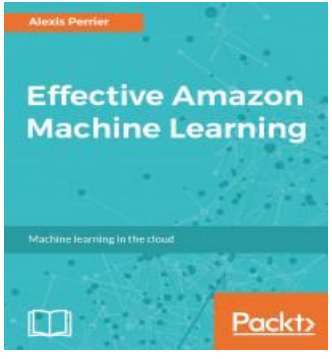
Deep learning par la pratique

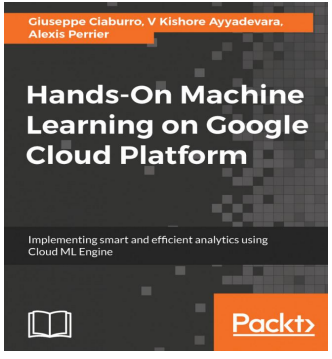
Mardi 11 Juin - matin
Le Perceptron

tour de table

Alexis Perrier
data scientist, consultant, formateur

- https://www.linkedin.com/in/alexisperrier/
- alexis.perrier@gmail.com





tour de table

Auto Evaluation émargement

https://evalcfd.orsys.fr/

Supports de cours

Le github https://github.com/SkatAI/deeplearning

- slides
- notebooks
- code
- documents pdf

Google Colab : https://colab.research.google.com/

- ou notebooks en local
- mais de préférence avec GPU

Le programme

Les réseaux de neurones artificiels facilitent l'apprentissage automatique et bouleversent de nombreux secteurs économiques. Durant cette formation vous utilisez les outils les plus répandus du domaine afin de réaliser et entrainer différents types de réseaux de neurones profonds sur des jeux de données diversifiés.

OBJECTIFS PÉDAGOGIQUES

À l'issue de la formation l'apprenant sera en mesure de :

Comprendre l'évolution des réseaux de neurones et les raisons du succès actuel du Deep Learning

Utiliser les bibliothèques de Deep Learning les plus populaires

Comprendre les principes de conception, les outils de diagnostic et les effets des différents verrous et leviers

Acquérir de l'expérience pratique sur plusieurs problèmes réels

https://github.com/SkatAI/deeplearning/blob/master/docs/DPL.pdf

Deep learning par la pratique

- Mardi 11 juin
  - matin : Rappels; Perceptron; SGD
  - aprés-midi : Perceptron multicouches
- Mercredi 12 juin
  - matin : CNN, Transfer learning, Tensorboard
  - aprés-midi : RNN, LSTM et GRU
- Jeudi 13 juin
  - matin : Autoencoders
  - aprés-midi : NLP, transformers, recap

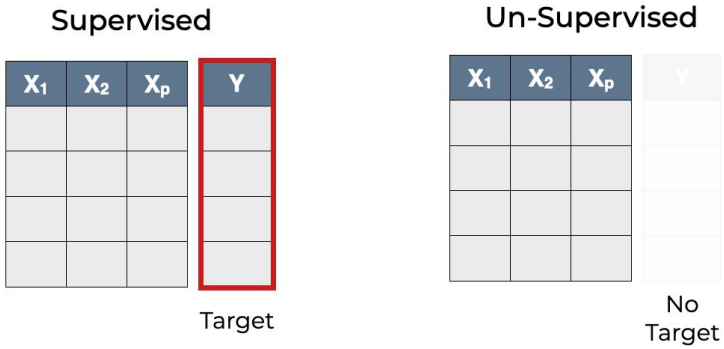
Ce matin :

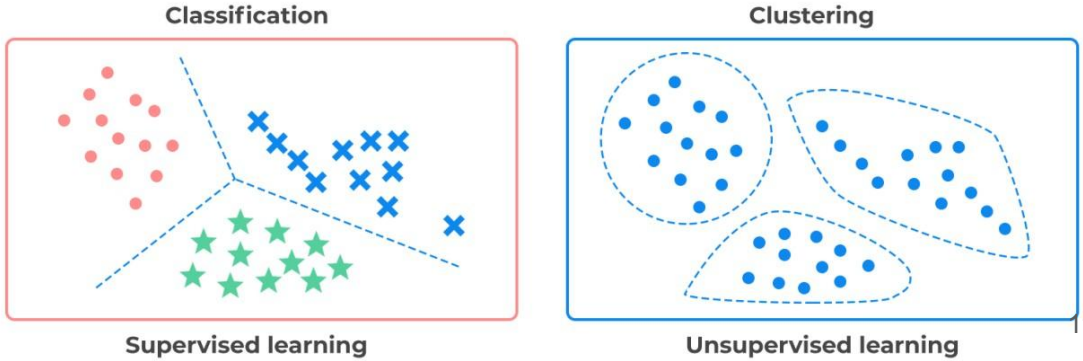
- Rappels :
  - classification / regression,
  - biais / overfitting,
  - métriques, fonction de coût,
  - techniques de régularisation
- Le Perceptron
- Fonctions d'activation, optimiseur
- Descente de Gradient - SGD
- Perceptron Multi Couche

