SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

SORBONNE UNIVERSITES

PROJET

RESAUX NEURONES

Étudiant :
DIARRASSOUBA SAKARIA

Enseignant :
ANNICK VALIBOUZE

15 avril 2020

Table des matières

Première partie

INTRODUCTION

Le cerveau humain, est le meilleur modèle de machine polyvalente, incroyablement rapide, dotée d'une incomparable capacité d'auto organisation et surtout très douée à réaliser des raisonnements complexes et prenant en compte certaines taches nécessaires pour obtenir un comportement intelligent.

Ces capacités sont duent au fait que le cerveau est constitué de milliards de cellules nerveux appelées **neurones** (environ 10^{15}) reliés entre eux (entre 10^3 et 10^4 connexions par neurones).

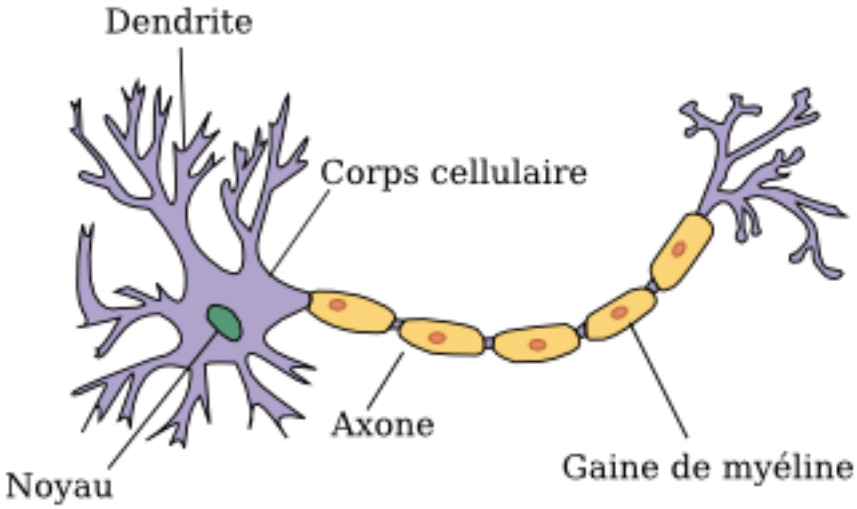


FIGURE 1 – neurone biologique

— Chaque neurone possède une **chevelure de dendrites** qui sont les récepteurs principaux du neurone pour capter les signaux qui lui parviennent

— **L'axone** qui est la fibre nerveuse sert de moyen de transport des signaux emis par le neurone. Il se distingue des dendrites par sa forme et ses proriétés.

— La connexion de l' axone d'un neurone aux dendrites ou au corps cellulaire d'un autre neurone est appelée **des synapses**. En d'autre terme elles permettent la transmission de l'information d'un neurone à un autre.

En 1943, Mc Culloh, un neuro-physiologiste, et Pitts, un logicien, ont proposé pour la première fois un modèle de neurone formel de la cellule nerveuse qui se résumait en une formule simple. L'objectif de ces derniers étaient d' imiter les neurones biologiques et aussi être capables de mémoriser des fonctions booléennes simples.

Les reseaux de neurones artificiels concus à partir de ce type de neurones furent inspirés du systeme nerveux. Ces réseaux de neurones artificiels ont été développés avec pour objectifs principaux d'une part la modélisation et compréhension du fonctionnement du cerveau et d'autre part pour réaliser des architectures ou des algorithmes d'intelligence artificielle.

C'est en 1958 que le premier plus simple réseau neurone artificiel appelé **Perceptron** fut proposé par Frank Rosenblatt et Bernard Widro. Ils le comparent aux neurones capables de répondre à des stimuli externes d'une maniere qui imite les vrais neurones biologiques dans leur publication « The perceptron : a probabilistic model for information storage and organization in the brain » (Brain, Rosenblatt, 1958). VOIR figure 2

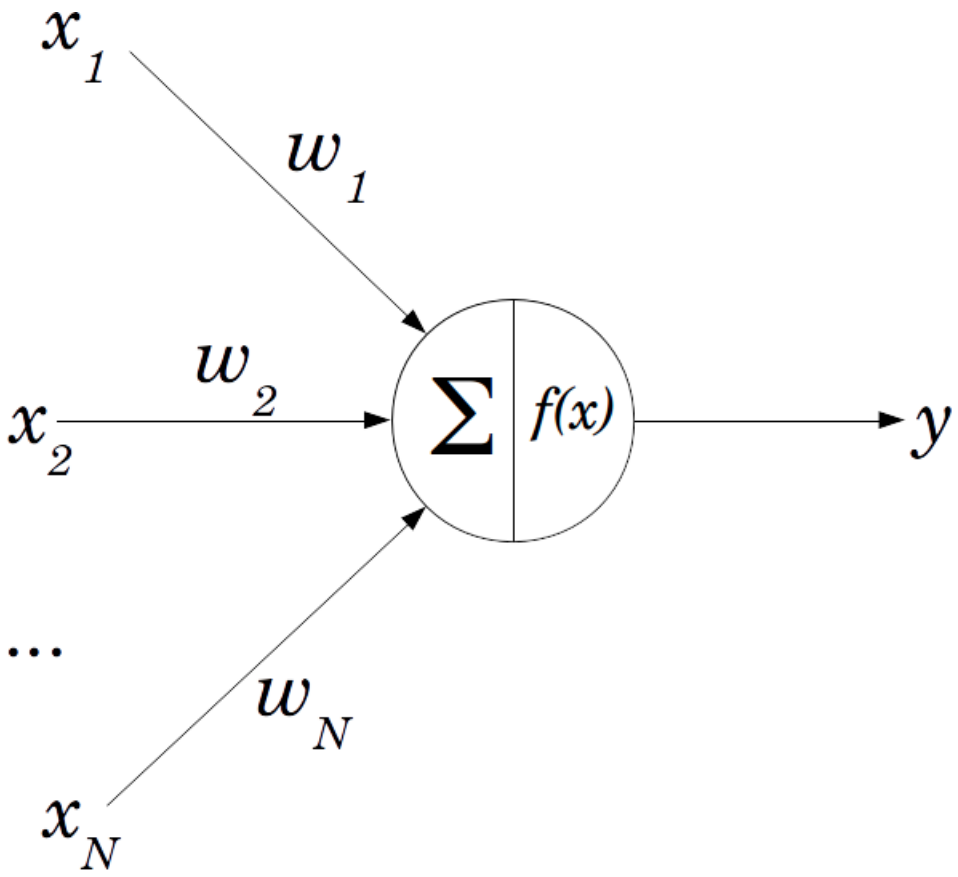


FIGURE 2 – simuli de neurone biologique

Historique

En 1949, D. Hebb initie, dans son ouvrage "The Organization of Behavior", la notion d'apprentissage. Deux neurones entrant en activité simultanément vont être associés (c'est-àdire que leurs contacts synaptiques vont être renforcés). On parle alors de la loi de Hebb et d'associationnisme.

En 1958, F. Rosenblatt développe le modèle du Perceptron. C'est un réseau de neurones inspiré du système visuel. Il possède deux couches de neurones : une couche de perception (sert à recueillir les entrées) et une couche de décision. C'est le premier modèle pour lequel un processus d'apprentissage a pu être défini.

S'inspirant du perceptron, Widrow et Hoff, développent, dans la même période, le modèle de l'Adaline (Adaptive Linear Element). Ce dernier sera, par la suite, le modèle de base des réseaux de neurones multi-couches.

En 1969, Les recherches sur les réseaux de neurones ont été pratiquement abandonnées lorsque M. Minsky et S. Papert ont publié leur livre « Perceptrons » (1969) et démontré les limites théoriques du perceptron, en particulier, l'impossibilité de traiter les problèmes non linéaires par ce modèle.

En 1982, Hopfield développe un modèle qui utilise des réseaux totalement connectés basés sur la règle de Hebb pour définir les notions d'attracteurs et de mémoire associative. Puis en 1984 c'est la découverte des cartes de Kohonen avec un algorithme non supervisé basé sur l'auto-organisation et suivi une année plus tard par la machine de Boltzman (1985).

Une révolution survient alors dans le domaine des réseaux de neurones artificiels : une nouvelle génération de réseaux de neurones, capables de traiter avec succès des phénomènes non-linéaires, proposé par Werbos : le Perceptron Multi-Couche apparaît en 1986 introduit par Rumelhart, et simultanément, sous une appellation voisine, chez Le Cun (1985). Ces systèmes reposent sur la rétropropagation du gradient de l'erreur dans des systèmes à plusieurs couches, chacune de type Adaline de Bernard Widrow, proche du Perceptron de Rumelhart.

Deuxième partie
LE RÉSEAU DE NEURONES

Le neurone formel est un modèle mathématique du neurone biologique. C'est est une fonction non linéaire, paramétrée, à valeurs bornées.

1 Structure d'un neurone formel

Un neurone formel se modélise comme un système à m entrées, pondérées par des coefficients parfois appelés coefficients synaptiques (poids) et une sortie.

La fonction de transfert également appelée fonction d'activation du neurone est généralement une fonction non linéaire croissante et bornée, la fonction d'activation du neurone s'exprime comme une somme pondérée de l'activité des neurones.

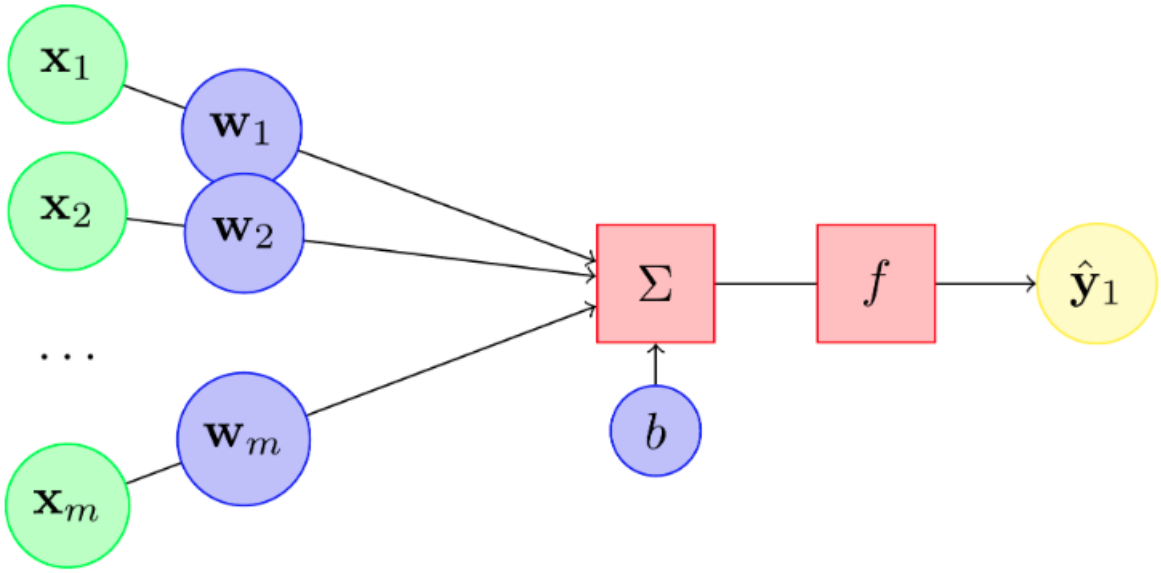


FIGURE 3 – neurone formel

— les entrées $(X_1, X_2, ..., X_m)$

— Les poids $(W_1, W_2, ..., W_m)$

— Une fonction Somme Σ
— Un biais (b)
— Une fonction d'activation
— Une sortie
des paramètres, notés W et b , influençent le fonctionnement du neurone.

2 Principe de fonctionnement des réseaux de neurones

Le fonctionnement d'un neurone artificiel, proposé par Warren McCulloch et Walter Pitts en 1943, est la suivante :

— Chaque neurone recevant une valeur en entrée va calculer une somme pondérée a partir d'un ensemble de poids et d'un biais qui lui sont propres : $t = \sum_{i=1}^{m} W_i X_i + b$

— Cette somme pondérée sera ensuite soumise à une « fonction d'activation », dont le resultat constituera la sortie du neurone : $\hat{y} = f(t)$
Une notation compact peut s'écrire : $Y = f(W^\top X + b)$

2.1 Exemples de fonctions d'activation

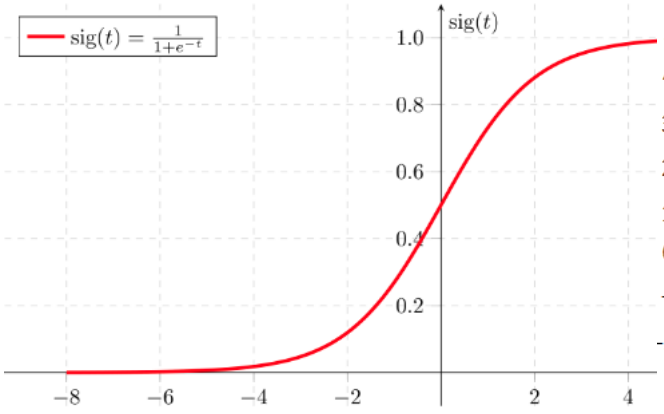


FIGURE 4 – fonction sigmoide : $f(x) = \dfrac{1}{1+e^{-x}}$

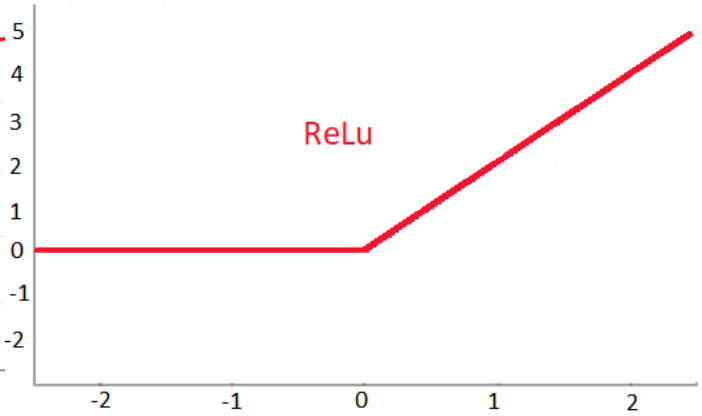


FIGURE 5 – fonction Relu (Seuil) : $f(x) = \max(0, x)$

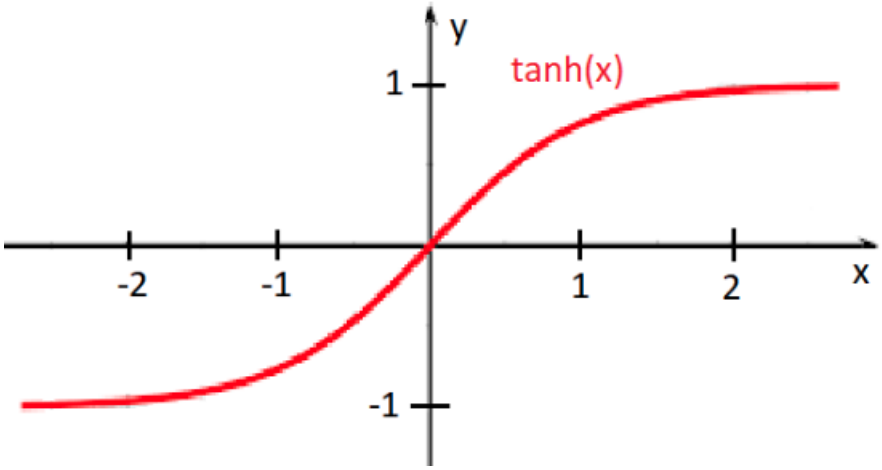


FIGURE 6 – fonction tangente Hyperbolique : $f(x) = \dfrac{1-e^{-x}}{1+e^{-x}}$

Remarque 2.1. *Dans le cas d'un problème de* ***régression****, il n'est pas nécessaire de transformer la somme pondérée reçue en entrée. La fonction d'activation dans ce cas est la fonction identité, elle retourne ce qu'elle a reçu en entrée.*

Dans le cas d'un problème de ***classification binaire****, on peut utiliser une fonction de* ***seuil*** *:*

$$f(t) = \begin{cases} 1 & si \quad t \leq 0 \\ 0 & sinon \end{cases}$$

2.2 L'apprentissage des reseau neurone

On peut considerer les réseaux de neurones comme une boite noire contenant l'information qu'elle doit aprendre et mémoriser. Mais au demarrage, la boite noire est vide et

ne contient aucune information, ni aucune connaissance sur son sujet, c'est pourquoi un apprentissage est necessaire.

L'aprentissage des réseaux de neurones consiste à adapter ses differents paramètres(poids) par un algorithme itératif d'ajustement ou d'adaptation en lui permettant de prendre en consideration toutes les données (exemples) qui lui sont fournies à son entrée et ainsi ajuster ses paramètres pour trouver le juste milieu afin de prendre en charge n'importe quel exemple ou donnée apparaissant à son entrée.

Il existe plusieurs modes d'apprentissage, à savoir :

— **l'apprentissage supervise** : A chaque donnée fournie à la machine, on lui associe un label(étiquette)

— **l'apprentissage non-supervise** :On n'indique pas a la machine ce qu'on veut comme sortie, la machine va tout simplement regrouper les données par similitude.

