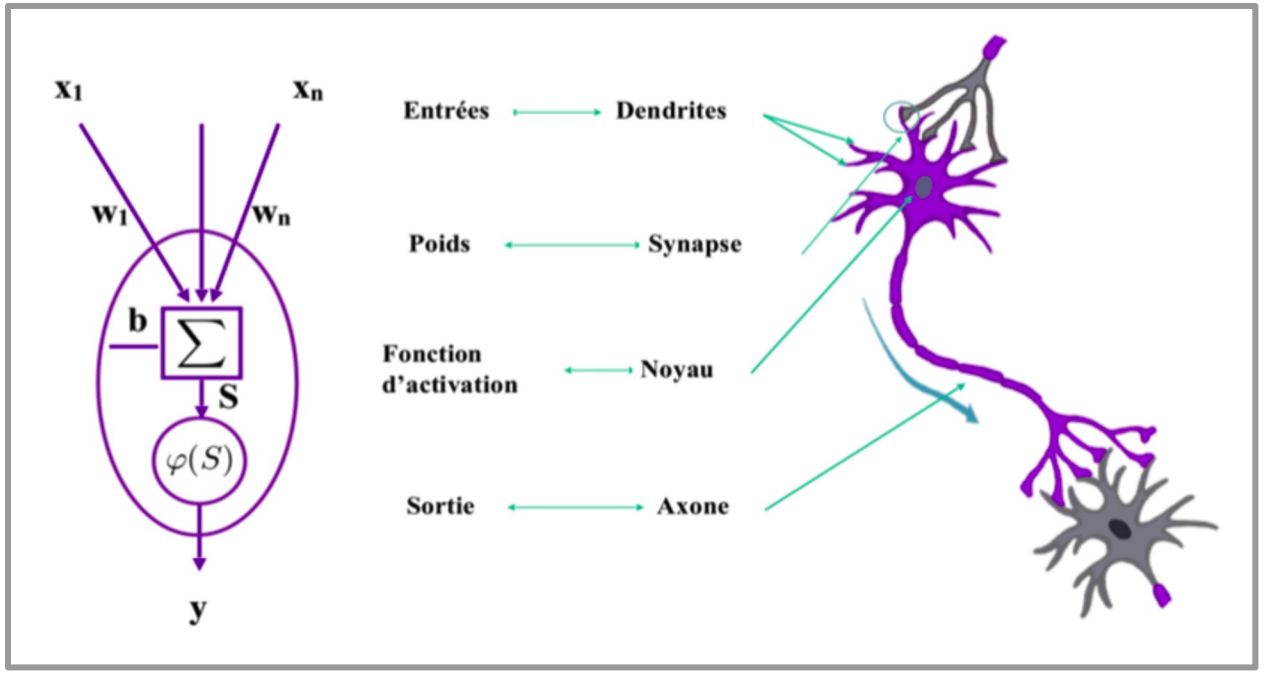
Réseaux de neurones : fonctionnement d'une couche

$$Y = F(W.X + B)$$

- ***X*** : La couche d'entrée
- ***Y*** : La couche de sortie résultante
- ***W*** : La matrice des poids reliant les neurones de ***X*** à ceux de ***Y***
  - les poids sont relatifs à l'importance de l'information véhiculée
- ***B*** : Un biais constituant le poids d'une entrée constante
  - permet d'ajouter de la flexibilité au réseau
- ***F*** : La fonction d'activation appliquée sur l'entrée