Rappels

supervisé et non supervisé

- Supervisé : existence d'une variable cible qui détient la vérité (ground truth)
- Non supervisé : pas de variable cible





non supervisé

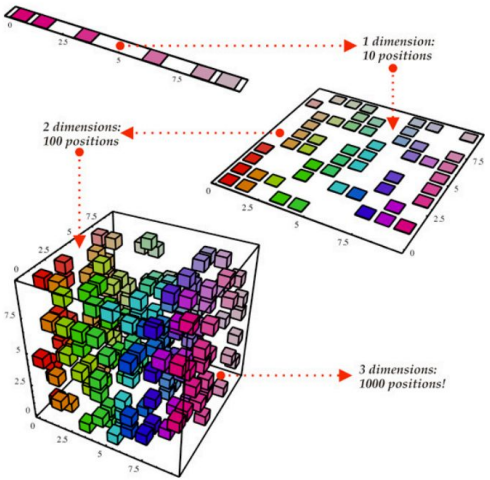
Tâches :

- ACP: reduction de dimension
- Clustering
- topic modeling, boule de crystal

Métriques

- variance capturée, erreur de reconstruction
- Silhouette
- perplexity, coherence

ACP



Clustering

Supervisé

Classification

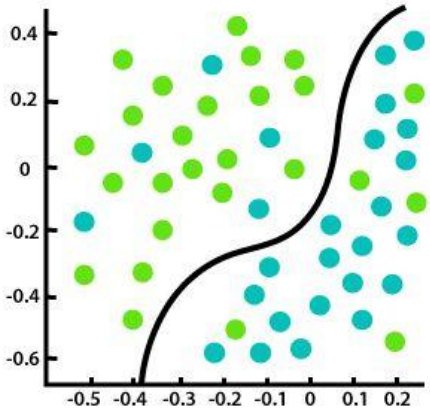
- binaire, multi classe, multi-étiquettes

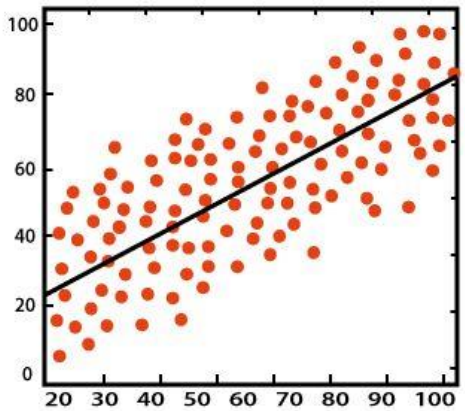
Métriques

- accuracy, recall, precision, AUC, …

Régression - Métriques

- MSE, MAE, RMSE, …

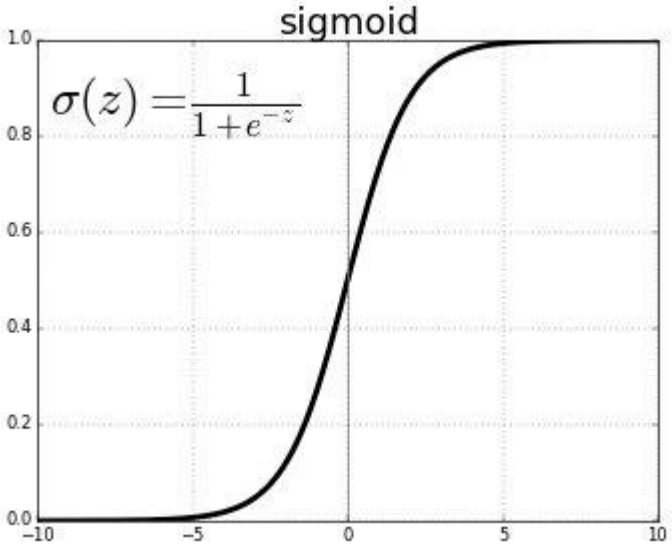


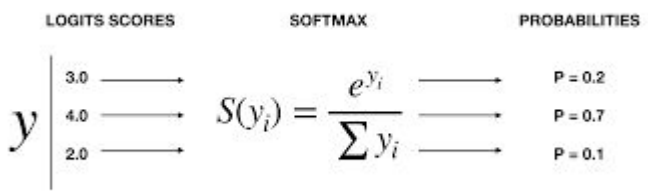


sigmoid : de la régression à la classification

La classification n'est que de la régression passée par une fonction sigmoid (softmax, logit)

- La régression produit des valeurs non bornées
- on interprète les valeurs de sorties de la fonction sigmoid comme la probabilité d'appartenir à une classe
- puis classification en fonction d'un seuil
  - tau = 0.5





Nature des données

type de data et problèmes associés:

- tabular,
- images,
- sons,
- vidéo,
- textes,
- séries temporelles

Et Vous ?
Sur quoi travaillez vous ?