— **l'apprentissage par renforcement**, la machine va apprendre par récompense. C'est-a-dire, lorsque celle-ci améliore sa performance, elle recoit une récompense et ceci va donc augmenter les chances que ce comportement se réproduise

source :**CHAOUCHE, Yannis, 2018. Identifiez les differents types de problemes de machine learning : OpenClassrooms**.

Dans le cadre de ce rapport, on s'intéressera à l'apprentissage supervisé. L'objectif de cet apprentissage consistera à modifier progressivement la matrice des poids W de manière à minimiser la fonction **Erreur : E**

Le principe est le suivant :

— A partir d'un ensemble de m données d'apprentissages $(X_i, W_i) \quad i \in \{1, ..., m\}$

— On va calculer la prédiction $\hat{y}_i$ pour chaque valeur de X_i
soit $\hat{y}_i = f(t) \quad$ avec $\quad t = \sum_{i=1}^{m} W_i X_i + b$
f une fonction d'activation, par exemple $f(x) = \dfrac{1}{1+e^{-x}}$

— En comparant ces prédictions aux valeurs attendues. On calcule une fonction d'erreur, telle que :

soit **l'erreur quadratique moyenne** $Erreur(\hat{y}_i, y_i) = \dfrac{1}{2}(\hat{y}_i - y_i)^2$

soit **l'entropie croisée**

$$Erreur(\hat{y}_i, y_i) = -\left[y_i \log f(X_i) + (1 - y_i) \log(1 - f(X_i))\right]$$

— On modifiéra les poids de manière a minimiser cette fonction d'erreur, par une méthode de descente de gradient

L'algorithme commence par une initialisation aléatoire du vecteur poids $W_0^0 = (W_1^0, W_2^0, ..., W_m^0)$ Puis, à chaque observation, on ajuste les poids de connexion de sorte à réduire l'erreur de

prédiction dans la direction opposée au gradient. En effet ce gradient indique la direction de plus grande variation de la fonction d'erreur pour trouver le minimum de cette fonction.

Formellement, à une itération de l'algorithme, on tire une nouvelle observation $(\overrightarrow{X_i}, y_i)$ et on actualise, pour tout j, les poids de la façon suivante :

$$W_j \longleftarrow W_j - \eta \frac{\partial Erreur(\hat{y}_i, y_i)}{\partial W_j}$$

avec η **:taux d'apprentissage**

Cet algorithme a un hyperparamètre $\eta > 0$, qui est la pas d'apprentissage. Il joue un role très important :

— S'il est trop grand, l'algorithme risque d'osciller autour de la solution optimale, voire diverger.

— S'il est trop faible, l'algorithme va converger très lentement

Donc il est important de bien choisir ce pas d'apprentissage η. Il existe des algorithmes permettant d'adapter ce pas d'apprentissage. Par exemple recherche linéaire par rebroussement
source :**introduction au machine learning chloé-agathe azencott**

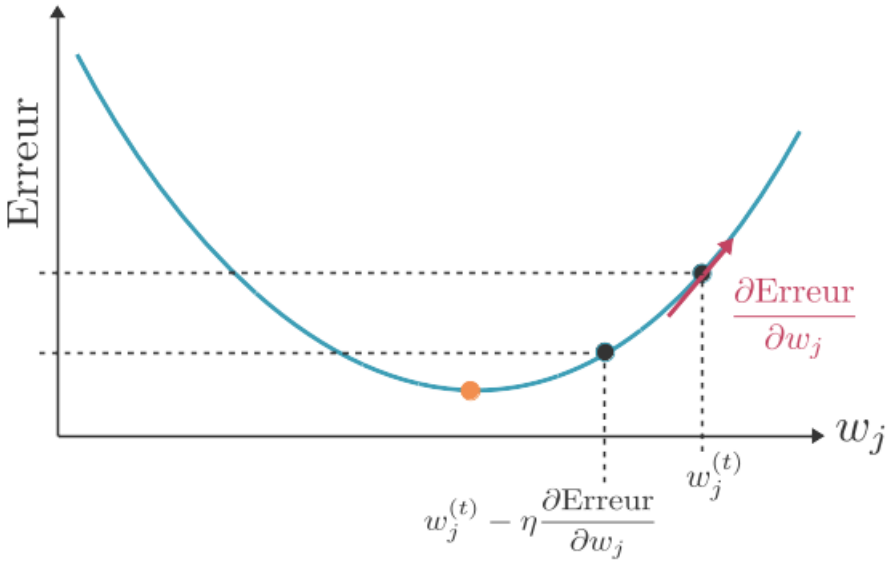


FIGURE 7 – Un déplacement dans le sens opposé au gradient (flèche rose) rapproche wj de la valeur minimisant l'erreur (point orange).

source : **Utilisez des modèles supervisés non linéaires openclassroom**

3 Généralité des Réseaux neurones multi-couches

3.1 Le Perceptron multi couche

Comme nous l'avons déja dit le cerveau humain est composé de millier et des milliers de neurones, alors il est évident qu'un simple neurone et seul ne peut rien faire à lui tous seul, il lui faut la coopération d'autres neurones. En suivant ce raisonnement il est évident qu'il faut trouver une architecture qui relie les neurones entre eux, en créant une liaison entre les neurones pour créer un réseau de neurones.

Cette nouvelle architecture est le perceptron multicouches (ou MLP pour Multi Layer Perceptron en anglais).

L'idee principale est de grouper des neurones dans une couche. En placant ensuite bout à bout plusieurs couches et en connectant complètement les neurones de deux couches adjacentes. Les entrées des neurones de la deuxieme couche sont donc en fait les sorties des neurones de la première couche

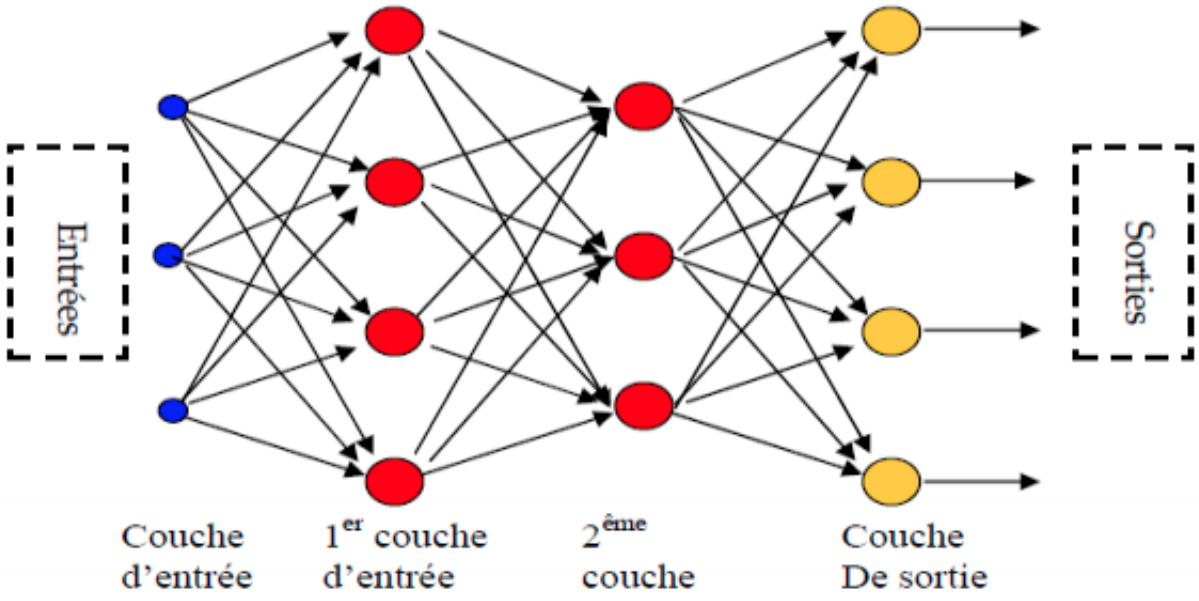


FIGURE 8 – Réseau neurone multi-couches

3.2 Méthode de rétopropagation du gradient

La rétropropagation est actuelement l'outil le plus utilisé dans le domaine des réseaux de neurones pour l'apprentissage de réseau. C'est une technique de calcul des dérivées qui peut d'être appliquée à n'importe quelle structure des fonctions dérivables.

Avant de définir la règle d'aprentissage, examinons la relation entre les sorties du réseau et les entrées, d'une part, et les poids d'autre part.

Pour un réseau multicouches à M entrees et N sorties, compose de L couches (couches cachées et couche de sortie) les états des neurones sont donnés par les équations suivantes :

$$S_i^k = \sum_{j=0}^{n_{k-1}} W_{ij}^k O_j^{k-1}(t) + \theta_i^k \qquad i \in \{1,2,3,....,n_k\} \qquad k \in \{1,2,....,L\} \tag{1}$$

$$O_0^k = 1 \qquad k \in \{1,2,....,L\} \tag{2}$$

$$O_i^0 = X_i(t) \qquad i = 1,2,....,m \tag{3}$$

$$Y_i = 1 \qquad i = 1,2,....,n \tag{4}$$

$$O_i^k = f^k\left[S_i^k\right] = f^k\left(\sum_{j=0}^{n_{k-1}} W_{ij}^k O_j^{k-1}(t) + \eta_k\right) \tag{5}$$

— Pour la couche k : $f^k(.)$ est sa fonction d'activation

— n_k est le nombre de neurones, O_i^k est la sortie du neurone i

— W_{ij}^k est le poids synaptique entre le neurone i de la couche k et le neurone j de la couche precedente (k-1)

— θ_i^k bais ou la valeur de seuil interne du neurone i

— $Y_i(t)$ et $X_i(t)$ sont les i^{emes} composantes du vecteurs d'entrées $X(t)$ et du vecteur de sortie $Y(t)$

La fonction d'activation généralement choisie est la fonction

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

L'objectif de la méthode de la rétropropagtion est d'adapter les poids W_{ij}^k de façon à minimiser la valeur moyenne de l'erreur sur l'ensemble d'entrainement. La fonction coût ou erreur la plus utilisée est donnée par :

$$E = \frac{1}{2}\sum_{t=1}^{T}\left[Y^d(t) - Y(t)\right]^2$$

Avec Y^d est le vecteur de sortie désirée, Y le vecteur de sortie effective du reseau et T la longueur de l'ensemble d'aprentissage.

On commence l'entrainement par un choix aléatoire des valeurs initiales des poids. On présente le premier vecteur d'entree. Une fois la sortie du réseau, l'erreur correspondante et le gradient de l'erreur par rapport à tous les poids sont calculées, les paramètres sont ajustés dans la direction opposée du gradient de l'erreur. On refait la même procédure pour tous les exemples d'aprentissages. Ce procesus est repété jusqu'à ce que les sorties du réseau soient suffisamment proches des sorties désirées.

Les parametres du réseau sont alors ajustés par la méthode de gradient en utilisant la

formule itérative :

$$W_{ij}^k(n) = W_{ij}^k(n-1) + \Delta W_{ij}^k(n)$$
$$\Delta W_{ij}^k(n) = -\mu \frac{\delta E}{\delta W_{ij}^k(n)}$$
$$\frac{\delta E}{\delta W_{ij}^k(n)} = \sum_{t=1}^{T} \frac{\delta E(t)}{\delta W_{ij}^k(n)}$$
$$\frac{\delta E(t)}{\delta W_{ij}^k(n)} = -\delta_i^k(t) O_j^{k-1}(t)$$

Où μ est une constante appelé pas d'aprentissage, n est le numéro de l'itérartion. δ_i^k est l'erreur équivalente à la sortie du neurone i de la couche k, pour les neurones des couches de sortie

$$\delta_i^k(t) = f^{k\prime}\left[S_i^k\right] \sum_{j=1}^{n_{k-1}} \delta_j^{j+1}(t) W_{ij}^{k+1}(n)$$

Alors, on a

$$W_{ij}^k(n+1) = W_{ij}^k(n) + \mu \delta_i^k(t) O_i^{k-1}$$

sources :
`https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf`
`http://www.anyflo.com/bret/cours/rn/rn5.htm#exemple`

Troisième partie
Réseau Neurone Convolutif

4 Généralité

Le réseau de neurone convolutif ou Convolution Neural Network (CNN) en anglais est un type de réseau neuronal très utilisé pour la classification d'image ou la reconnaissance visuelle. L'idée principale des CNN est d'essayer de faire ressortir certaines parties de l'image qui pourraient être intéressantes et les analyser, independamment de leurs positions dans l'image.
`https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`

5 Principe de CNN

Le principe de la convolution est de construire une nouvelle image, où chaque pixel est construit à partir des pixels environnants

Définition 5.1. *une image est une matrice de valeurs correspondant aux pixels. L'image possède un canal si elle est en noir et blanc ou trois canaux si elle est en couleur. C'est à*

dire qu'on a 3 valeurs pour chaque pixels. Ces trois valeurs représentent le niveau de rouge, vert et bleu. Les valeurs d'un pixel sont mappées sur 256 bits, c'est-à-dire allant de 0 à 255.

Une matrice de valeurs de pixels est sous la forme de [LARGEUR, HAUTEUR, CANAUX].
Par exemple, pour une image d'une taille de 100 par 100 pixels, sa matrice correspondante aura une taille de 100 (longueur) par 100 (largeur) par 3 (nombre de canaux).

FIGURE 9 – image à la matrice

Un pixel a une valeur spécifique de rouge, vert et bleu qui représente la couleur d'un pixel donné.

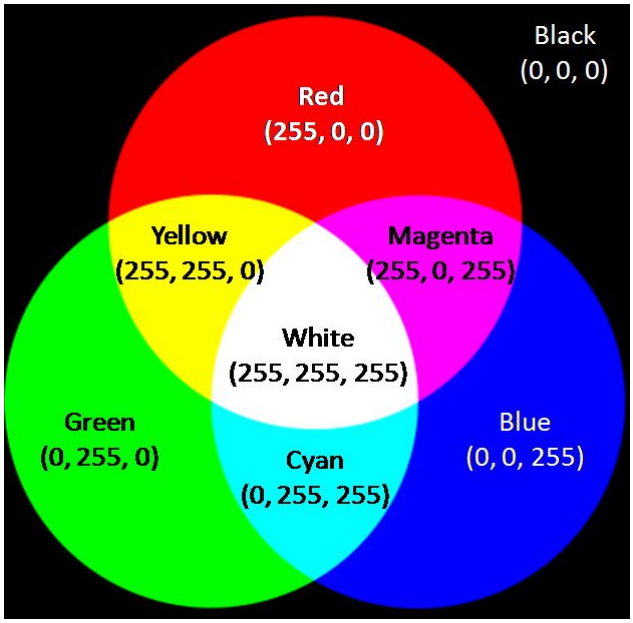


FIGURE 10 – pixel

Définition 5.2. *Le filtre est une matrice de valeurs, de petites dimensions et bien souvent de forme carrée. La taille de filtre la plus utilisée est de* 3×3.

Elle sert à faire ressortir certaines caractéristiques d'une image donnée (couleur, contour, luminosité, netteté, etc...). Ce filtre va être déplacé par pas successifs sur l'ensemble de l'image

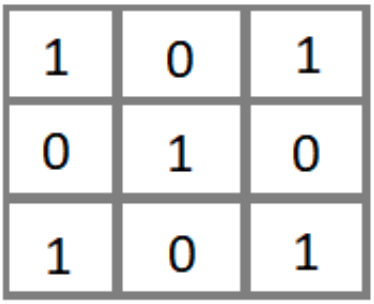


FIGURE 11 – filtre

Pour chaque position du filtre, les valeurs des deux matrices en superposition (filtre et image à traiter) sont multipliées. Chaque valeur est projetée dans une nouvelle matrice. Cette matrice représente une nouvelle image qui fait ressortir les caractéristiques recherchées au travers du filtre.

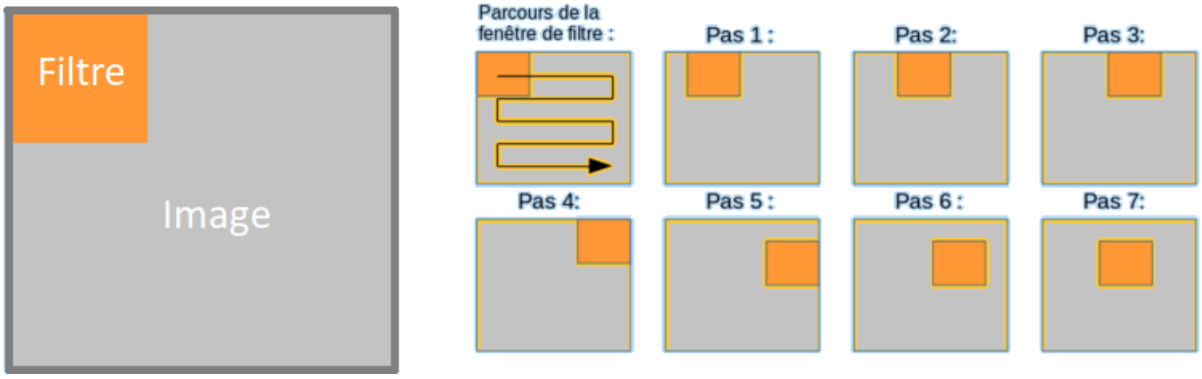


FIGURE 12 – les pas du filtre

`www.natural-solutions.eu/blog/la-reconnaissance-dimage-avec-les-rseaux-de-neurones`

Remarque 5.1. *Le réseau de neurones convolutif est très similaire à un réseau de neurones artificiels acycliques dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux.*

Ils ont fait beaucoup de succès dans les reconnaissances faciales, la détection d'objets très utilisée dans les robots et les voitures automatiques. En gros, tout ce qui concerne la vision par ordinateur et les images.