Réseaux de neurones : parallèle avec un vrai neurone

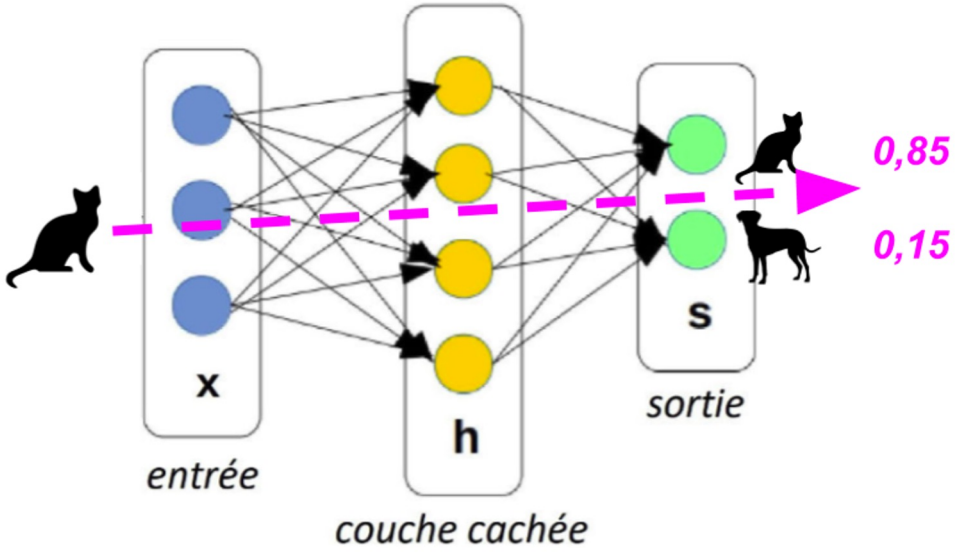


Réseaux de neurones : apprentissage & optimisation

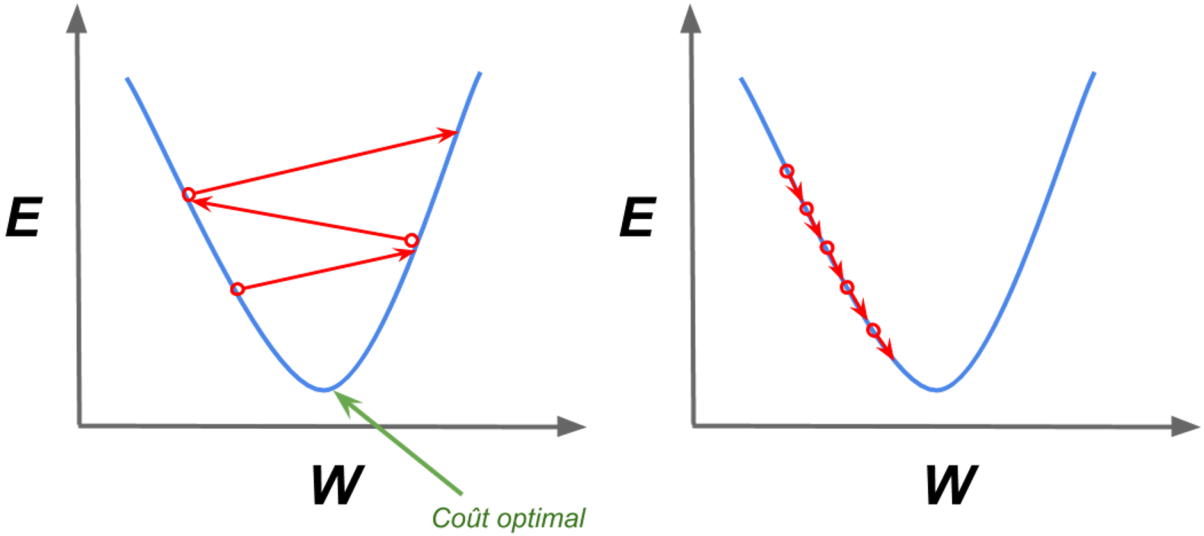
- Erreur (ou coût / *loss*) entre la sortie obtenue et la sortie de référence
  - exemple : ***E = 1 - 0,85***
- *Rétropropagation de l'erreur dans le réseau pour adapter les poids (descente de gradient)*