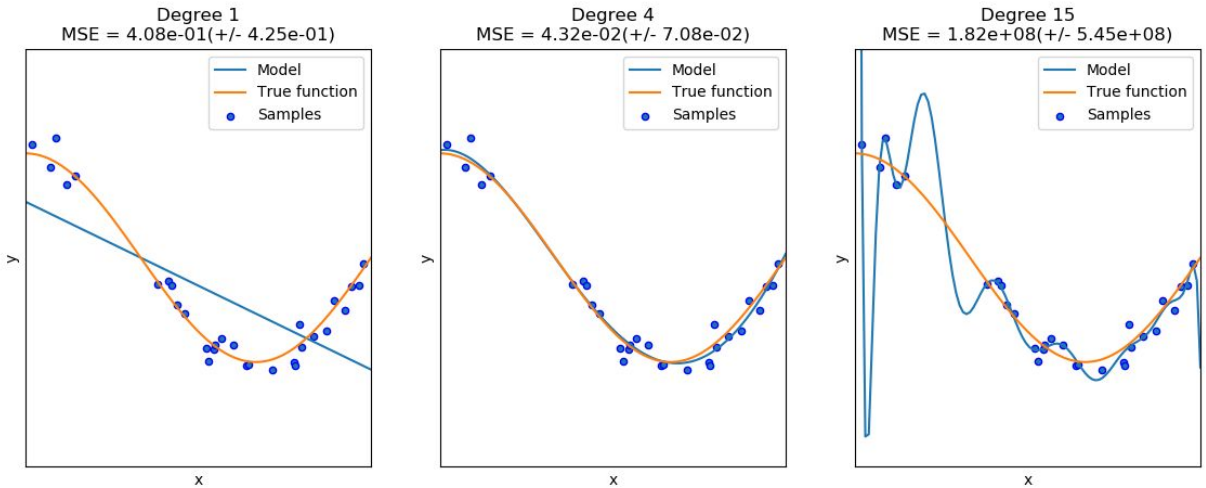
Concepts clés en machine learning

- biais et overfit
- fonction de coût
- régularisation

Evaluation d'un modèle prédictif

Séparer les données en sets d'entraînement, de test et de validation (60, 20%, 20%)

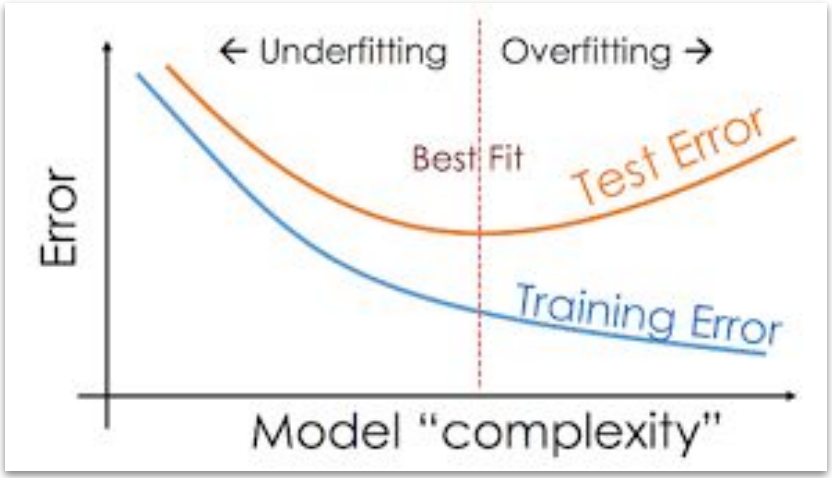
Le modèle doit savoir **extrapoler**
sur des données nouvelles