6 Architecture du CNN

Nous pouvons dire que les réseaux de neurones convolutionnels comportent deux parties bien distinctes. En entrée, une image est fournie, suivie de la partie convolutif puis la

partie calssifieur.

La partie convolutive est constituée par un empilement de 3 ou 4 couches de traitement indépendantes

— **La couche de convolution**

— **La couche de correction ReLU**

— **La couche de Pooling**

— **La couche de Full connected**

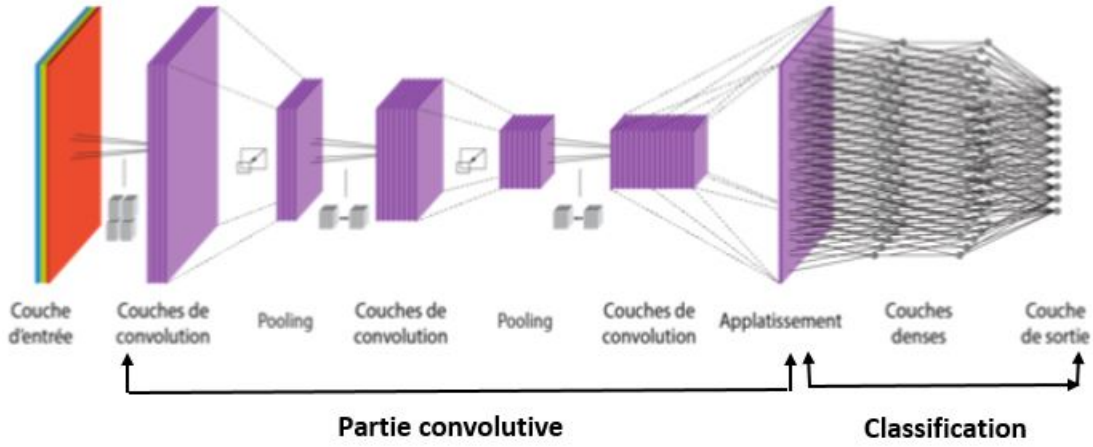


FIGURE 13 – Architecture standard d'un réseau de neurone convolutionnel

Couche de convolution(CONV)

La couche de convolution est la première couche de la partie convolutive. Son but est de repérer la présence d'un ensemble de features(caracteristques) dans les images reçues en entrée par un filtrage convolutif, c'est -à-dire faire "glisser" une fenêtre représentant la feature sur l'image, et de calculer le produit de convolution entre la feature et chaque portion de l'image balayée.

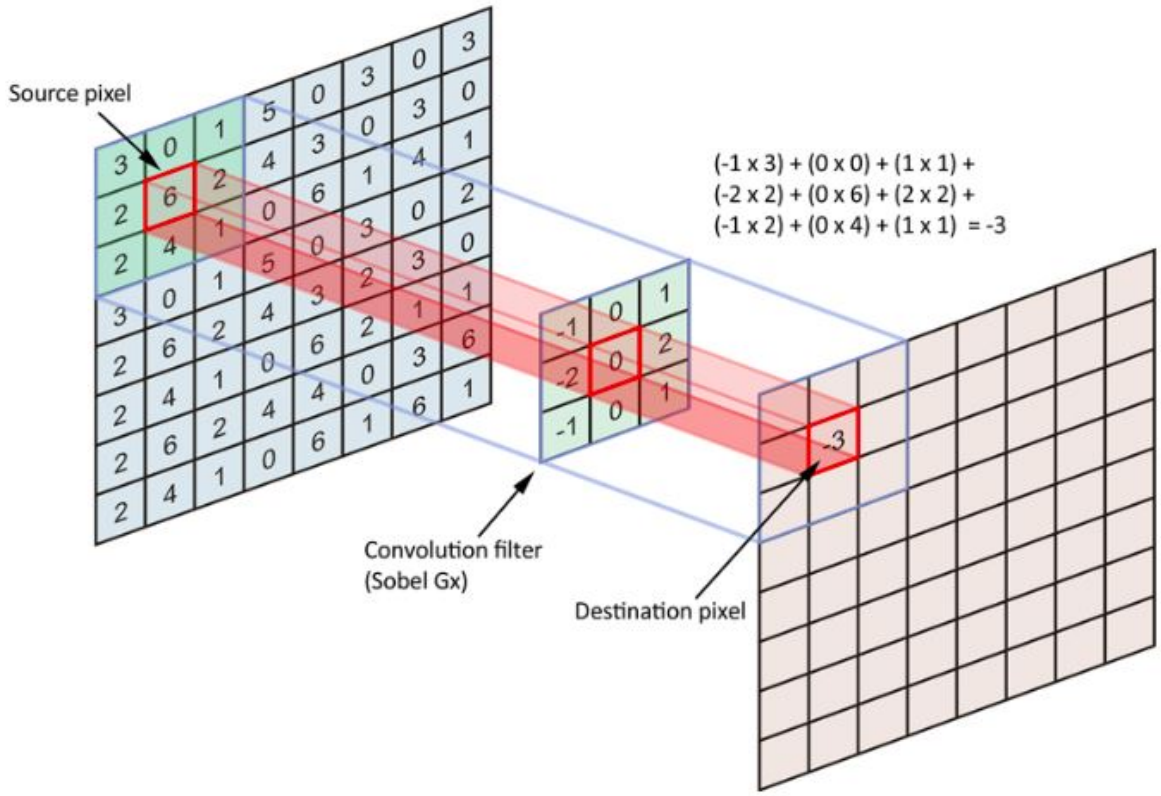


FIGURE 14 – convolution

Trois paramètres permettent de dimensionner le volume de la couche de convolution **la profondeur, le pas et la marge.**

1. **Profondeur de la couche** : nombre de noyaux de convolution (nombres de filtres), la taille F des filtres et chaque filtre est de dimensions $F \times F \times D$ pixels.

2. **Le pas (S)** : contrôle le chevauchement des champs récepteurs. Plus le pas est petit, plus les champs récepteurs se chevauchent et plus le volume de sortie sera grand.

3. **La marge a (0) ou zero padding (P)** : Cette marge permet de contrôler la dimension spatiale du volume de sortie. En particulier, il est parfois souhaitable de conserver la même surface que celle du volume d'entrée.

Couches de correction (RELU)

La fonction ReLU (abréviation de Unités Rectifié linéaires) : $F(x) = \max(0, x)$ est appliquée à chaque pixel d'une image après la convolution, et remplace chaque valeur négative par un 0. Si cette fonction n'est pas appliquée, la fonction créée sera lineaire et le probleme XOR persiste puisque dans la couche de convolution, aucune fonction d'activation n'est appliquée

Couche de pooling (POOL)

Le pooling réduit la taille d'une image ainsi que la quantité de paramètres et de calcul dans le réseau. Il est donc fréquent d'insérer une couche de pooling entre deux couches convolutives successives d'une architecture CNN pour contrôler l'overfitting (sur-apprentissage).

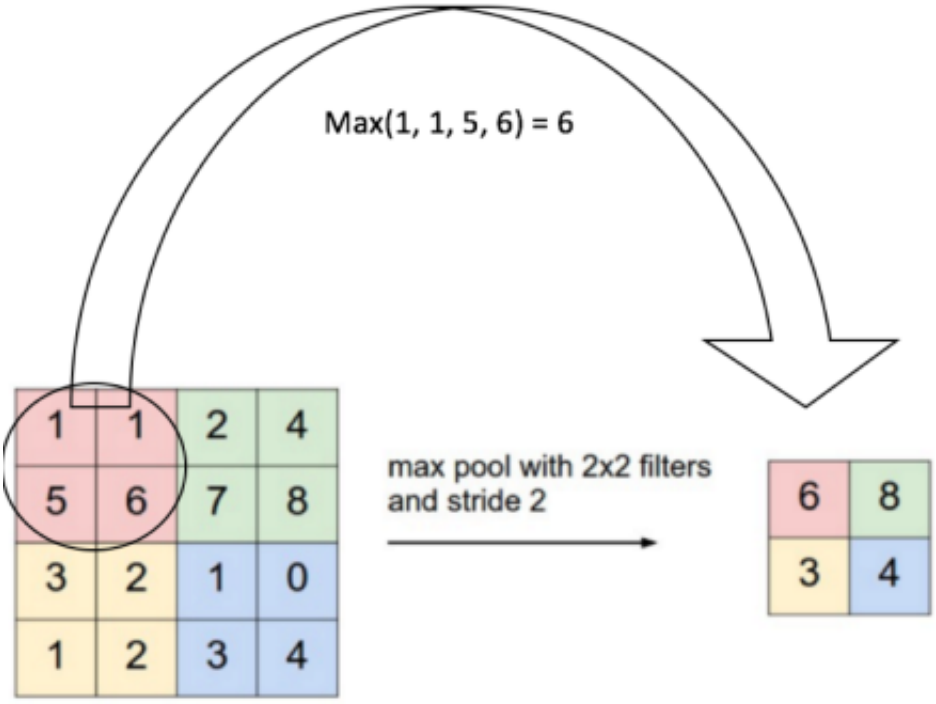


FIGURE 15 – couche de Max-pooling

Couche Full connected (FC) ou Flattening

Elle consiste simplement à mettre bout à bout toutes les images (matrices) que nous avons obtenues pour en faire un long vecteur. Ses matrices de nombres, sont récupérées ligne par ligne et ajoutées au vecteur final.

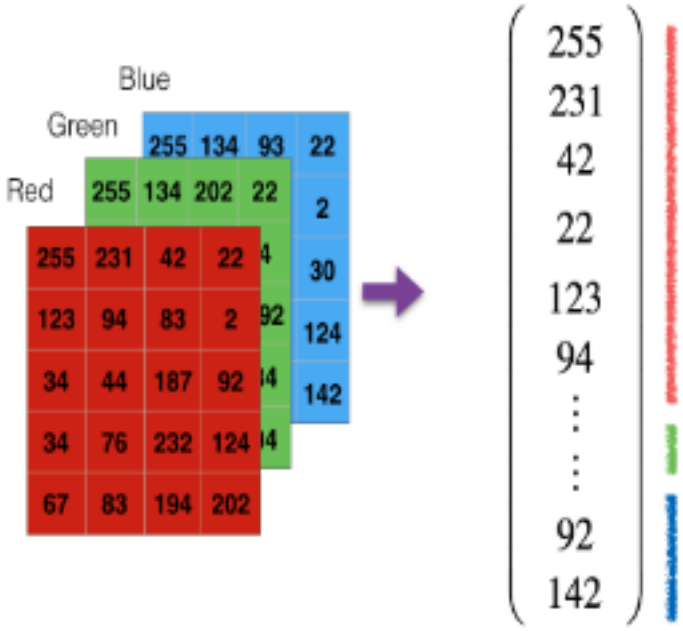


FIGURE 16 – flattening

Dans la partie classifieur, une fois l'image transformée par couche de Full connected, alors s'applique toutes les notions de la partie I de réseau de neuron ci dessus.

`https://ludo-louis.fr/differents-types-reseaux-neurones-reseau-convolution/`
`http://www.natural-solutions.eu/blog/la-reconnaissance-dimage-avec-les-rseaux-de-n`
`https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-netwo`
`https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`

Quatrième partie
Generative Adversarial Network (GAN)

7 Présentation des GANs

De nos jours, les exemples de deepfakes se multiplient, augmentent le doute quant à l'authenticité des informations vues ou entendues sur internet. A l'origine de ces vidéos contrefaites se trouve une nouvelle méthode de machine learning : les réseaux génératifs antagonistes, couramment appelés GANs pour "Generative Adversarial Networks".

Introduits en 2014 par le chercheur américain Ian Goodfellow, Yoshua Bengio, et Aaron Courville, les GANs sont capables de produire eux-mêmes des données(images)très semblales à la réalité

Son développement rapide au cours des dernières années et ses multiples applications en font l'une des découvertes récentes les plus prometteuses du Machine Learning. **Yann LeCun** (chercheur en IA chez Facebook) l'a présentée comme « l'idée la plus intéressante

des 10 dernières années dans le domaine du Machine Learning ».

source : `http://wintics.com/fr/quand-la-data-science-devient-creative-avec-les-gan/`

7.1 Objectif

L'objectif fixé d'un utilisateur de GAN, à partir de données (références ou authentiques), est de générer de nouveaux échantillons(images) provenant de la même distribution.

7.2 Qu'est ce que veut dire "proche" ?

Nous supposons que les données de l'ensemble de référence suivent une certaine distribution de probabilité $\mathbb{P}_{data}$. C'est cette distribution que nous cherchons à approximer à l'aide de notre modèle GAN. Une fois cette loi approximée par $\mathbb{P}_{model}$, nous pourrons échantillonner selon cette loi et ainsi fabriquer une nouvelle données(images) qui sera proche des données authentiques.

Le coeur du problème réside donc dans le fait de savoir comment approximer cette loi.

8 Architecture des GANs

En termes techniques, les GANs sont des algorithmes d'apprentissage non supervisé (unsupervised learning) de deux réseaux de neurones convolutifs qui sont :

★ **le Discriminateur**

★ **le Générateur**

8.1 Discriminant D

L'objectif du discriminant est de distinguer si une image provient d'une base de données authentiques ou semble avoir été créée artificiellement.

Lors de l'entraînement, le discriminant se comportera comme un classifieur binaire. plus rigoureusement, comme une fonction qui prend en argument une image et qui renvoie une probabilité de son authenticité comprise entre 0 (un faux) et 1 (une image réelle)

8.2 Générateur G

Le générateur est la partie principale du GAN. Son role principal est de générer des données (telles que des images, de la vidéo, de l'audio ou du texte) à partir des données existantes

Plus rigoureusement, le générateur peut être défini comme une fonction P_G, paramétrée par θ. Cette fonction prend en argument un vecteur latent noté z, de taille arbitraire,

et transforme ce vecteur en image.

Ensuite nous cherchons à trouver le meilleur θ de sorte que la distribution de probabilité des images générées $P_G(.\theta)$ soit la plus proche possible de la distribution de probabilité réelle des données $P(.\theta)$.

D'une manière plus imagée, voici à quoi ressemble l'architecture du GAN

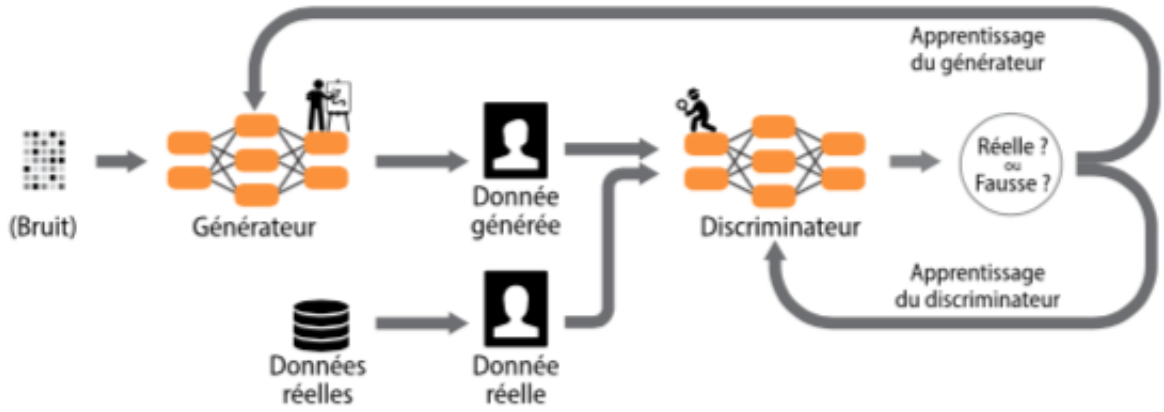


FIGURE 17 – Architecture du GAN

Cette approche, relativement recente, constitue l'une des plus importantes avancées en intelligence artificielle et un très grand nombre de variantes ont été proposées : DCGAN, cGAN, WGAN, CycleGAN, ProGAN, StyleGAN, BigGAN, GANSynth, MaskGAN, etc.

9 Processus d'apprentissage du GAN

Le mode d'entraînement est antagoniste, c'est à dire que le discriminant et le générateur vont "jouer" l'un contre l'autre. C'est ainsi que la fonction de coût du réseau GAN, sera représentée par un **minimax**.

Le générateur va générer des images et le discriminant essaie de deviner que ces images ne sont pas réelles. Grâce à ça, le générateur va pouvoir s'améliorer, et générer de nouvelles images, et ainsi de suite. A chaque itération, deux étapes vont être exécutées :

1. Au départ, on soumet un bruit aléatoire au générateur. le générateur crée des données (images) qui sont au départ loin des données(images) authentiques.