$$W' = W - \lambda \Delta W \ , \ \lambda > 0$$

$$\Delta W = \frac{\partial E}{\partial W}$$



Réseaux de neurones : apprentissage & optimisation



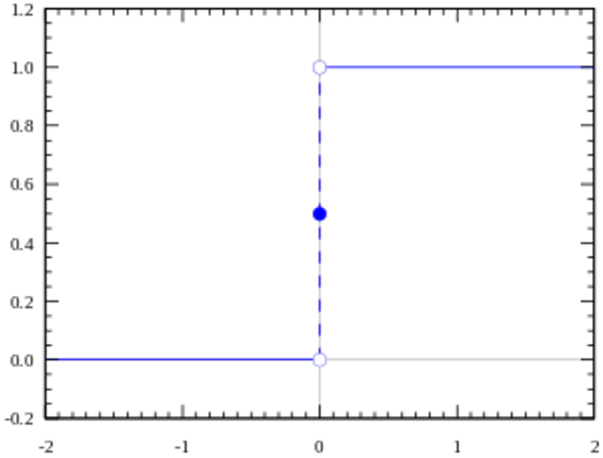
- Taux d'apprentissage :
  - le but est de converger vers un coût minimum
  - trop grand : risque de rater le coût optimal voire de diverger
  - trop petit : on est sûr de converger mais on y arrive très lentement

Réseaux de neurones : apprentissage & optimisation

- Plusieurs fonctions de coût/erreur possibles :
  - erreur moyenne quadratique
  - entropie croisée
  - ...
- Choix d'un algorithme d'optimisation :
  - *Adadelta*
  - *Adam*
  - ...
- Définition d'une métrique représentative de la tâche :
  - précision (***= prédictions correctes / total des prédictions***)
  - ...

Réseau de neurones : retour sur la fonction d'activation

- Supposons un réseau à 1 classe (sortie = Vrai/Faux)
- La valeur d'un neurone appartient à ***[-inf ; + inf]***
- Problème : Comment décider si un neurone est activé ou non ?
- Solution : définir une fonction seuil :



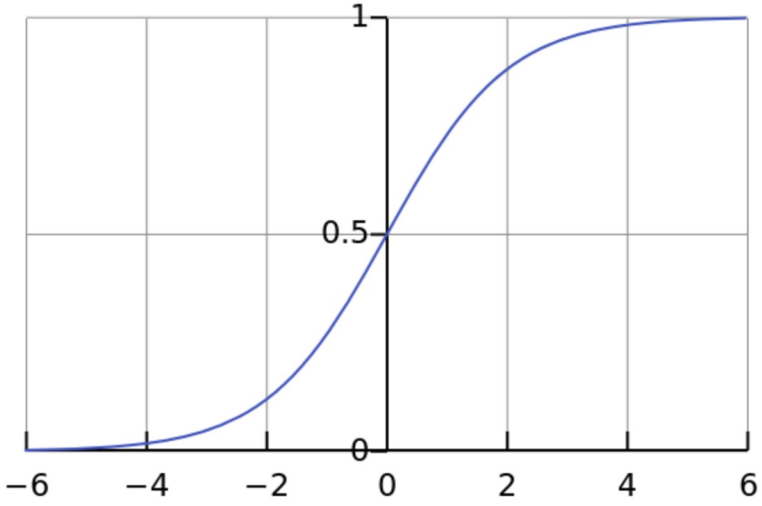
Réseau de neurones : retour sur la fonction d'activation

- Problème : comment faire si il y a plusieurs classes (et pas juste Vrai/Faux) ?
  - comment faire la distinction si plusieurs neurones sont activés (*1*) ? Quelle classe choisir ?
  - on préférerait dans ce cas avoir des valeurs intermédiaires comparables (*0, 0.2, 0.5, 1*) : on choisit le plus gros score
- Solution : choisir une fonction linéaire ***A = cx***
  - ainsi on a bien des valeurs proportionnelles à l'entrée et comparables entre elles
- Problème : la rétropropagation de l'erreur se base sur une dérivée
  - la dérivée de ***A*** est une constante
  - la rétropropagation ne tient plus compte de l'entrée ***x***
- Solution : utiliser des fonctions d'activation dérivables