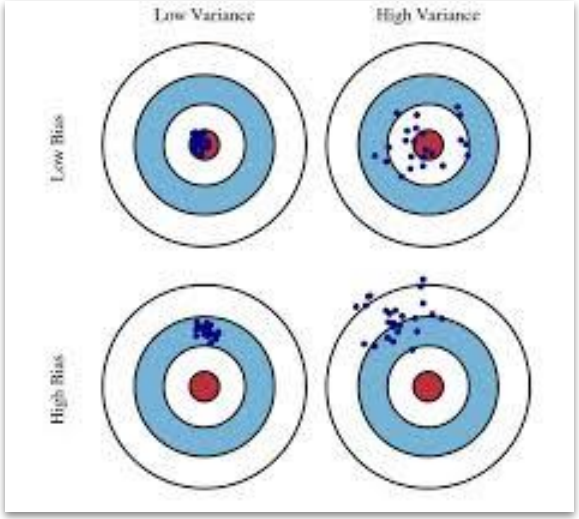


biaisé bien overfitting

Biais et Overfitting

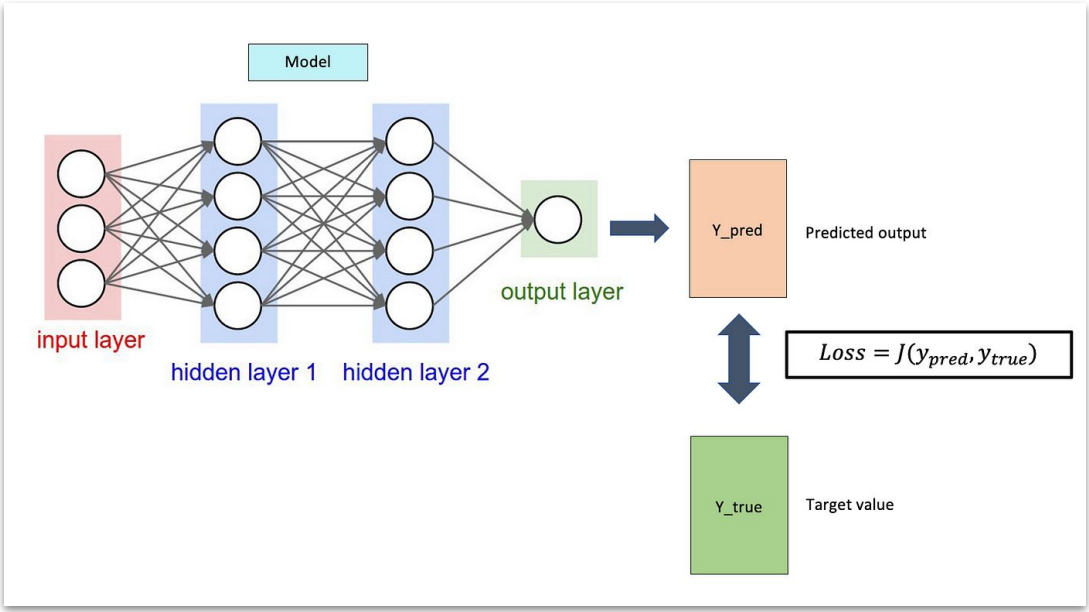
- biais: mauvaise predictions
- overfit: sur ajustement : le modèle ne sait pas extrapoler
- detection de l'overfit
  - learning curve
  - écart entre score sur dataset d'entraînement et de validation



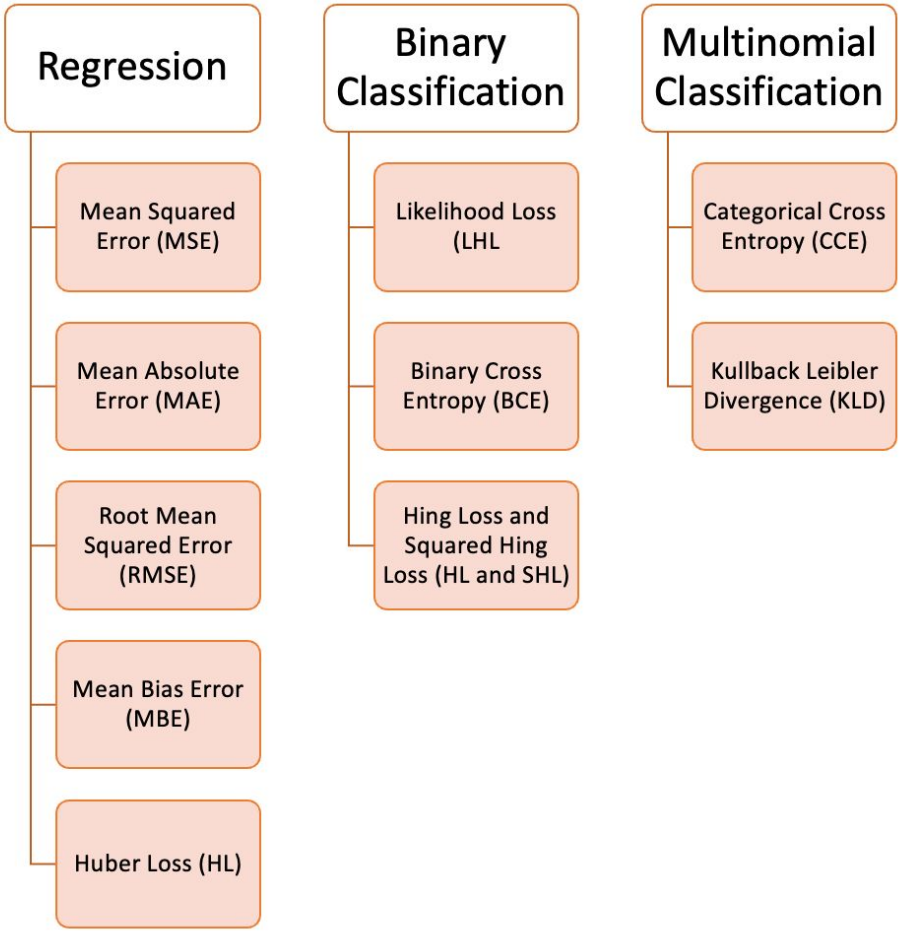


Fonction de coût - Loss function

- Estimation de la distance entre les vraies valeurs et les valeurs prédites par le modèle
- Un bon modèle réduit cette distance au maximum
- La fonction de coût définie l'objectif de l'optimisation du modèle