2. Le discriminateur, de son côté, va la comparer aux images réelles mis à sa disposition. et ensuite lui donner une probabilité de son authenticité comprise entre 0 (un faux) et 1 (une image réelle). Cette estimation est renvoyée au générateur.
3. Par le biais de ces nouvelles données, le générateur va essayer de tromper le discriminateur en créant une nouvelle donnée (image) plus proche des images authentiques.
4. Le discriminateur tente de déjouer le générateur en comparant les données crées au données authentiques, puis renvoye l'erreur au générateur en mettant à jour les poids.

5. Les deux réseaux s'entrainent simultanément en fournissant un retour d'information sur les modifications qu'ils apportent à chaque itération grâce à la rétropropagation.

6. Après plusieurs itérations, le discriminateur devient incapable de distinguer les données (images) originelles et artificielles.

   C'est ainsi que, les images sont jugées réalistes lorsque le générateur parvient à tromper le discriminateur

Notation

— P_{data} : la distribution des données réelles data

— P_z : une distribution connue qu'on pourra échantillonner de façon à obtenir un bruit aléatoire z utilisé comme entrant de notre réseau (usuellement gaussien ou uniforme,..).

— $G : z \to x$ tel que si $z \sim P_z$, alors $x \sim P_d$ $\quad G = G(z, \theta_g)$ un réseau de neurones de paramètres g correspond au générateur.

— $D : x \to y \in (0, 1)$ où y est la probabilité estimée que x provienne de P_{data} $D = D(x, \theta_d)$ un réseau de neurones de paramètres d .

10 Fonction de coût

Pour avoir un GAN qui génère des images similaires aux images réelles, nous essayons d'augmenter la similitude des données générées par le générateur avec les données réelles. Pour mesurer cette similitude, nous faisons récours aux fonctions de coûts. Les deux réseaux ont leurs propres fonctions de coûts lors de l'entrainement.

A partir des données originelles, le discriminateur fait une classification. L'erreur de classification va être calculée, et le gradient va se rétropropager dans le discriminant afin d'ajuster ses poids. en d'autre terme on entraîne D pour maximiser la probabilité d'assigner un label correctement. Alors sa fonction de coût est le suivant :

$$\mathbb{E}_{x \sim P_d} [\log(D(x)]$$

C'est l'espérance de bonnes classifications du discriminant

Le fait que le Discriminateur tente de maximiser les bonnes classifications, le générateur quant à lui va tenter de tromper le discriminant en générant des données aussi

proches de l'ensemble d'entraînement que possible. Alors sa fonction de coût est :

$$\mathbb{E}_{z\sim P_z}\left[\log(1 - D(G(z))\right]$$

C'est le terme antagoniste de la fonction de coût. Ce terme doit être minimisé par le générateur, puis maximisé par le discriminant.

Pour bien comprendre qui doit maximiser et qui doit minimiser la fonction de coût. Comme dans le processus d'entrainement, la sortie du discriminant correspond à la probabilité que l'image appartienne aux données réelles. **D(x)** doit être proche de 1 pour que le discriminant ait raison, et D(G(z)) doit être proche de 0 pour que le discriminant ait raison. Donc le générateur sera performant si D(G(z)) est proche de 1, ce qui correspond à minimiser le terme précédent. Voici finalement la fonction de coût complète est :

$$\min_G \max_D \mathbb{E}_{x\sim P_d}\left[\log(D(x)\right] + \mathbb{E}_{z\sim P_z}\left[\log(1 - D(G(z))\right]$$

Formellement, on obtient l'algorithme suivant (source Goodfellow et al. 2014)

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

  end for
  The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

FIGURE 18 – Algorithme du GAN

Exemple 10.1. ***Exemple GAN autoencoder*** *un autoencodeur (les réseaux diabolos) profond sur une base d'images :*

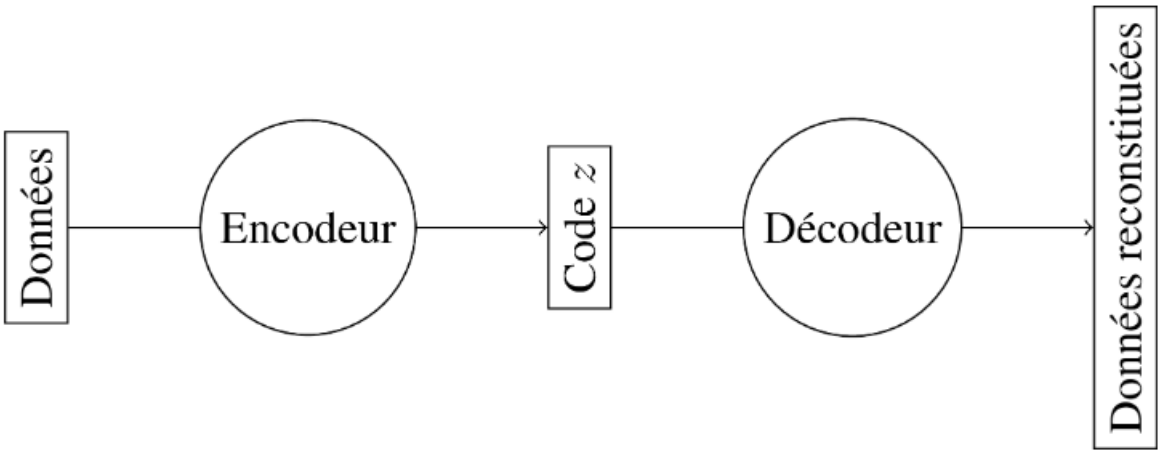


FIGURE 19 – Architecture du GAN

On tire un code z au hasard, et on le passe dans le décodeur pour générer une donnée synthétique.

Remarque 10.1. ***Génération d'images par GAN***

Les images produites par des GAN peuvent être si réalistes que l'œil humain ne peut pas imaginer qu'elles ont été créées de toutes pièces (ou plutôt de tous pixels) par un ordinateur.

Un challenge

Je vous invite à tester votre acuité visuelle, ou plutôt votre incapacité à distinguer des images réelles ou artificielles `http://www.whichfaceisreal.com/`

https ://www.thispersondoesnotexist.com/ Comme l'indique son URL, aucun des visages que vous verrez apparaitre à chaque rafraichissement de page n'est réel.

Exemples d'images générer par les GANs

Les images de chats présentées ci-dessous ne sont pas de vraies photos. Elles ont été générées par un modèle GAN entrainé sur la base de 10 000 photographies de chats. A noter que plus la base de données d'entrainement est grande, plus les images générées seront réalistes.

FIGURE 20 – Architecture du GAN

11 Quelques Avantages et Dangers des GANs

Depuis leur introduction en 2014, les GANs ont été utilisés avec beaucoup de succès dans plusieurs domaines applicatifs :

11.1 Domaines d'applications

— la génération automatique des images photoréalistes (visualisation, design, mode, style artistique, scènes de jeux vidéo, imagerie astronomique)

— L'augmentation de la résolution d'une image.

— le secteur de la conduite autonome : pour générer les modèle de carosserie des voitures

— le secteur de la médecine :la génération de médicaments, où les modèles génératifs couplés à l'apprentissage par renforcement permettent d'accélérer massivement la synthèse de nouvelles molécules.

11.2 Dangers

Tous les avantages ci dessus sont également la source d'inquiétude, notamment en terme de sécurité, en d'autre terme l'utilisation de GANs à mauvais escient :

En effet, imaginez-vous, assis dans votre voiture autonome qui roule tranquillement. A l'approche d'une intersection et à la vue d'un panneau STOP, votre voiture va normalement détecter la présence de ce dernier, l'identifie comme étant un panneau STOP, et s'arrêter avant de s'engager. Mais là, c'est tout le contraire qui se produit et cela pouurait avoir des conséquences très graves.

En fait, vous avez été victime de l'oeuvre des GANs appelée adversarial examples. L'algorithme a parfaitement fonctionné. Il n'y a pas eu de bug, l'algorithme a simplement identifié le panneau STOP comme un panneau différent, par exemple un panneau de priorité. C'est un travail de piratage

La Robustesse des algorithmes de machines Learning fasse à ce genre d'attaque est le sujet sur lequel je ferai mon stage à CEA

le travail est réalisé par Nicolas Papernot, Patrick McDaniel, Z. Berkay Celik (PSU), Somesh Jha (UofW), Anantrham Swami (USARL) et Ian Goodfellow
`https://arxiv.org/pdf/1602.02697.pdf`

Références

[1] Jakub Langr, Valdimir Bok *" GANs in Action : Deep learning with Generative Adversarial Networks"* .

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (2014). *« Generative Adversarial Networks »*
https ://arxiv.org/abs/1406.2661

Cinquième partie
CARTE DE KOHONEN (SOM)

11.3 Généralité

Les cartes auto-organisatrices Kohonen (Self-Organizing Map, SOM) appelées aussi cartes topologiques de Kohonen sont des méthodes neuronales développées par Teuvo Kohonen en 1982 pour réaliser des tâches de classification automatique. C'est un type de réseau de neurones artificiels dont l'apprentissage se déroule de manière non supervisée. Leurs rôles principaux consistent à faire une projection non linéaire des données de grande dimension sur un espace de faible dimension.
`http://cedric.cnam.fr/vertigo/Cours/ml/coursQuantificationVectorielle.html`

11.4 Principe de la méthode

Cette méthode consiste à projeter l'espace des données $\mathcal{D}$ sur un espace de faible dimension ; en général **1, 2 ou 3D**, appelé **Carte** noté $\mathcal{C}$. Il est constitué d'un ensemble de neurones interconnectés selon une structure de voisinage de graphe non orienté . Chaque neurone possède :

— Des coordonnées fixes sur l'espace de la carte

— Des coordonnées adaptables sur l'espace appelées aussi vecteurs référents, responsables d'une zone de cet espace

Cette projection doit respecter la structure des données. Elle est basée sur des méthodes de quantification vectorielle qui cherchent à partitionner l'ensemble des observations disponibles grouper par similarité. Ces groupements sont caractérisés par leurs structures de voisinage qui peuvent être matérialisées à l'aide d'un espace discret appelé « carte topologique ». Cet espace forme un treillis de faible dimension sur lequel les structures de voisinages sont prises en considération par le modèle. Le choix de la topologie dépend de la nature des problèmes.
Trois types de voisinages couramment utilisés pour les cartes de Kohonen, voisinages linéaire, rectangulaire, triangulaire.

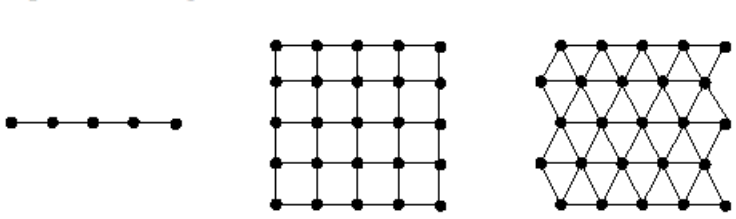


FIGURE 21 – Les cartes de Kohonen

La structure de la carte induit une distance discrète δ sur $\mathcal{C}$ de sorte que pour toute paire de neurone (c, r), la distance $\delta(c, r)$ est définie comme étant **la longueur du plus court chemin entre c et r sur le graphe de C**. Par exemple la distance $\delta(c, c1) = 6$ (le plus court chemin en rouge), celle entre c et c2 est de 4 (le plus court chemin en rouge). Il existe bien sûr plusieurs plus courts chemins possibles.

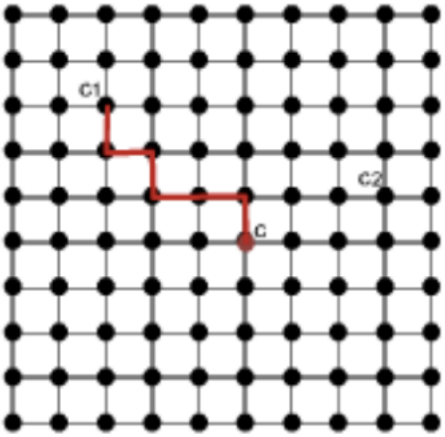


FIGURE 22 – Distance entre c et c1 : $\delta(c, c1) = 6$

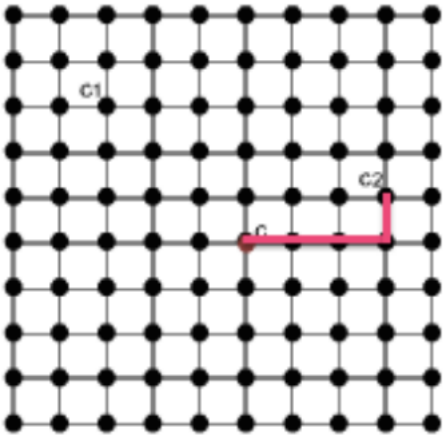


FIGURE 23 – Distance entre c et c2 : $\delta(c, c2) = 4$

Ceci permet de définir la notion de voisinage comme étant la longueur du chemin le plus court reliant deux neurones sur la carte. Ainsi le voisinage d'ordre **d** d'un neurone peut être défini comme l'ensemble des neurones situés à une distance d du neurone. Autrement dit, le sous-ensemble des neurones r dont la distance avec c (c'est-à-dire $\delta(c, r) \leq d$.

$$V_c(d) = \{r \in \mathcal{C}, \delta(c, r) \leq d\}$$

On montre sur la figure suivante Vc(1), Vc(2) et Vc(3) qui sont les voisinages du neurone c d'ordre 1, 2 et 3.

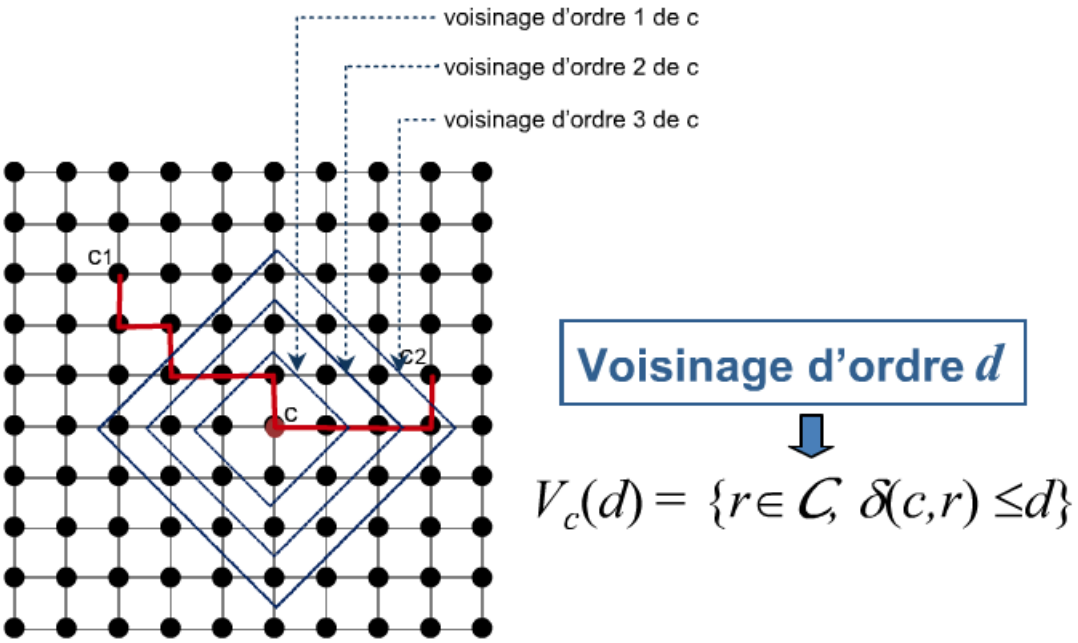


FIGURE 24 – Voisinages du neurone c d'ordre 1, 2 et 3.