Réseau de neurones : retour sur la fonction d'activation

- *Sigmoid :*

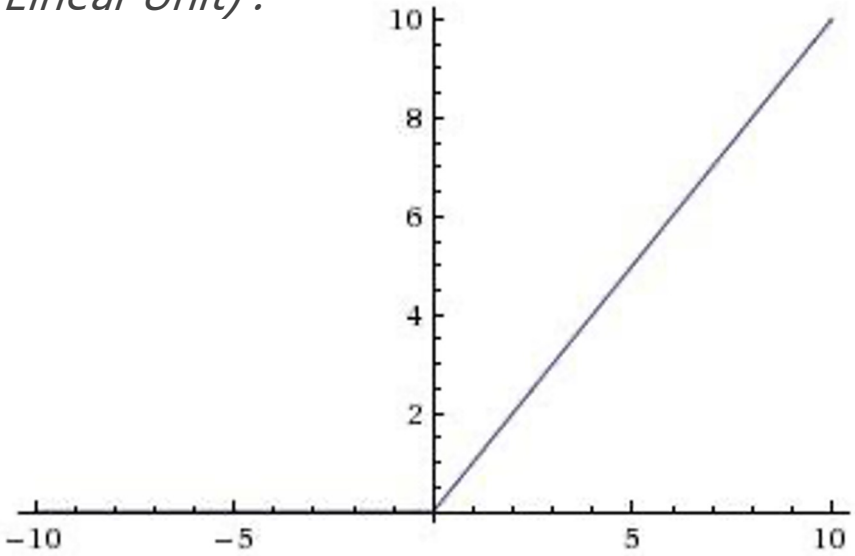
$$A = \frac{1}{1+e^{-x}}$$



Réseau de neurones : retour sur la fonction d'activation

- *ReLu (Rectified Linear Unit) :*

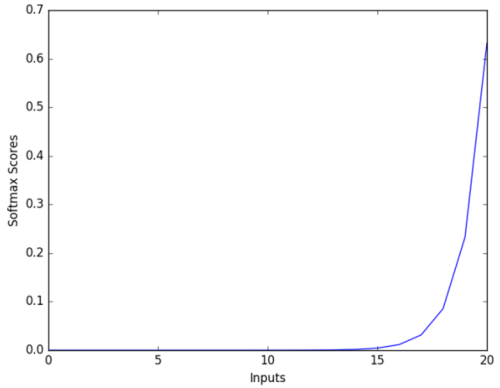
A = max(0,x)



Réseau de neurones : retour sur la fonction d'activation

- *Softmax :*
  - fonction exponentielle normalisée
  - transforme un vecteur en une série de proportions dont la somme vaut 1
  - représente une loi catégorielle (loi de probabilité) sur *K* différents résultats possibles

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ pour tout } j \in \{1, \dots, K\}$$



- Exemple :

$$\mathbf{z} = (1\ ;\ 3\ ;\ 2,5\ ;\ 5\ ;\ 4\ ;\ 2)$$

$$\sigma(\mathbf{z}) = (0,011\ ;\ 0,082\ ;\ 0,050\ ;\ 0,605\ ;\ 0,222\ ;\ 0,030)$$

TensorFlow

1. Présentation
2. Installation

TensorFlow : présentation

- Librairie *Python* pour l'entraînement de réseaux de neurones
- Outils open source développés par *Google*
- De très nombreuses architectures et fonctions prédéfinies
- Une grande communauté d'utilisateurs comparé à d'autres librairies (*theano, torch*...)
- De meilleures performances en temps de calcul
- Fonctionne sur *CPU* ou *GPU* (voire même *TPU*)
    - L'exemple de ce TP étant simple, on utilisera simplement le *CPU*

TensorFlow : installation

- Plusieurs installations possibles : voir sur le site
- Une façon simple (Windows/Mac/Linux) :
  - *Python* et *pip* déjà installé
  - pip install --upgrade pip
  - pip install matplotlib
  - pip install numpy
  - pip install tensorflow
  - (--user pour conflits)