Quelques fonctions de coût



Quelques fonctions de coût

Task	Error type	Loss function	Note
Regression	Mean-squared error	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Easy to learn but sensitive to outliers (MSE, L2 loss)
	Mean absolute error	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	Robust to outliers but not differentiable (MAE, L1 loss)
Classification	Cross entropy = Log loss	$-\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] =$	Quantify the difference between two probability

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$$

2– Huber Loss

$$l(y_i, \hat{y}_i) = \begin{cases} (y_i - \hat{y}_i)^2 & for\ |y_i - \hat{y}_i| \le \delta \\ 2\delta|y_i - \hat{y}_i| - \delta^2 & otherwise \end{cases}$$

Régularisation : limiter l'overfit sans ajouter de biais

Ajouter une contrainte au modèle pour empêcher le modèle de coller de trop près aux données d'entraînement.

contraintes sur le modèle :

- L1, L2 sur la fonction de coût
- réduire la complexité :
  - moins de couches dans le NN

Arbres de décision:

- limiter la profondeur, échantillons par feuille
- bagging : forêts aléatoires

L1 Regularization

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

L2 Regularization

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

Loss function

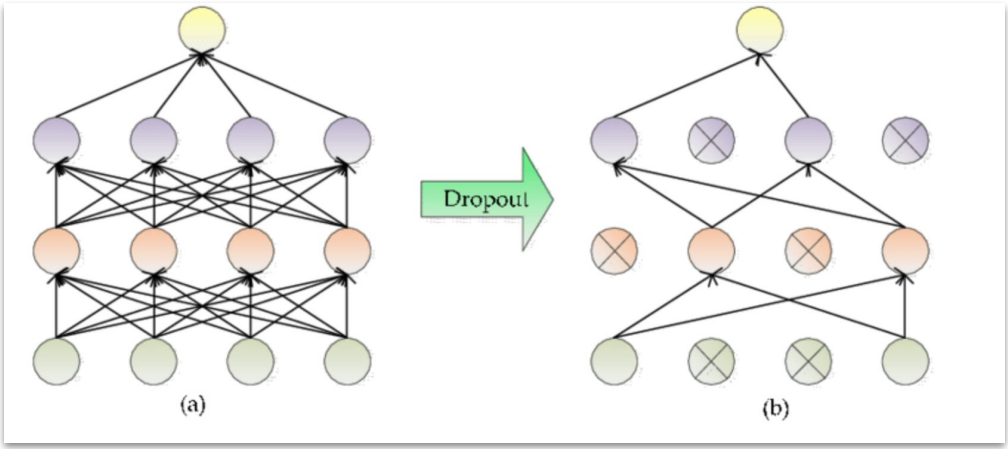
Regularization Term

Régularisation des réseaux de neurones

- réduire complexité : le nombre de couches, de neurones
- ajouter du bruit au données

dropout

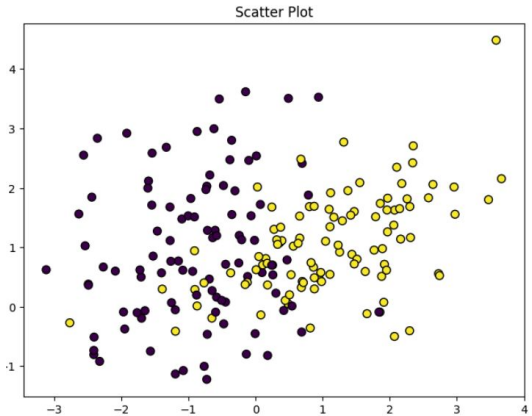
- de façon aléatoire (p = 0.2) on annule un certain nombre de noeuds à chaque itération
- simple, efficace
- équivaut au bagging

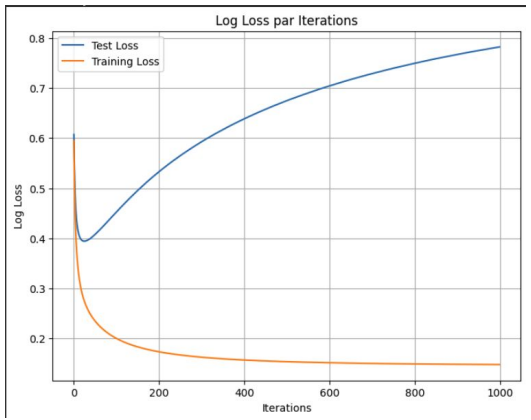


Notebook régularisation et SGD

Chargez le notebook SGD_Overfit_et_regularisation sur Colab

https://github.com/SkatAI/deeplearning/blob/master/notebooks/SGD_Overfit_et_regularisation.ipynb





Perceptron
Rosenblatt 1957

Perceptron - 1957

11 avril 1957, la reine Elizabeth II visite Paris en voiture accompagnée par le président de la République René Coty
©Getty - Express

Quelques dates

1943 : McCulloch et Pitts proposent la notion de neurone artificiel combinant des entrées pour produire une sortie, sans algorithme d'apprentissage pratique.