Les distances qui lient les neurones les uns aux autres permettent de varier l'influence relative des différents neurones qui peut être quantifiée par une fonction de voisinage K. On utilise souvent une famille de fonction K^T paramétrée par T appelée parfois fonctions

d'activation pour mieux contrôler la taille du voisinage à partir de la distance δ définie sur la carte.

la fonction de voisinage à seuil

$$K^T(\delta) = \begin{cases} 1 & \text{si} \quad \delta \leq T \\ 0 & \text{sinon} \end{cases}$$

la fonction K de type gaussien

$$K^T(\delta) = \exp\left(\frac{-|\delta|}{T}\right) \quad et \quad K^T(\delta) = \exp\left(\frac{\delta^2}{-T^2}\right)$$

avec T est un terme de paramètre permettant de contrôler l'influence entre neurones

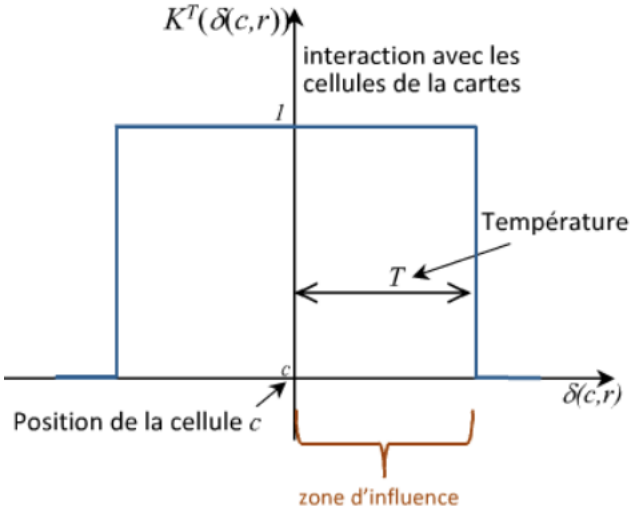


FIGURE 25 – Fonction de voisinage à seuil

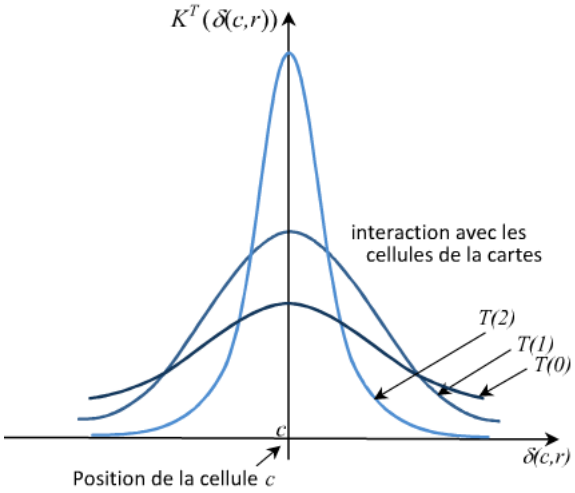


FIGURE 26 – Fonction de voisinage de type gaussien

11.5 Algorithme des cartes de Kohonen

Le but de cet algorithme d'apprentissage est de pouvoir projeter les données de façon à révéler la structure en formant des sous-ensembles suffisamment compacts. Pour y parvenir on procède en minimisant une fonction de coût qui respecte l'ordre topologique induit par la carte. La fonction de coût est :

$$J^T_{som}(\chi, \mathcal{W}) = \sum_{\mathbf{z}_i \in \mathcal{A}} \sum_{c \in C} K^T(\delta(c, \chi(\mathbf{z}_i)))\|\mathbf{z}_i - \mathbf{w}_c\|^2$$

avec

— $\mathcal{A}$ ensemble des échantillons

— χ représente une fonction d'affectation

— $\mathcal{W}$ représente l'ensemble des p vecteurs référents qui forment la carte

— L'expression $\chi(z_i)$ représente le neurone particulier de la carte $\mathcal{C}$ qui est affecté à l'observation z_i

— $\delta(c, \chi(z_i))$: représente la distance sur la carte C entre un neurone c quelconque et le neurone $\chi(z_i)$ affecté à l'observation z_i.

Algorithme de la carte

vecteurs référents sont obtenus par apprentissage à partir des données. Je vais maintenant présenter l'algorithme utilisé pour l'apprentissage de la carte.

— Etape 0 : Initialisation des vecteurs référents W.

— Etape 1 : A chaque itération n
On présente à l'entrée de la carte, un exemple d'apprentissage $X(n)$ choisi

— Comparaison de l'exemple à tous les vecteurs poids, le neurone gagnant j est celui dont le vecteur référents(poids) $W_j(n)$ est le plus proche de l'entrée $X(n)$:

$$j^* = argmin(d\left(W_i\left(n\right),\ X\left(n\right)\right) \quad (1)$$

Où i désigne tour à tour chaque neurone du réseau et Wi son vecteur référent.

— Évaluation du voisinage du neurone gagnant dans la carte

$$K_{j^*}^T\left(j\right) = K^T\left(\delta\left(j, j^*\right)\right) \quad (2)$$

— Mise à jour des poids pour tous les neurones de la carte, l'adaptation est d'autant plus forte que les neurones sont voisins de j* :

$$W_j\left(n+1\right) = \ W_j\left(n\right) + \ \alpha\left(t\right) * K_{j^*}^T(j)[X\left(n\right) - W_j\left(n\right)] \quad (3)$$

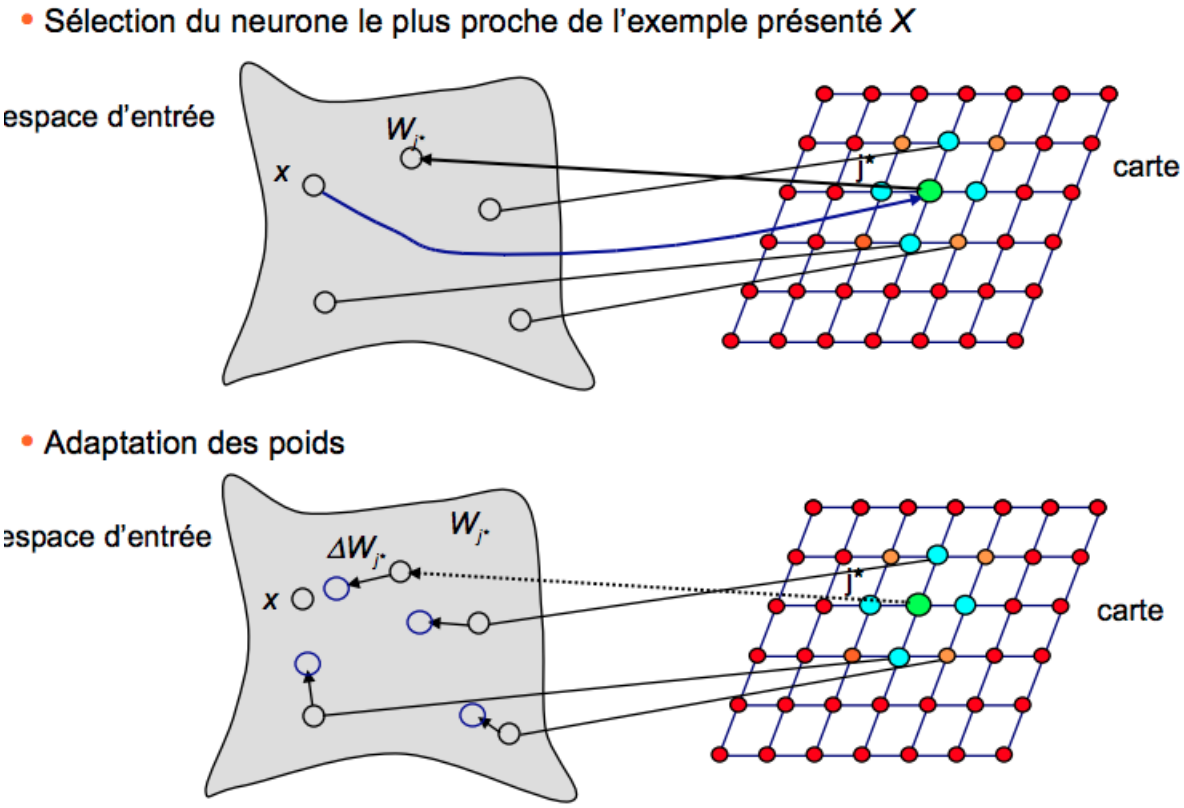
– Pour tout j appartenant au voisinage de j*
– α est un paramètre d'apprentissage appelé pas d'apprentissage, qui décroit en fonction de n, et est inférieur à 1. Cette fonction décroissante α est un élément important car permettant de trouver un compromis entre la vitesse de convergence et la qualité du dépliement de la carte.

— Utilisation d'un critère d'arrêt : nombres d'itérations, différence entre les vecteurs poids, erreur de quantification ...

Les phases 1, 2 et 3 s'appellent respectivement :
Compétition : Après sélection d'un exemple dans la banque de données, on cherche le neurone qui lui ressemble le plus. Ce neurone dit « neurone gagnant » est souvent noté BMU (Best Matching Unit).

Coopération : Dans cette étape, on détermine le voisinage du neurone gagnant. Il désigne la région de la carte la plus active et qui s'approche le mieux, au sens de la distance utilisée, de l'observation. La taille du voisinage du neurone gagnant est contrôlée par le rayon d'apprentissage.
Adaptation : On modifie le neurone gagnant pour qu'il ressemble plus à l'exemple puis on diffuse l'information sur son voisinage. Cette modification se fait à l'aide de la fonction d'activation qui permet de contrôler l'influence du neurone gagnant sur son voisinage.

Interprétation de l'algorithme



Les poids du neurone gagnant j* et de ses voisins sont rapprochés plus fortement du vecteur d'entrée X
L'amplitude de l'adaptation est fonction de la distance du neurone au neurone gagnant (flèches de longueurs différentes)

11.6 Classification automatique avec les cartes de kohonen

Une fois une carte apprise, nous disposons des profils types de chaque neurone appelés codebooks. Deux neurones proches dans la carte ont des profils types proches et un ensemble de codebooks contigus représentent un profil particulier dans les données.

Les codebooks constituent un jeu de données sur lequel on peut appliquer une classification hiérarchique ou un k-means pour regrouper les neurones similaires.

La classification d'un nouvel élément peut se faire comme suit :

— Présenter le nouvel élément à l'entrée de la carte

— Identifier le neurone gagnant, celui dont le codebook est le proche du nouvel élément

— Identifier le cluster du neurone gagnant

Références

[1] T. Kohonen, Biol. Cybern., . *"Self-organized formation of topologically correct feature maps (1982),"* volume (43), pages 59-69

[2] T. Kohonen, Springer*" Self-Organizing Map (1997) "* .

[3] Z. Lo, B. Bavarian, Biological Cybernetics.*"On the rate of convergence in topology preserving neural networks (1991)."* volume 63, pages 55-63.

ϕ

Sixième partie
CONCLUSION

A partir du comportement du cerveau humain et d'un modèle simple neuronal biologique, les chercheurs ont arrivé à construire des modèles neuronaux artificiels plus complexes. Ces réseaux de neurones sont un système avec plein de potentiel. Ils peuvent reconnaitre des objets sur une image, les compter et comprendre le langage naturel.

Il existe différentes architectures possibles pour construire des réseaux de neurones. Chacune d'entre elles possèdent des forces et des faiblesses, ce qui les rend plus en adéquation pour une tâche donnée. Certaines sont plus réputées que d'autres pour des taches bien spécifiques, entre autres les GANs. Leurs apparitions ont révolutionné l'intelligence artificielle. Leurs premières applications se sont surtout développées autour de la génération d'images mais les possibilités de cette nouvelle technologie sont très nombreuses. Elle semble être un outil puissant au service de l'innovation produit. Par contre un problème de sécurité se pose quant à l'utilisation des GANs à mauvais escient.

Septième partie
Annexe

11.7 PMC avec Comparaison

PMC avec Comparaison

DIARRASSOUBA SAKARIA

18/03/2020

Contents

Introduction

Dans cette analyse, nous utiliserons les données d'une entreprise d'automobile qui vient de faire sortir un nouveau produit: voiture de luxe absolument pas chère. Pour faire le mrketing de cette voiture en masse, elle a mis une publicité de cette voiture sur les réseau sociaux, et les utilisation de ces réseaux sociaux, voyant la voiture, décident d'acheter la voiture "OUI" ou "NON". Cette entreprise contient des données de base comme l'identité du client, l'âge, le sexe, le salaire et la reponse. L'objetif de notre analyse est de comparaison des performances du PMC et d'autres méthodes statistiques

importation des données

```
cust=read.csv("Social_Network_Ads.csv",header = T)
donnes=data.frame(cust[1],cust[2],cust[3],cust[4],cust[5])
names(donnes)=c("ID","Sexe","Age","salaire","acheté")
head(donnes)
```

```
##         ID   Sexe Age salaire acheté
## 1 15624510   Male  19   19000      0
## 2 15810944   Male  35   20000      0
## 3 15668575 Female  26   43000      0
## 4 15603246 Female  27   57000      0
## 5 15804002   Male  19   76000      0
## 6 15728773   Male  27   58000      0
```

Dans la variable **acheté** on a:
** 0 : l'utisateur n'a pas acheté la voiture**
** 1: l'l'utisateur a acheté la voiture**
Chaque ligne représente un client, On n'aura pas besoin de la variable qualitative **"Age"**, car elle n'impacte pas la variable **"acheté"** et de meme la varaiable **"ID"** n'a pas de correlation avec la variable **"acheté"**. Donc suprimons ces variables

```
client=donnes[,c(3:5)]
head(client)
```

```
##   Age salaire acheté
## 1  19   19000      0
## 2  35   20000      0
## 3  26   43000      0
## 4  27   57000      0
## 5  19   76000      0
## 6  27   58000      0
```

On déccoupe le jeu de données client en 2 parties de manière aléatoire, la première partie servira à l'apprentissage du modèle et la seconde pour la validation du modèle:

```
set.seed(1234)
names(client)=c("X1","X2","X3")
apprent_idx <- sample(nrow(client), 7/8 * nrow(client))
client_apprent <- client[apprent_idx, ]
client_test <- client[-apprent_idx, ]
dim(client_apprent)
```

```
## [1] 350   3
```

```
head(client_apprent)
```

```
##     X1     X2 X3
## 46  23  20000  0
## 249 41  52000  0
## 243 50  88000  1
## 248 57 122000  1
## 341 53 104000  1
## 253 48 134000  1
```

Standartdisation des données

On applique la standardisation sur les colonnes **Age** et **salaire**des données d'apprentissages et de test

```
apprent_ech=scale(client_apprent[1:2],center = T,scale = T)
x3=data.frame(client_apprent$X3)
train=cbind(apprent_ech,x3) # on ajoute la colonne ciblé : acheté
```

```
test_ech=scale(client_test[1:2],center = T,scale = T)
x3=data.frame(client_test$X3)
test=cbind(test_ech,x3)
head(test)
```

```
##            X1         X2 client_test.X3
## 8  -0.6217934  2.1971170              1
## 19  0.8486640 -1.0805581              1
## 20  1.0587293 -1.0536920              1
## 22  0.9536966 -0.5163682              1
## 25  0.8486640 -1.2148891              1
## 55 -1.1469568 -0.2745725              0
```

notre variable cible ** acheté** est codée en 0/1. L'algorithme n'a jamais pu converger lors de l'apprentissage du modèle. Et pourtant j'y ai passé du temps, en essayant de jouer sur les différents paramètres.

Selon moi, le problème vient de l'estimation des probabilités à l'aide de la fonction de transfert sigmoïde. Les valeurs sont très proches de 0 ou 1, là où les dérivées (gradients) sont quasi-nulles. Donc les corrections des

coefficients se font mal durant le processus d'apprentissage. Il n'est pas possible de progresser efficacement vers la minimisation de la fonction de perte.

C'est ainsi que j'ai codé la variable cible en **0.8 pour 1** et ** 0.2 pour 0** lorsqu'on utilise la fonction de transfert sigmoïde. On situera ainsi dans la zone où sa pente reste importante.

```
y_train <- ifelse(train$client_apprent.X3=="1",0.8,0.2)
print(table(y_train))
```

```
## y_train
## 0.2 0.8
## 225 125
```

```
y_test=ifelse(test$client_test.X3=="1",0.8,0.2)
print(table(y_test))
```

```
## y_test
## 0.2 0.8
##  32  18
```

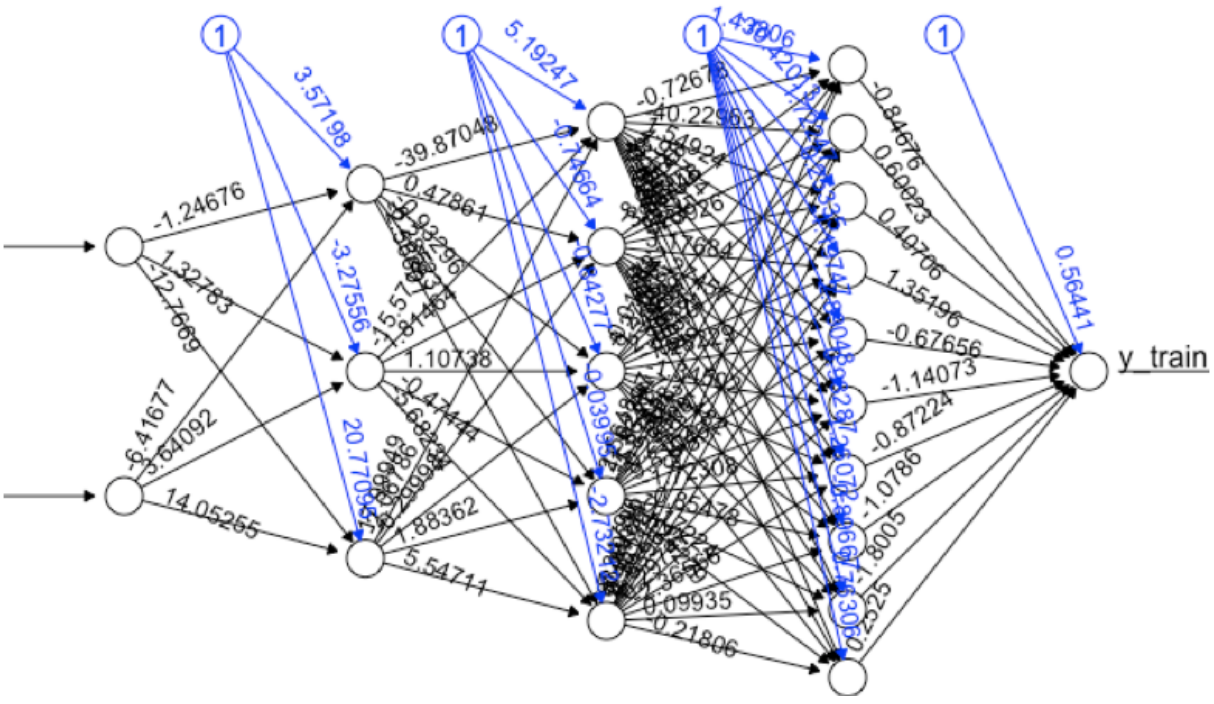
apprentissage du modele

On observe 2 paramètres clés : **hidden** permet de spécifier le nombre de neurones dans la couche cachée, s'il y a plusieurs couches, nous utilisons un vecteur
linear.output = logistic introduit la fonction d'activation logistique dans le neurone de sortie.

```
# le parametre hidden =c(3,5,10) est optimale
modele <- neuralnet(y_train ~ X1 + X2, data= train,hidden = c(3,5,10), act.fct = "logistic")

plot(modele)
```

Le réseau a 3 couches cachés de 3 unités, ensuite 5 unités puis 10 unités et le biais est bien présent sur chaque couche.