Travaux pratiques

1. Mise en place d'un réseau simple *feed forward*
2. Tâche de reconnaissance d'images de vêtements

TP : import des librairies

- Créer un fichier *python* (*.py*)
- L'appeler avec la commande *python* dans le terminal :

```
python TP.py
```

- Dans le fichier, import des librairies auxiliaires :

```
from __future__ import absolute_import, division, print_function, unicode_literals
import os
import numpy as np
import matplotlib.pyplot as plt
```

TP : import des librairies

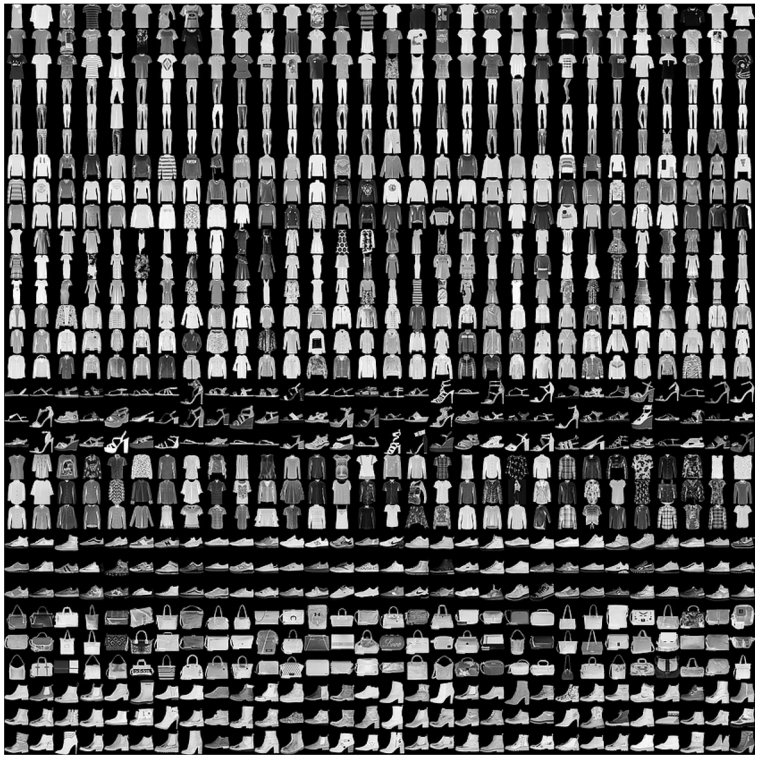
- Import *TensorFlow :*

```
import tensorflow as tf
from tensorflow import keras
```

- Affichage de la version*:*

```
print(tf.__version__)
```

TP : importer le *Fashion MNIST Dataset*



- Contient 70 000 images de vêtements en noir et blancs (*Zalando*)
- Les images sont annotées en 10 catégories :
  - *T-shirt, Pantalon, Pull, Robe, Manteau, Sandales, Chemise, Baskets, Sac, Bottines*
- Les images montrent des articles en résolution *28x28 pixels*

TP : importer le *Fashion MNIST Dataset*

- Importer les données :

```
fashion_mnist = keras.datasets.fashion_mnist.load_data()
```

- Le corpus est pré-découpé en 2 sous-ensembles :
  - 60 000 images pour l'apprentissage
  - 10 000 images pour le test
- Chaque sous-ensemble contient une paire de listes :
  - la liste des images, soit des tableaux de dimension (28,28) contenant des entiers compris dans l'intervalle [0,255] (intensité du noir)
  - la liste des étiquettes correspondantes sous forme d'indices entiers dans l'intervalle [0,9] selon l'ordre énuméré précédemment

```
(train_images, train_labels), (test_images, test_labels) = fashion_mnist
```

- Créer une liste des étiquettes correspondant aux indices :

```
class_names = ['T-shirt', 'Pantalon', 'Pull', 'Robe', 'Manteau','Sandales', 'Chemise', 'Baskets', 'Sac', 'Bottines']
```

TP : regarder les données

- Afficher les éléments suivants :