1957 : Rosenblatt développe le **perceptron**, qui combine linéairement les entrées et les seuils pour prendre une décision binaire, avec un algorithme d'apprentissage des poids à partir des données.

1969 : Minsky et Papert : perceptron multicouche

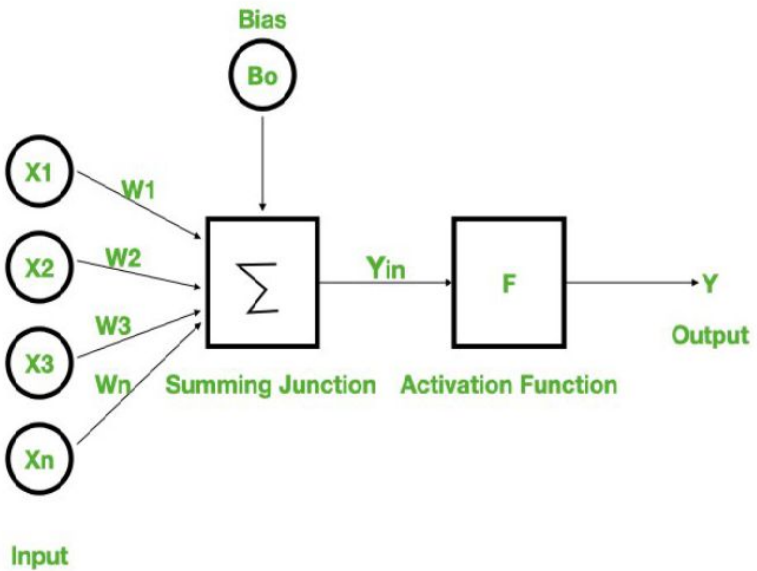
Années 1980 : Développement de la rétropropagation (backpropagation)

https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels

A la base, le neurone

- coefficients - poids - weights
- un biais
- une fonction dite d'activation

Equivalent à une régression linéaire



https://www.geeksforgeeks.org/how-neural-networks-can-be-used-for-data-mining/

Presentation du perceptron

classification binaire de données **séparables linéairement**

- N échantillons
- parametres : coefficients, taux d'apprentissage, seuil de classification

- initialisation: coefs = 0 (N)
- pour chaque échantillon choisi aléatoirement :
  - `valeur = dot_product(échantillon, coefs) + bias`
  - prediction = 1 si `valeur` > **seuil** sinon prediction = 0
  - si prédiction != vraie valeur => maj des coefs:

`coefs = coefs + taux_apprentissage (vrai - prediction) * échantillons`

  - sinon => rien

Régression

Fonction Heavyside

Mise à jour

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P$ *and* $\mathbf{w}.\mathbf{x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N$ *and* $\mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

Perceptron single layer ~= régression logistique

dans les 2 cas: combinaison linéaire des valeurs d'entrées

Peu de différences :

	perceptron single layer	**régression logistique**
fonction activation	heavyside	sigmoid (logit)
entraînement	gradient stochastique	MLE / max de vraisemblance
interprétation	-	Statistique

Limite du perceptron

Un perceptron à une seule couche ne peut pas apprendre des fonctions non linéaires, malgré le fait que la fonction d'activation (fonction de Heaviside) soit non linéaire.

Les limites d'apprentissage du perceptron sont liées à son architecture trop simple.

Le perceptron n'a pas de couches cachées qui sont nécessaires pour capturer des relations non linéaires entre les entrées et la sortie.

Pour apprendre des fonctions non linéaires, des architectures de réseaux de neurones plus avancées sont utilisées. Par exemple les perceptrons multicouches avec des couches cachées associées à des fonctions d'activation non linéaires,.