J' utilise la fonction compute() pour obtenir les probabilités d'affectation en prédiction sur l'échantillon test. Voici les 10 premières valeurs

```
#prédiction - proba d'affectation
proba_pred_modele <- compute(modele,covariate=test[-ncol(test)])
print(proba_pred_modele$net.result[1:10])
```

```
##  [1] 0.7587357 0.6867082 0.7528550 0.6029571 0.7450535 0.2114316 0.2149082
##  [8] 0.2041430 0.2112504 0.2184869
```

Je les compare au seuil 0.5 pour obtenir les classes prédites. Il est dès lors possible de confronter les valeurs observées et prédites de la variable cible.

Au passage, une fonction pour donner en quelque sorte la matrice de confusion

```
#fonction pour evaluation des modèles
evaluation.prediction <- function(y_obs,y_pred){
 #matrice de confusion
 mc <- table(y_obs,y_pred)
 print("Matrice de confusion")
 print(mc)
 #taux d'erreur
 err <- 1-sum(diag(mc))/sum(mc)
 print(paste("Taux d'erreur =", round(err,3)))
 #precision
 precision <- sum(diag(mc))/sum(mc)
 print(paste("Precision =",round(precision,3)))
}
```

Nous les comparons au seuil 0.5 pour obtenir les classes prédites. Il est dès lors possible de confronter les valeurs observées et prédites de la variable cible.

```
#traduire en "yes" "no" en comparant à 0.5
pred_modele <- ifelse(proba_pred_modele$net.result > 0.5,"yes","no")

# evaluation
evaluation.prediction(test$client_test.X3,pred_modele)
```

```
## [1] "Matrice de confusion"
##      y_pred
## y_obs no yes
##     0 28   4
##     1  1  17
## [1] "Taux d'erreur = 0.1"
## [1] "Precision = 0.9"
```

Résumé, j ai 3 mal classés, avec un taux d'erreu de **6%**, et une précision de ** 94% **

D'autres classifieurs

méthode SVM

On utilise le package **''e1071"** pour l'implémentation des SVM. Nous demandons à la procédure svm() de construire un classifieur linéaire (kernel = 'linear')

```
#library(e1071)
modele_svm=svm(y_train ~ X1 + X2, data=train,  kernel="linear")
```

```
pred_svm=round(predict(modele_svm,test))
evaluation.prediction(test$client_test.X3,pred_svm)
```

```
## [1] "Matrice de confusion"
##      y_pred
## y_obs  0  1
##     0 30  2
##     1  6 12
## [1] "Taux d'erreur = 0.16"
## [1] "Precision = 0.84"
```

Résumé, j ai 8 mal classés, avec un taux d'erreu de **16%**, et une précision de ** 84% **

```
X_train <- as.matrix(train[-ncol(train)])
X_test <- as.matrix(test[-ncol(test)])
```

méthode deepnet

```
#library(deepnet)
#apprentissage
set.seed(100)

deep_modele <- nn.train(x=X_train,y=y_train,hidden=c(5),numepochs=250)
```

```
#proba prediction
proba.pred.dpn <- nn.predict(deep_modele,X_test)
summary(proba.pred.dpn)
```

```
##        V1
##  Min.   :0.1229
##  1st Qu.:0.1942
##  Median :0.3852
##  Mean   :0.4086
##  3rd Qu.:0.5388
##  Max.   :0.7999
```

```
#prédiction
pred.dpn <- ifelse(proba.pred.dpn>0.5,"yes","no")
#evaluation
evaluation.prediction(test$client_test.X3,pred.dpn)
```

```
## [1] "Matrice de confusion"
##      y_pred
## y_obs no yes
##     0 28   4
##     1  4  14
## [1] "Taux d'erreur = 0.16"
## [1] "Precision = 0.84"
```

Résumé, j ai 8 mal classés, avec un taux d'erreu de **16%**, et une précision de ** 84% **

Package	Taux erreur	prédiction
neuralnet	0.06	0.94
Deepnet	0.16	0.84
SVM	0.16	0.84

l'idée était de voir un peu le comportement des différentes méthodes de R dans l'implémentation d'un perceptron multicouche. Au vu des performences, il ressort que le neuralnet a une très bonne prediction

11.8 Carte de Kohonene

projet carte de Kohonen

DIARRASSOUBA SAKARIA

13/03/2020

Contents

Présentation

Dans cette analyse, nous utiliserons les données sur les clients d'un centres commercial qui contiennent des données de base comme l'identité du client, l'âge, le sexe, le revenu annuel et le score des dépenses. L'objectif de cette analyse est d'identifier le segment de clientèle via la Carte de Kohonen, afin de comprendre quel est le segment de clientèle qui devient la cible de l'équipe marketing pour planifier les stratégies de marketing.

Importation des données

```
# importation des données
mall =read.csv("Mall_Customers.csv", header = TRUE)
mallCust=data.frame(mall[1],mall[2],mall[3],mall[4],mall[5])
names(mallCust)=c("ID","Sexe","Age","revenus_annuels","score_depense")
head(mallCust)
```

```
##   ID   Sexe Age revenus_annuels score_depense
## 1  1   Male  19              15            39
## 2  2   Male  21              15            81
## 3  3 Female  20              16             6
## 4  4 Female  23              16            77
## 5  5 Female  31              17            40
## 6  6 Female  22              17            76
```

nombre de variables manquantes

```
colSums(is.na(mallCust))
```

```
##              ID            Sexe             Age revenus_annuels
##               0               0               0               0
##   score_depense
##               0
```

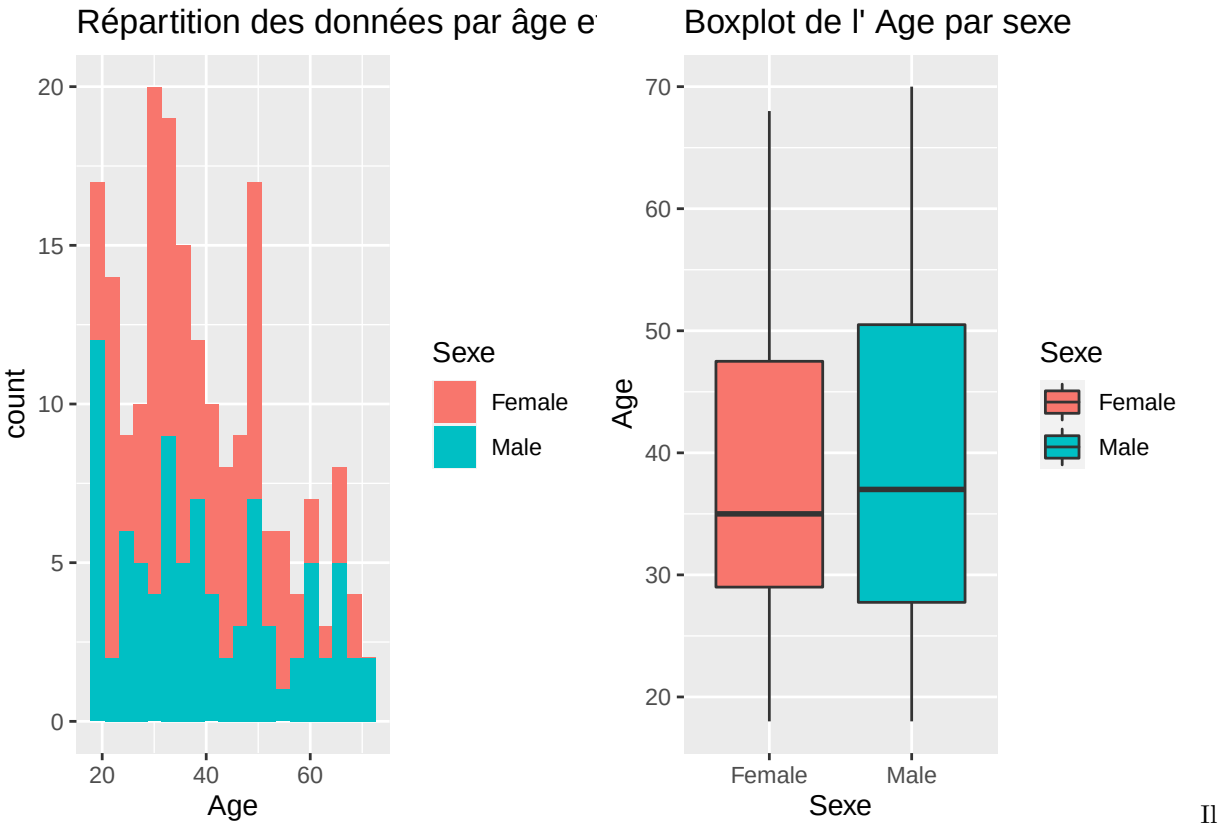
Age - Variable

Répartition des données par sexe.

```r
ageHist <- ggplot(mallCust, aes(Age, fill=Sexe)) + geom_histogram(bins = 20) +
          ggtitle("Répartition des données par âge et par sexe")

ageBox <- ggplot(mallCust, aes(x=Sexe, y=Age, fill = Sexe)) +
          geom_boxplot() +
          ggtitle("Boxplot de l' Age par sexe")

plot_grid(ageHist, ageBox)
```



Il ressort de répartition de que les femmes sont les plus fréquentes dans ce centre commercial ce qui est en adéquation avec la réalité

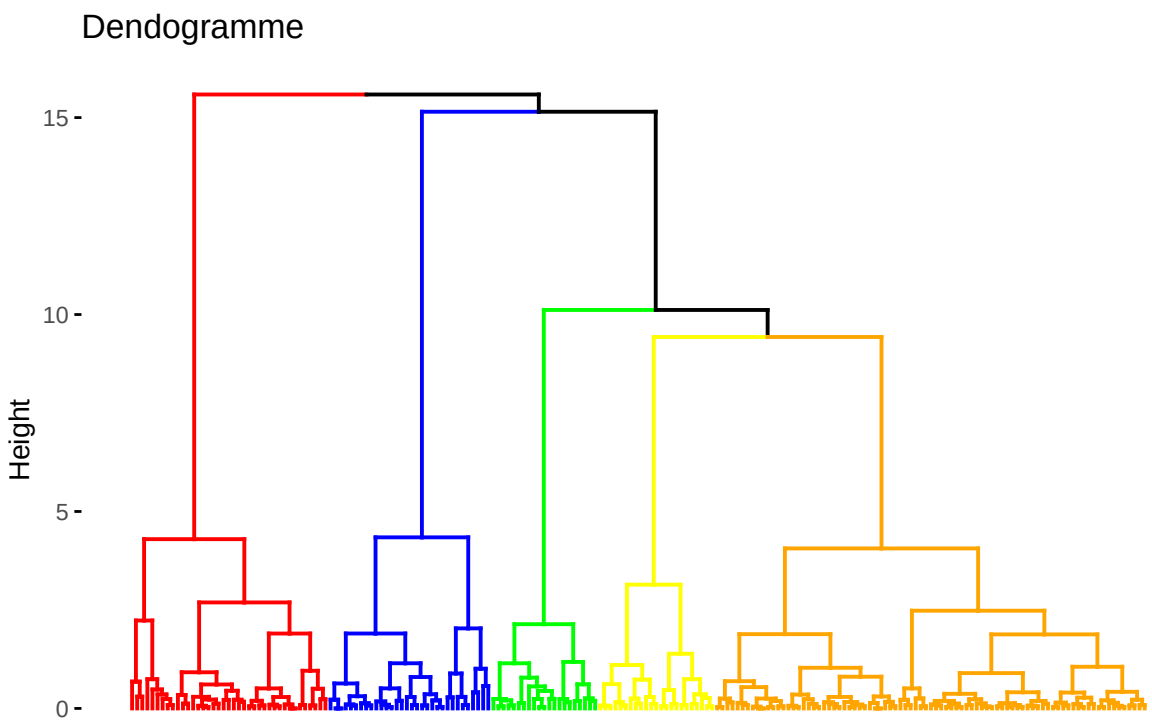
Méthode utilisé pour le clustering

Choix des revenus et scores dépenses annuels pour le regroupement des sujets et l'échelle des données ## Méthode du Dendrogramme

Utilisation du Dendrogramme pour connaitre le nombre de classes des clients

```r
# sacle :pour rendre les données à la meme échelle
custom=scale(mallCust[,c("revenus_annuels", "score_depense")], center = T, scale = T)
den<-hclust(dist(custom,method="euclidean"),method='ward.D2')
fviz_dend(den,show_labels = FALSE,main="Dendogramme",
          k = 5,
          k_colors = c("red","blue","green","yellow","orange")

          )
```

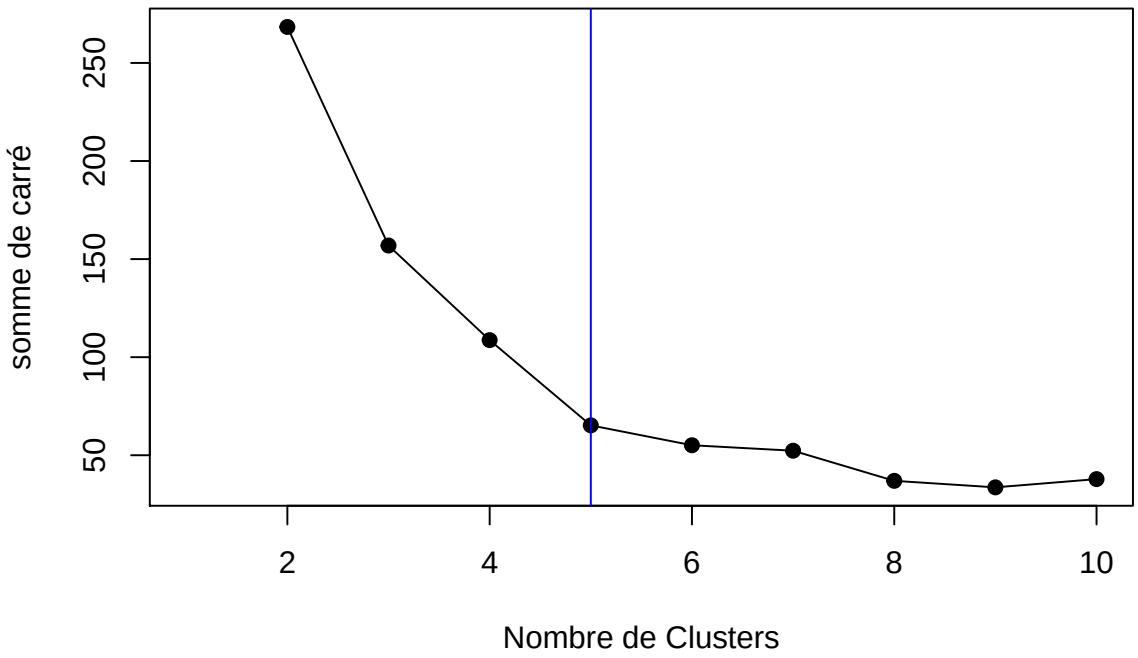


Au vu de l'arbre hiérarchique, on peut tailler l'arbre pour construire des classes.
** Pour une hauteur de coupe 10, on définit 3 classes Pour une hauteur de coupe 5, on définit 5 classes **