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

TP : regarder les données

- Afficher une image :

```
image_to_show = train_images[0]

plt.figure()
plt.imshow(image_to_show, cmap=plt.cm.binary)
plt.colorbar()
plt.grid(False)
plt.show()
```

TP : regarder les données

- Afficher plusieurs images avec les étiquettes associées :

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

TP : configurer le modèle

- Transformer les informations d'entrée dans un format mieux reconnu par le réseau de neurones :
  - Mettre les valeurs des pixels de l'intervalle [0,255] sous forme *float* dans un intervalle [0,1]

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

TP : configurer le modèle

- Construire le réseau *feed forward* :
  - 1ère couche est la couche d'entrée : elle a pour rôle de transformer un tableau de 28x28 en un vecteur de 784
  - 2ème couche est la couche cachée : c'est une couche connectée de 128 neurones utilisant une fonction d'activation *reLu*
  - 3ème couche est la couche de sortie : c'est une couche connectée de 10 neurones correspondant aux 10 classes à associer et utilisant une fonction *softmax* afin d'obtenir des probabilités d'appartenance pour chaque classe

```
model = keras.Sequential([
    keras.Input(shape=(28,28)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

TP : configurer le modèle

- Régler les dernières options :
  - fonction de coût
  - fonction d'optimisation
  - métrique externe

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

TP : entraîner le système

- Appeler le modèle :
  - on lui demande d'apprendre à associer un ensemble d'entrée (les images) à un ensemble de sortie (les étiquettes)
  - on règle le nombre d'époques d'apprentissage
  - on voit le coût diminuer tandis que la précision s'améliore
  - on doit normalement obtenir une précision autour de *88%* sur l'ensemble d'apprentissage

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

TP : évaluer le système

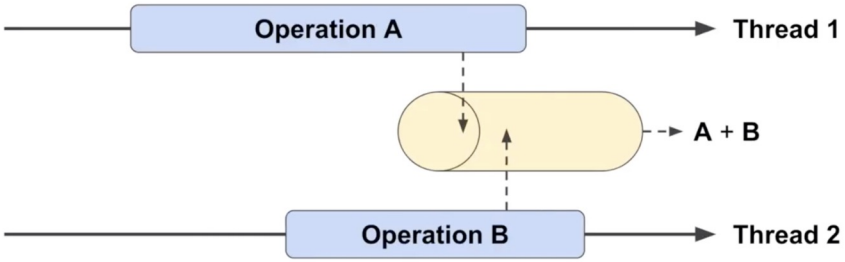
- Demander l'évaluation et afficher les résultats sur l'ensemble de test :
  - on doit normalement obtenir une précision autour de *86%*
  - c'est un exemple de sur-apprentissage, le système est plus performant sur les données sur lesquelles il a appris que sur de nouvelles données
  - on pourrait essayer un apprentissage avec un nombre d'époques plus important ce qui nous donnerait peut être plus que *88%* sur l'apprentissage mais les améliorations sur le test ne seraient pas proportionnelles

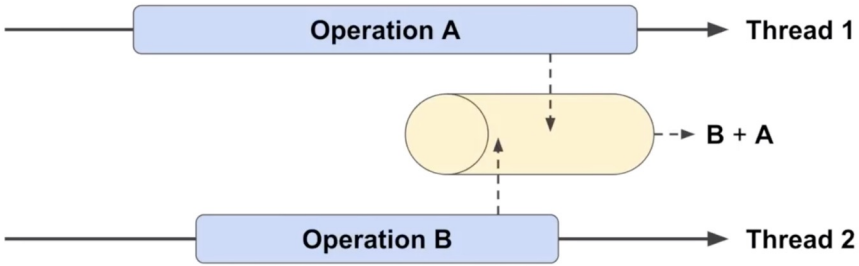
```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

Variabilité dans les résultats

- Si on entraîne une même configuration deux fois de suite, on peut obtenir des résultats différents
  - Pas lié à TensorFlow
- Minimiser le temps d'apprentissage : exécution en parallèle les opérations mathématiques quand cela est possible
- Supposons une opération C dépendant de deux opérations A et B exécutées en parallèle :
  - Si A termine avant B on a C = A + B
  - Si B termine avant A on a C = B + A