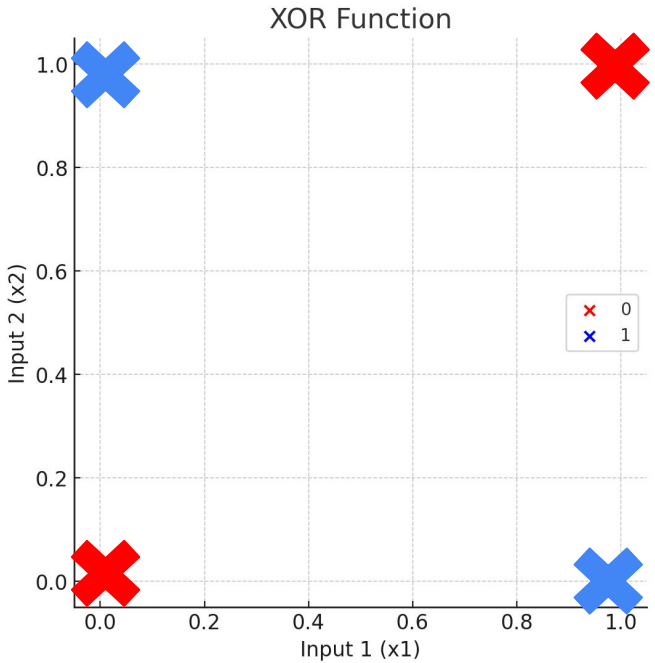
XOR vs OR

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

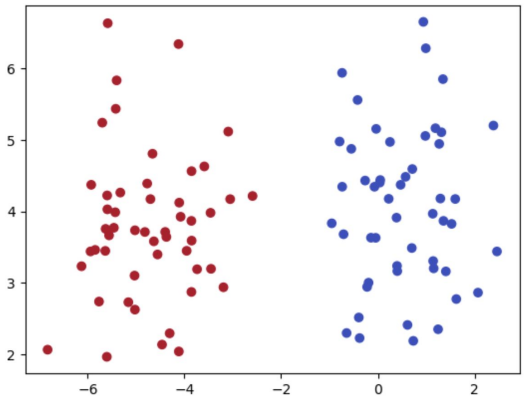
XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0



notebook perceptron

- code du perceptron
  - version basique
  - puis ajouter le learning rate
- Demo OR, XOR
- influence de l'initialisation des poids et du biais
- generer des blobs
  - séparable et non-séparable
- experimenter avec
  - learning rate
  - epochs



https://github.com/SkatAI/deeplearning/blob/master/notebooks/perceptron.ipynb

Perceptron une couche et plusieurs noeuds classification Multi classes

chaque noeud

- a son propre set de coefficients et son biais
- est entraîné pour une des classes

Questions ?

questions

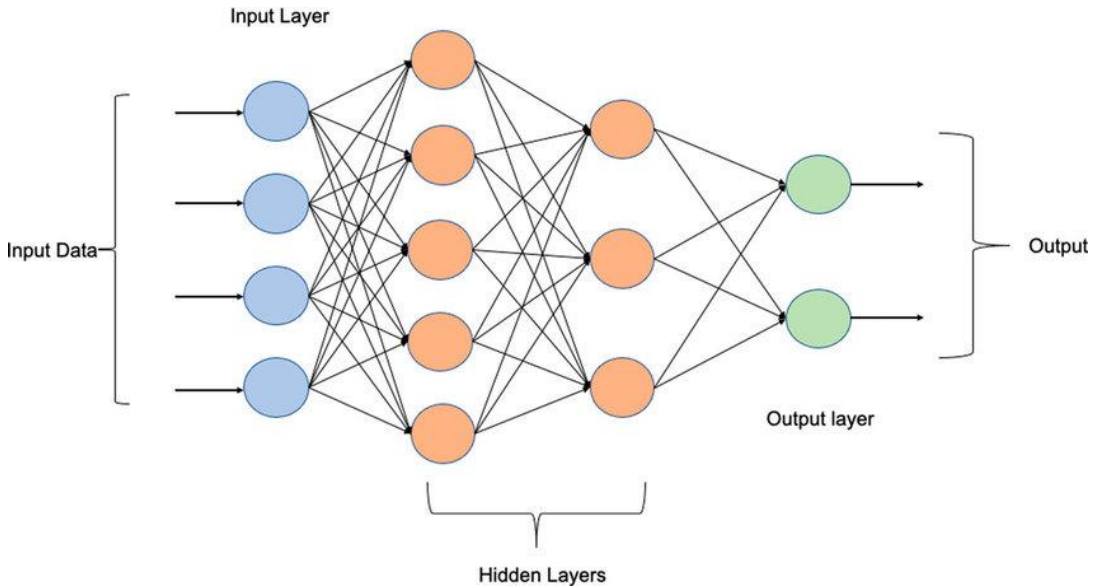
pourquoi introduire un biais ?

- degré de liberté supplémentaire qui permet la translation de la courbe de décision
- si l'échantillon n'a que des valeurs = 0 => tous les outputs = 0
- change le niveau d'activation du perceptron

Perceptron Multi Couche (MLP)

MLP

perceptron multi couche = perceptron + des couches cachées (couches internes)



fonctions d'activation

sigmoid [0,1]

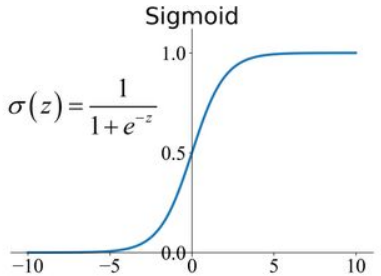
- gradient vanishing, calculs, pas centré en 0 => ralenti la convergence

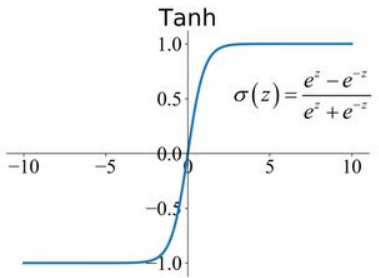
tanh : [0,1]

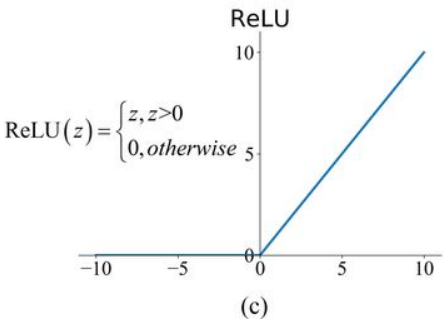
- calcul cher (exp)