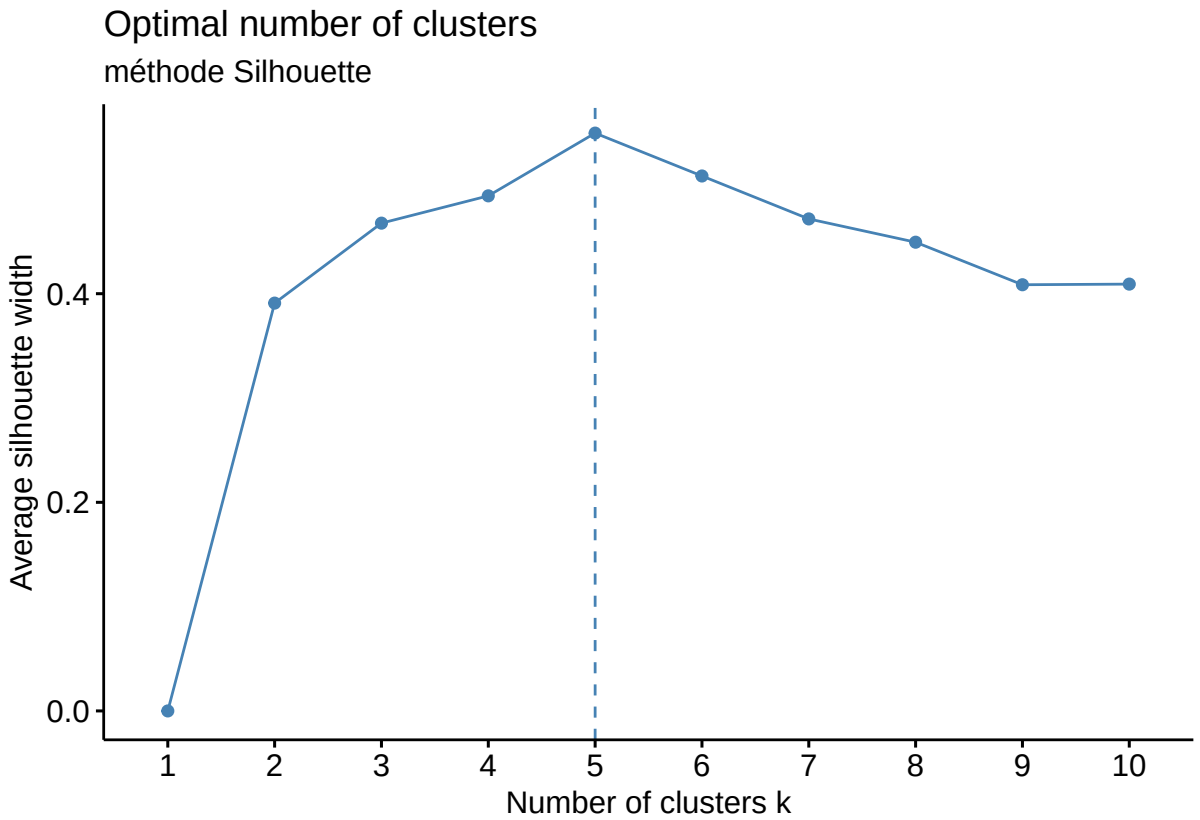
Méthode du Kmeans (Elbow,Sihouette)

Trouver le meilleur k pour le Kmeans

```
wss <- function(data, maxCluster = 10) {
    # Initialisation avec les somme de  carré
    SSw <- (nrow(data) - 1) * sum(apply(data, 2, var))
    SSw <- vector()
    for (i in 2:maxCluster) {
        SSw[i] <- sum(kmeans(data, centers = i)$withinss)# application de kmean
    }
    plot(1:maxCluster, SSw, type = "o", xlab = "Nombre de Clusters", ylab = "somme de carré", pch=19)
    abline(v=5, col="blue")
}
set.seed(100)
wss(custom)
```




```r
# méthode Silhouette
fviz_nbclust(mallCust[, 4:5], kmeans, method = "silhouette")+
  labs(subtitle = "méthode Silhouette ")
```

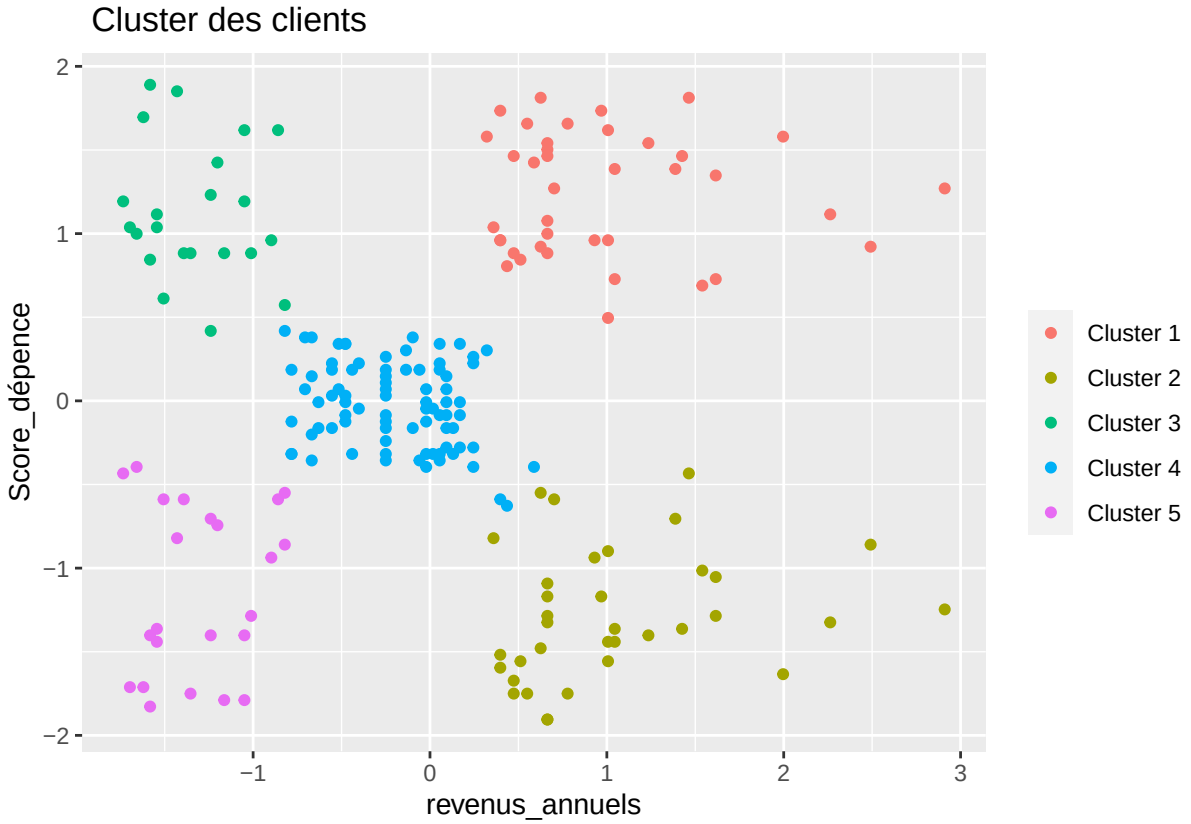


D'après les résultats de la méthode du Elbow (Coude) et Silhouette, on peut voir que le coude est un cercle de flexion k=5, donc k=5 est le nombre de groupes de clients que nous utilisons dans ce cas pour la suite .

Représentation du score de dépense en fonction du revnus selon les 5 clusters des clients

```
set.seed(111)
da=data.frame(custom)

cust.KM<-kmeans(custom,5)
ggplot(da, aes(x = revenus_annuels, y = score_depense)) +
    geom_point(stat = "identity", aes(color = as.factor(cust.KM$cluster))) +
    scale_color_discrete(name=" ",
                         breaks=c("1", "2", "3", "4", "5"),
                         labels=c("Cluster 1", "Cluster 2", "Cluster 3", "Cluster 4", "Cluster 5")) +
    ggtitle(" Cluster des clients")+
  xlab("revenus_annuels")+ylab("Score_dépence")
```



Ajoutons la colonne cluster

```
# Adding 'Cluster' column
mallCust$Cluster <- cust.KM$cluster
head(mallCust)
```

```
##   ID   Sexe Age revenus_annuels score_depense Cluster
## 1  1   Male  19              15            39       5
## 2  2   Male  21              15            81       3
## 3  3 Female  20              16             6       5
## 4  4 Female  23              16            77       3
## 5  5 Female  31              17            40       5
## 6  6 Female  22              17            76       3
```

La phase d'apprentissage consiste à trouver les paramètres du modèle optimal :

Grid : la grille, sa taille, sa forme, le type fonctionde voisinage, etc...

Rlen : le nombre de fois que l'ensemble des données sera présenté au réseau

Alpha : le pas de l'apprentissage pour contrôler la vitesse d'apprentissage

Radius : le rayon du voisinage

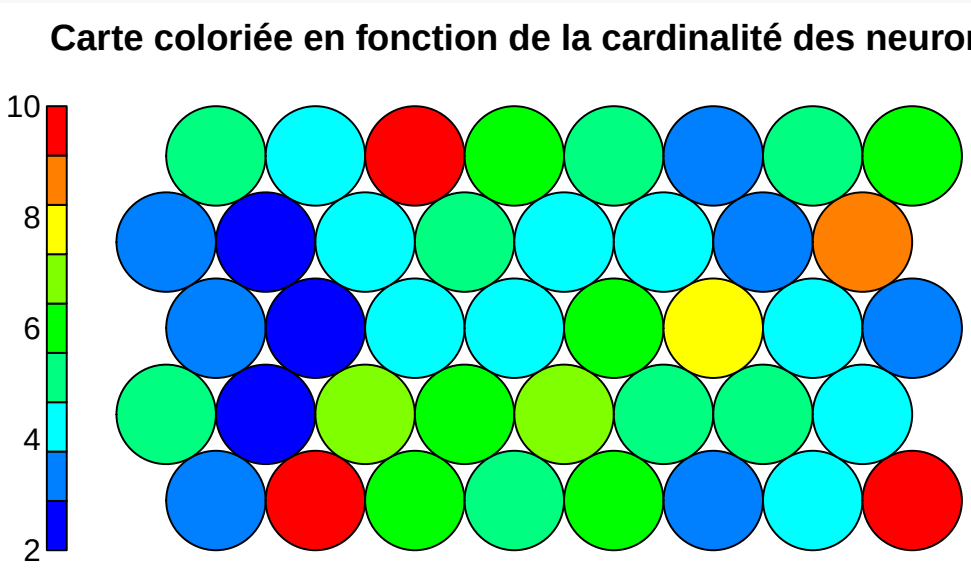
Init : les valeurs initiales des vecteurs référents

```
set.seed(111)
# Choix du type de carte et de sa taille
som.grid <- somgrid(8, 5, topo="hexagonal", neighbourhood.fct="bubble")
# Apprentissage
som.model <- som(custom, grid =som.grid, rlen=1000, alpha=c(0.05,0.01), keep.data = TRUE)
# Palette de couleur pour l'affichage des cartes
coolBlueHotRed <- function(n, alpha = 1) {rainbow(n, end=4/6, alpha=alpha)[n:1]}
```

La librairie kohonen de R présente plusieurs types de visualisation permettant de mesurer de la pertinence de la carte obtenue.

La carte coloriée en fonction de la cardinalité (le nombre d'individus capturés par un neurone) des neurones permet de mésurer la qualité de carte. La distribution de la cardinalité doit être uniforme. Des neurones présentant des cardinalités assez importantes montrent que la taille de carte est petite. De même, la présence de beaucoup de neurones avec des cardinalités nulles suggère que la carte est trop grande.

```
# Affichage des cartes
plot(som.model, type="count", main="Carte coloriée en fonction de la cardinalité des neurones", palette
```



On peut aussi représenter les neurones dans l'espace des données à l'aide des vecteurs référents. Cela permet de voir la qualité de la quantification vectorielle de l'espace d'entrée.

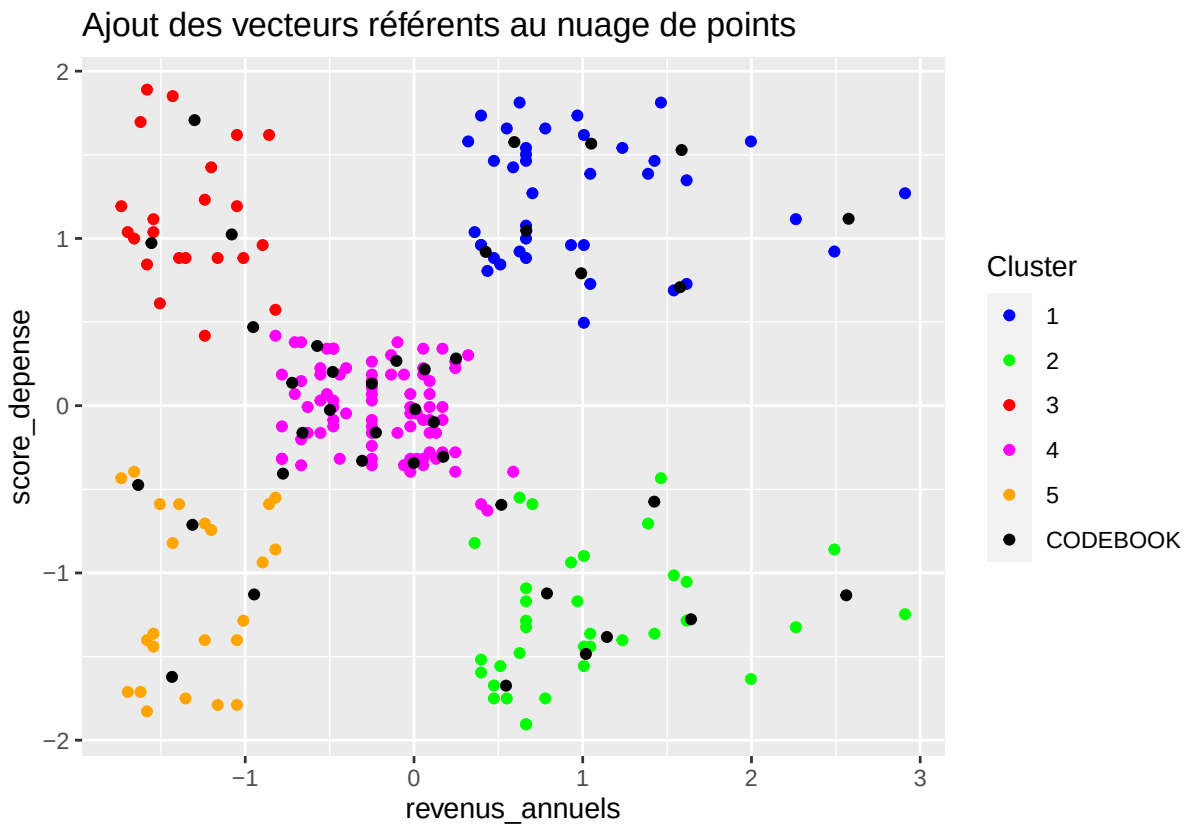
```
df = data.frame(som.model$codes)

df1=data.frame(df[,1],df[,2])
colnames(df1) = c("revenus_annuels", "score_depense")
df1$Cluster = "CODEBOOK"
```

```
dfA = data.frame(custom) %>% setNames(nm=c("revenus_annuels", "score_depense"))
dfA$Cluster = mallCust[,c(4,5,6)]$Cluster

df1 = rbind(dfA,df1)

colours = c('blue', 'green', 'red','magenta','orange', 'black')
qplot(x = revenus_annuels, y = score_depense, color = Cluster, data = df1, geom = "point") + scale_colo
```

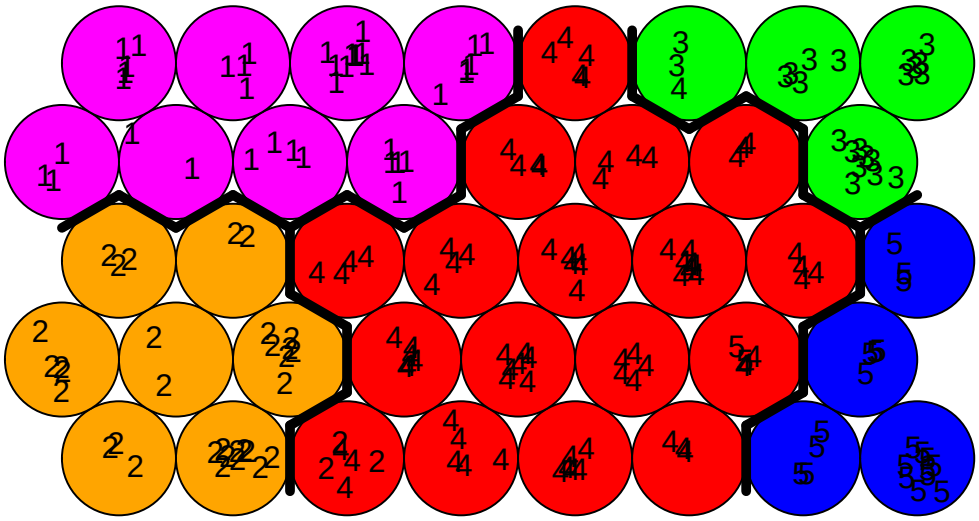


La dernière partie consiste à appliquer la classification automatique des données à partir des cartes de Kohonen. L'idée est de retrouver les cinq groupes qu'on a créés au début du l'exemple.