Variabilité dans les résultats

- Ces variations dans l'ordre des opérations ne devraient mathématiquement pas avoir d'incidence sur le résultat
- Cependant il y en a une car nous utilisons des valeurs float à précision limitée
    - Les arrondis accumulés sont source de variation

```
2. * 5. / 7.
```

```
1.4285714285714286
```

```
2. / 7. * 5.
```

```
1.4285714285714284
```

TP : sauvegarder notre modèle

- Il peut être pratique de sauvegarder le modèle appris pour l'exécuter sur le test sans avoir à réapprendre à chaque lancement du programme
- Ajouter avant la création du modèle :

```
save_dir = "saved_model"
os.makedirs(save_dir, exist_ok=True)
checkpoint_path = save_dir + "/cp.ckpt.keras"
cp_callback = keras.callbacks.ModelCheckpoint(checkpoint_path)
```

- Dans la fonction d'apprentissage du modèle, ajouter l'option suivante après celle des époques :

```
# model.fit(... , callbacks = [cp_callback])
```

- Après avoir appris et sauvegarder un modèle, il n'est plus nécessaire d'appeler la fonction d'apprentissage, mais à la place, celle de chargement du modèle pré-appris (beaucoup plus rapide) :
  - essayer avec/sans le chargement du modèle pour voir les résultats (médiocres) obtenus par un système n'ayant rien appris

```
model.load_weights(checkpoint_path)
```

TP : observer les prédictions

- Récupérer les prédictions de l'ensemble de test

```
predictions = model.predict(test_images)
```

- Afficher les éléments suivants :
  - que représentent-ils ?

```
predictions[0]
np.argmax(predictions[0])
test_labels[0]
```

TP : observer les prédictions

- Créer une fonction affichant une image et sa prédiction :
  - affichage bleu/rouge si correct/incorrect
  - ne pas oublier d'appeler ***plt.show()*** pour afficher le résultat après l'appel de la fonction

```
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)
```

TP : observer les prédictions

- Créer une fonction affichant un histogramme des différents scores pour chaque classe pour une image donnée :

```
def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array[i], true_label[i]
  plt.grid(False)
  plt.xticks(range(10), [str.lower(cn[:4])+'.' for cn in class_names], rotation=45)
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

TP : observer les prédictions

- Faire des tests pour différentes images :

```
i = 0

plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions,  test_labels)

plt.show()
```

TP : observer les prédictions

- Afficher plusieurs images et leur résultat à la fois :

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions, test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions, test_labels)
plt.show()
```

Bonus : Utiliser le système librement

- Ne pas se limiter au corpus de TEST
- Utiliser mes images

Métriques d’évaluation

1. précision
2. rappel
3. f-mesure

Métriques : précision, rappel, f-mesure

- J'ai ***10*** *éléments* à classer parmi trois classes ***A, B et C***
  - certains éléments n'ont pas d'étiquettes
- Voici la référence :

E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
A	.	B	B	B	.	C	C	C	.

- Voici l'hypothèse produite par le système :

E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
.	.	.	B	B	B	.	C	.	.

Métriques : précision, rappel, f-mesure

- La précision est :

$$\frac{\textit{l'ensemble des éléments correctement détectés}}{\textit{l'ensemble des éléments détectés}}$$

éléments	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
référence	A	.	B	B	B	.	C	C	C	.
hypothèse	.	.	.	B	B	B	.	C	.	.