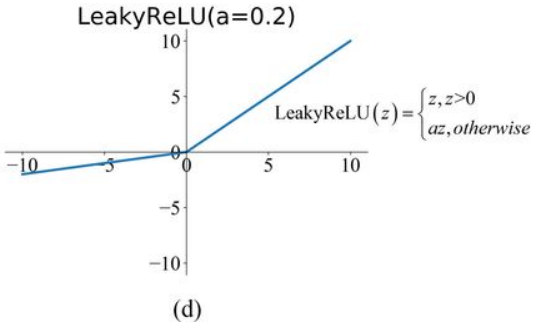
ReLu [0, inf]

- gradient = 0 ou 1
- convergence accélérée
- choix préféré





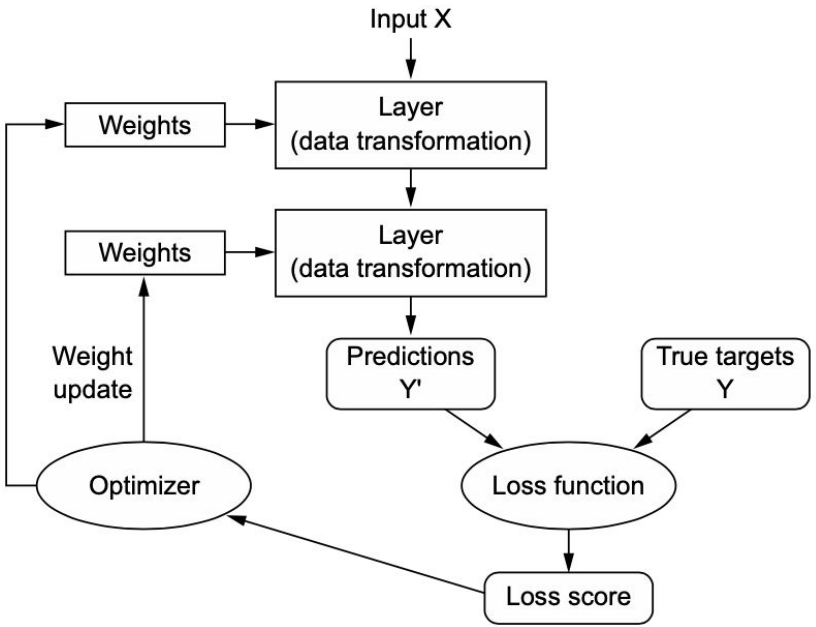




Optimizer - solver

SGD vs ADAM vs RMSprop

L'optimizer implémente l'algo de **backpropagation** qui mets à jour les coefficients du neurone en fonction du gradient de la fonction de coût



MLP

- peut apprendre des fonctions non-linéaires grâce à la fonction d'activation et aux couches cachées

mais

- existence possible de plusieurs minimums locaux => sensible aux valeurs d'initialisation
- il faut optimiser les hyper-paramètres : # couches, # noeuds, # epochs, le learning rate
- sensible à la dynamique des variables => il faut **normaliser**

MLP avec scikit-learn

MLPClassifier

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,),
activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200,
shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False,
momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
n_iter_no_change=10, max_fun=15000)                                    [source]
```

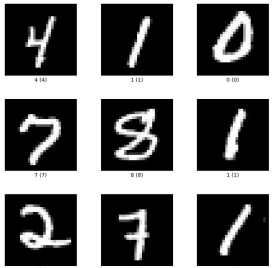
Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

atelier MLP : MNIST avec scikit-learn MLPClassifier

- implémentation avec scikit-learn sur mnist
- MNIST
  - c'est le Iris / Titanic du deep learning
  - images 28x28 - chiffres 0 à 9
  - 60k train, 10k test
  - https://www.tensorflow.org/datasets/catalog/mnist
  - original 1998 http://yann.lecun.com/exdb/mnist/



- tuning des hyper-parametres
  - nombre couches, nombres de noeuds
  - version binaire ou multiclasse vs regression

visualiser les coefs internes

see https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py

Cycle de vie de la valeur d'un noeud

Cycle de vie de la valeur d'un noeud:

- initialisation a des valeurs, (strategie Xavier ou He)
- **forward pass :**
  - du début a la fin (input to output, gauche à droite)
  - pour chaque noeud, calcul de la moyenne pondérée avec les coeffs du noeud => calcul de l'output
  - output values: mise en mémoire pour le pass backward
- calcul de la valeur fonction de coût, de l'erreur en sortie du graphe
- **backward pass**:
  - calcul gradient de la fonction de coût
  - propagation de la fin au début
  - maj des coefficients de chaque noeud avec le gradient descent
- maj des paramètres

Pour inference, les valeurs des noeuds sont fixes et le réseau peut faire des prédictions avec le forrward pass

Récap de la session