- P_A = 0 % (rien trouvé)
- P_B = 2/3 = 67 %
- P_C = 1/1 = 100 %
- $P_{général}$ = 3/4 = 75 %

Métriques : précision, rappel, f-mesure

- Précision :
  - autrement dit c'est se demander : sur le nombre de fois ou je tire, combien de fois j'ai touché ?
  - inconvénient : si je ne classe qu'un seul élément et que j'ai juste alors j'ai une précision de 100% mais il y avait peut-être encore beaucoup d'éléments à classer
  - une autre métrique est nécessaire

Métriques : précision, rappel, f-mesure

- Le rappel est :

$$\frac{\textit{l'ensemble des éléments correctement détectés}}{\textit{l'ensemble des éléments qu'il faut détecter}}$$

éléments	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
référence	A	.	B	B	B	.	C	C	C	.
hypothèse	.	.	.	B	B	B	.	C	.	.

- $R_A = 0/1 = 0\ \%$
- $R_B = 2/3 = 67\ \%$
- $R_C = 1/3 = 33\ \%$
- $R_{général} = 3/7 = 43\ \%$

Métriques : précision, rappel, f-mesure

- Rappel :
  - autrement dit c'est se demander : sur le nombre de cibles présentes, combien en ai-je touché ?
  - inconvénient : pour une classe donnée, si je classe tout, alors j'ai un rappel de 100% alors que je me serais trompé de nombreuses fois
  - précision et rappel sont complémentaires

Métriques : précision, rappel, f-mesure

- La f-mesure est la moyenne harmonique du rappel et de la précision
  - La moyenne harmonique, à la différence de la moyenne arithmétique classique fait la moyenne d'éléments parmi lesquelles existent des relations de proportionnalités inversées

$$\frac{2 \times rappel \times précision}{rappel + précision}$$

- $F_A = 0\ \%$
- $F_B = 67\ \%$
- $F_C = 50\ \%$
- $F_{général} = 54\ \%$

Métriques : précision, rappel, f-mesure

- Concernant le TP :
  - sur l'ensemble des étiquettes, la problématique de la précision et du rappel ne se posait pas : tous les éléments ayant une étiquette attribuée, le rappel et la précision sont égaux
  - en revanche si on avait regardé classe par classe, les précisions et rappels respectifs n'auraient pas été identiques (exemple : j'ai trouvé 5 pulls alors qu'il y en avait réellement 7)

Métriques : la notion de frontières dans les étiquettes

- Il peut arriver dans certaines tâches d'étiquetage (comme l'étiquetage de mots) qu'une étiquette couvre plusieurs éléments
  - on appelle cela un concept
- Dans ce cas on peut faire une évaluation élément par élément comme précédemment, ou bien on peut vouloir avoir une métrique tenant compte de ces frontières :
  - dans ce cas un concept n'est correct que s'il commence et termine par les bons éléments
  - on peut utiliser le formalisme ***BIO*** qui consiste à mettre un préfixe ***B*** à une étiquette débutant le concept, ***I*** dans le concept et ***O*** hors de tout concept
  - en dehors de cela, les calculs de précision, rappel et f-mesure sont similaires

Métriques : la notion de frontières dans les étiquettes

- Exemple (avec une seule classe ***c***) :

référence	***O***	***B-c***	***B-c***	***I-c***	***B-c***	***O***	***B-c***	***O***
hypothèse	***B-c***	***O***	***B-c***	***I-c***	***B-c***	***I-c***	***I-c***	***I-c***

- Si je ne regarde que les étiquettes alors :
  - il y a 5 étiquettes à trouver dans la référence
  - on a étiqueté 7 éléments, 4 sont corrects
  - j'ai donc une précision de 4/7 = 57 %, un rappel de 4/5 = 80 % et une f-mesure de 67 %
- Si je regarde les concepts alors :
  - il y a 4 concepts à trouver dans la référence
  - on a étiqueté 3 concepts, 1 est correct
  - j'ai donc une précision de 1/3 = 33 %, un rappel de 1/4 = 25 % et une f-mesure de 29 %

Métriques : la notion de frontières dans les étiquettes

- Évaluer en concept est plus exigeant qu'en étiquette d'où une baisse plus ou moins significative des performances
- Ajouter les préfixes ***B*** et ***I*** sur les étiquettes lors de l'apprentissage est une information supplémentaire qui peut aider un système d'apprentissage automatique à mieux détecter les frontières des concepts

Merci !