La décroissance de l'inertie intra-classe suggère un nombre de classe égale à 5

```
set.seed(111)
# On fixe le nombre de classes à 3
code.books = data.frame(som.model$codes) %>% setNames(nm=c("revenus", "depense"))
model.kmeans <- kmeans(x = code.books, centers = 5, nstart=100)
plot(som.model, type="mapping",
bgcol = c('blue', 'green', 'red','magenta','orange')[model.kmeans$cluster],
labels = mallCust$Cluster,
main = "Les clusters sur la carte de kohonen")
add.cluster.boundaries(som.model, clustering = model.kmeans$cluster)
```

Les clusters sur la carte de kohonen



Les cartes topologiques de Kohonen constituent un outil puissant de réduction de dimension et de classification automatique. Elles peuvent être vues comme :

Une extension non linéaire de l'ACP (Analyse en Composantes Principales) dans le cadre d'une réduction de dimension Une extension non linéaire de K-means dans le cadre d'une classification automatique

On retrouve bien les cinq groupes définis au début l'exemple. Les neurones fournissent une classification plus fine que les k-means.

Information pour le cluster 1

```
#summary(cluster1)
```

```
   Gender       Age         Annual_Income   Spending_Score
Female:14   Min.   :19.00   Min.   :15.0   Min.   : 3.00
Male  : 9   1st Qu.:35.50   1st Qu.:19.5   1st Qu.: 9.50
            Median :46.00   Median :25.0   Median :17.00
            Mean   :45.22   Mean   :26.3   Mean   :20.91
            3rd Qu.:53.50   3rd Qu.:33.0   3rd Qu.:33.50
            Max.   :67.00   Max.   :39.0   Max.   :40.00
```

Information pour le cluster 2

```
#summary(cluster2)
```

Gender	Age	Annual_Income	Spending_Score
Female:48	Min. :18.00	Min. :39.0	Min. :34.00
Male :33	1st Qu.:27.00	1st Qu.:48.0	1st Qu.:44.00
	Median :46.00	Median :54.0	Median :50.00
	Mean :42.72	Mean :55.3	Mean :49.52
	3rd Qu.:54.00	3rd Qu.:62.0	3rd Qu.:55.00
	Max. :70.00	Max. :76.0	Max. :61.00

Figure 1: résumé du cluster 3.

Gender	Age	Annual_Income	Spending_Score
Female:16	Min. :19.00	Min. : 70.0	Min. : 1.00
Male :19	1st Qu.:34.00	1st Qu.: 77.5	1st Qu.:10.00
	Median :42.00	Median : 85.0	Median :16.00
	Mean :41.11	Mean : 88.2	Mean :17.11
	3rd Qu.:47.50	3rd Qu.: 97.5	3rd Qu.:23.50
	Max. :59.00	Max. :137.0	Max. :39.00

Information pour le cluster 3

```
#summary(cluster3)
```

Information pour le cluster 4

```
#summary(cluster4)
```

Gender	Age	Annual_Income	Spending_Score
Female:13	Min. :18.00	Min. :15.00	Min. :61.00
Male : 9	1st Qu.:21.25	1st Qu.:19.25	1st Qu.:73.00
	Median :23.50	Median :24.50	Median :77.00
	Mean :25.27	Mean :25.73	Mean :79.36
	3rd Qu.:29.75	3rd Qu.:32.25	3rd Qu.:85.75
	Max. :35.00	Max. :39.00	Max. :99.00

Information pour le cluster 5

```
#summary(cluster5)
```

Gender	Age	Annual_Income	Spending_Score
Female:21	Min. :27.00	Min. : 69.00	Min. :63.00
Male :18	1st Qu.:30.00	1st Qu.: 75.50	1st Qu.:74.50
	Median :32.00	Median : 79.00	Median :83.00
	Mean :32.69	Mean : 86.54	Mean :82.13
	3rd Qu.:35.50	3rd Qu.: 95.00	3rd Qu.:90.00
	Max. :40.00	Max. :137.00	Max. :97.00

Figure 2: résumé du cluster 5.

Interprétation pour le groupe/segment de clients :

Cluster 1. Les clients ayant un revenu annuel élevé mais un score de dépenses faible.

Cluster 2. Clients ayant un revenu annuel moyen et un score moyen en matière de dépenses.

Cluster 3. Clients ayant un faible revenu annuel et un faible niveau de dépenses.

Cluster 4. Clients ayant un revenu annuel faible mais un niveau de dépenses élevé.

Cluster 5. Clients ayant un revenu annuel élevé et un score élevé en matière de dépenses.

les données fournies ci-dessus, nous pourrions prendre un résumé d'analyse qui peut être utilisé pour le plan de marketing comme suit :

J'ai pu constater que le pourcentage de clients féminins (56%) est légèrement supérieur à celui des clients masculins (44%), ce qui nous a permis de cibler davantage les clients masculins pour les campagnes de marketing ou les promotions que les clients féminins, même si le pourcentage différent n'est pas trop important. Dans ce cas, nous pouvons également choisir judicieusement la cible des clients masculins en combinant des facteurs liés à leur âge et à leur groupe.

Le centre commercial peut mener des campagnes de marketing ou des programmes de fidélisation auprès des clients des groupes 5 et 4, qui sont des clients ayant un niveau de dépenses élevé, en particulier les clients âgés de 20 et 30 ans, afin de conserver cette clientèle et d'augmenter les possibilités de vente.

Les groupes 1 et 3 ont en général un faible niveau de dépenses, malgré leur niveau de revenu, et les clients sont généralement âgés de plus de 40 ans. Avec ces données, nous pourrions envisager de rechercher et d'ajouter certaines marques qui sont populaires parmi les clients de ces âges, et de mener des campagnes pour les cibler avec les bons produits.

Comme nous pouvons le voir à partir des données ci-dessus, le groupe 2 a un score de dépenses moyen, afin d'augmenter les ventes dans ce groupe de clients, nous pourrions leur donner quelques promotions pour les encourager à acheter plus de produits.

11.9 Application des CNN

Convolutional-Neural-Network-Image-Classification

April 12, 2020

1 DIARRASSOUBA SAKARIA

1.1 TP réalisé pour l'obtention d'un Certificat en Réseau de neurone convolutif en ligne sur Udemy

L'objectif de ce TP consistait à construire un modèle de Réseau de Neurone Convolutif pour prédire de quel animal s'agit il sur une base de données contenant les images de chiens et les image de chats.

[1]:
```
# Importation de bibliothèques de CNN
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.optimizers import RMSprop
from keras.layers import  Dropout, Flatten, Dense, GlobalMaxPooling2D, Conv2D,␣
↪MaxPooling2D
from keras.callbacks import CSVLogger
from livelossplot import PlotLossesKeras
```

```
Using TensorFlow backend.
```

[2]:
```
# les Hyperparamètres
IMAGE_SIZE = 150
IMAGE_WIDTH, IMAGE_HEIGHT = IMAGE_SIZE, IMAGE_SIZE
EPOCHS = 20
BATCH_SIZE = 32

input_shape = (IMAGE_WIDTH, IMAGE_HEIGHT, 3)
```

construction du modele inspirée de https://towardsdatascience.com/image-classifier-cats-vs-dogs-with-convolutional-neural-networks-cnns-and-google-colabs-4e9af21ae7a8

[3]:
```
# initialisation du modele
classifier = Sequential()
```

[4]:
```
# Step 1 - construction de la première couche de  Convolution

classifier.add(Conv2D(filters=32,kernel_size=3,strides=1,border_mode='same',␣
 ↪input_shape=input_shape,activation="relu"))
# construction de la première couche de max-pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

```
/Users/vw12/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3:
UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(filters=32,
kernel_size=3, strides=1, input_shape=(150, 150,…, activation="relu",
padding="same")`
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[5]:
```
# Ajout de la seconde couche de convolution et de max-pooling
classifier.add(Conv2D(64, 3, 1, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

```
/Users/vw12/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(64, (3, 1),
activation="relu")`
```

[6]:
```
# Ajout de la troisième  couche de convolution et de max-pooling
classifier.add(Conv2D(128, 3, 1, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

```
/Users/vw12/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(128, (3, 1),
activation="relu")`
```

[7]:
```
# Ajout de la quatrieme couche de convolution et de max-pooling
classifier.add(Conv2D(256, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

[8]:
```
# Step 3 - Flattening

classifier.add(Flatten())

classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dropout(0.5)) # qui premet de desactiver certains neurones avec␣
 ↪certaines proba

classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dropout(0.5)) # qui premet de desactiver certains neurones avec␣
 ↪certaines proba
```

```
classifier.add(Dense(units = 1, activation = 'sigmoid')) # car on a 2 classes␣
 ↪chien et chat
```

[9]:
```
# entrainer CNN sur les images avec le pas d'apprentissage =0.0001
classifier.compile(loss='binary_crossentropy',
            optimizer=RMSprop(lr=0.0001),
            metrics=['accuracy'])
```

Pour éviter le surapprentissage car nous avons 8000 images, on va utiliser un fonction **ImageDataGenerator** dans **kers documentation** pour générer de nouvelles images afin d'entrainer de reseau et aussi évaluer la performance de sur le jeu de test

[10]:
```
# augmentation des données
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)
prediction_datagen = ImageDataGenerator(rescale=1./255)
```

[11]:
```
# preparation des données
training_set = train_datagen.flow_from_directory(
        'dataset/training_set',
        target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
        batch_size=BATCH_SIZE,
        class_mode="binary")

test_set = test_datagen.flow_from_directory(
        'dataset/test_set',
        target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
        batch_size=BATCH_SIZE,
        class_mode="binary")

prediction_set= prediction_datagen.flow_from_directory(
        'dataset/prediction',
        target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
        batch_size=BATCH_SIZE,
        class_mode='binary')
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
Found 13 images belonging to 1 classes.
```

[14]:
```
# entrainement du modèle
classifier.fit_generator(
        training_set,
        steps_per_epoch=250,
        epochs=EPOCHS,
        validation_data=test_set,
        validation_steps=63)
```

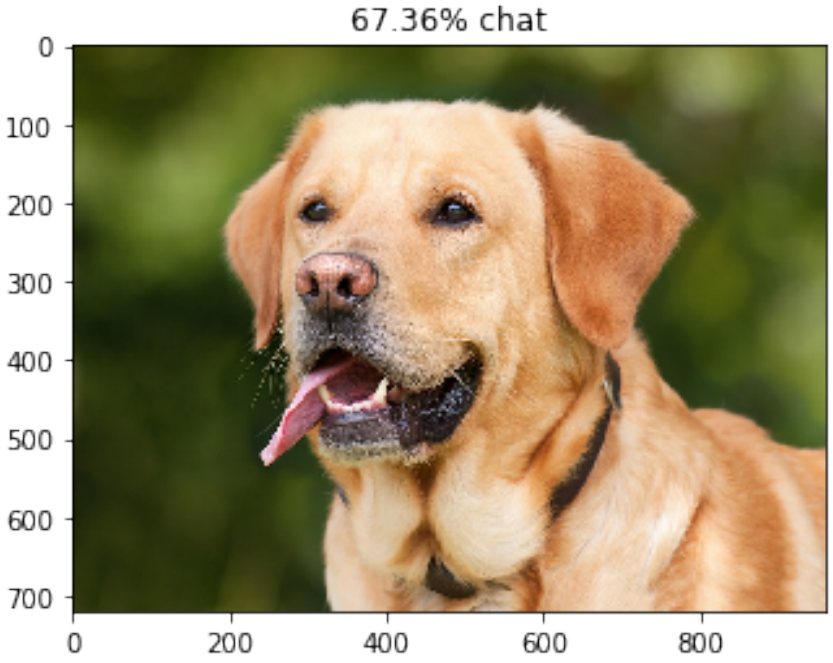
```
Epoch 1/20
250/250 [==============================] - 393s 2s/step - loss: 0.6899 -
accuracy: 0.5326 - val_loss: 0.6472 - val_accuracy: 0.5335
Epoch 2/20
250/250 [==============================] - 401s 2s/step - loss: 0.6554 -
accuracy: 0.6148 - val_loss: 0.5550 - val_accuracy: 0.6715
Epoch 3/20
250/250 [==============================] - 393s 2s/step - loss: 0.6195 -
accuracy: 0.6590 - val_loss: 0.4504 - val_accuracy: 0.6910
Epoch 4/20
250/250 [==============================] - 394s 2s/step - loss: 0.5815 -
accuracy: 0.6942 - val_loss: 0.5714 - val_accuracy: 0.7280
Epoch 5/20
250/250 [==============================] - 395s 2s/step - loss: 0.5590 -
accuracy: 0.7144 - val_loss: 0.3585 - val_accuracy: 0.7425
Epoch 6/20
250/250 [==============================] - 393s 2s/step - loss: 0.5287 -
accuracy: 0.7376 - val_loss: 0.8286 - val_accuracy: 0.7265
Epoch 7/20
250/250 [==============================] - 397s 2s/step - loss: 0.5092 -
accuracy: 0.7481 - val_loss: 0.4559 - val_accuracy: 0.7730
Epoch 8/20
250/250 [==============================] - 401s 2s/step - loss: 0.4846 -
accuracy: 0.7710 - val_loss: 0.2286 - val_accuracy: 0.7725
Epoch 9/20
250/250 [==============================] - 397s 2s/step - loss: 0.4727 -
accuracy: 0.7771 - val_loss: 0.5106 - val_accuracy: 0.7825
Epoch 10/20
250/250 [==============================] - 403s 2s/step - loss: 0.4558 -
accuracy: 0.7883 - val_loss: 0.3695 - val_accuracy: 0.7640
Epoch 11/20
250/250 [==============================] - 410s 2s/step - loss: 0.4439 -
accuracy: 0.7935 - val_loss: 0.5947 - val_accuracy: 0.7820
Epoch 12/20
250/250 [==============================] - 403s 2s/step - loss: 0.4324 -
accuracy: 0.8033 - val_loss: 0.2470 - val_accuracy: 0.8200
Epoch 13/20
250/250 [==============================] - 405s 2s/step - loss: 0.4151 -
accuracy: 0.8109 - val_loss: 0.2248 - val_accuracy: 0.8100
```

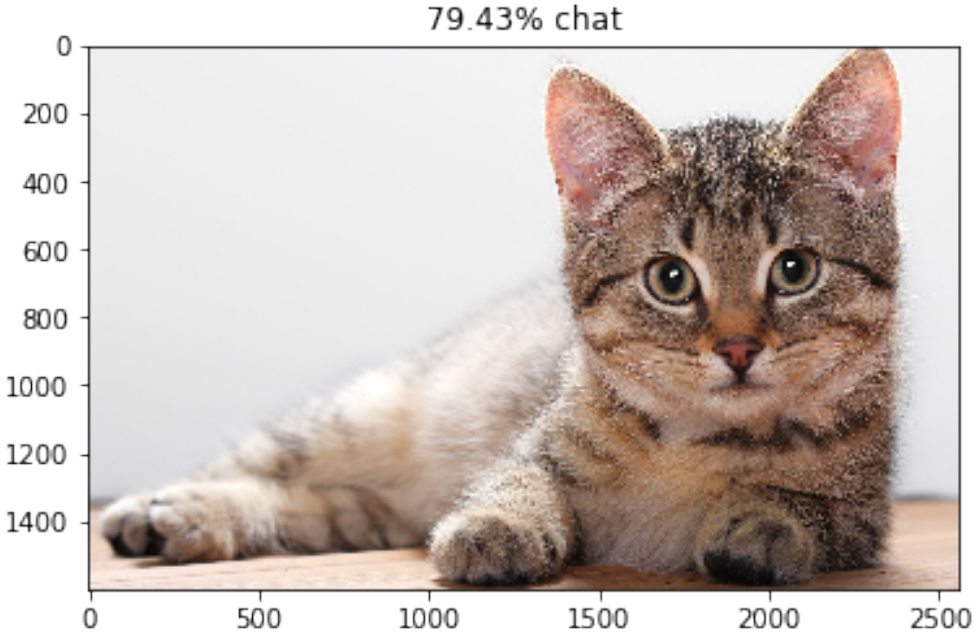
```
Epoch 14/20
250/250 [==============================] - 413s 2s/step - loss: 0.4105 -
accuracy: 0.8148 - val_loss: 0.2898 - val_accuracy: 0.8210
Epoch 15/20
250/250 [==============================] - 413s 2s/step - loss: 0.4004 -
accuracy: 0.8195 - val_loss: 0.3689 - val_accuracy: 0.8020
Epoch 16/20
250/250 [==============================] - 409s 2s/step - loss: 0.3841 -
accuracy: 0.8275 - val_loss: 0.2588 - val_accuracy: 0.8260
Epoch 17/20
250/250 [==============================] - 409s 2s/step - loss: 0.3799 -
accuracy: 0.8286 - val_loss: 0.3288 - val_accuracy: 0.8325
Epoch 18/20
250/250 [==============================] - 406s 2s/step - loss: 0.3719 -
accuracy: 0.8366 - val_loss: 0.3923 - val_accuracy: 0.8280
Epoch 19/20
250/250 [==============================] - 408s 2s/step - loss: 0.3608 -
accuracy: 0.8459 - val_loss: 0.2761 - val_accuracy: 0.8335
Epoch 20/20
250/250 [==============================] - 424s 2s/step - loss: 0.3560 -
accuracy: 0.8456 - val_loss: 0.5815 - val_accuracy: 0.8190
```

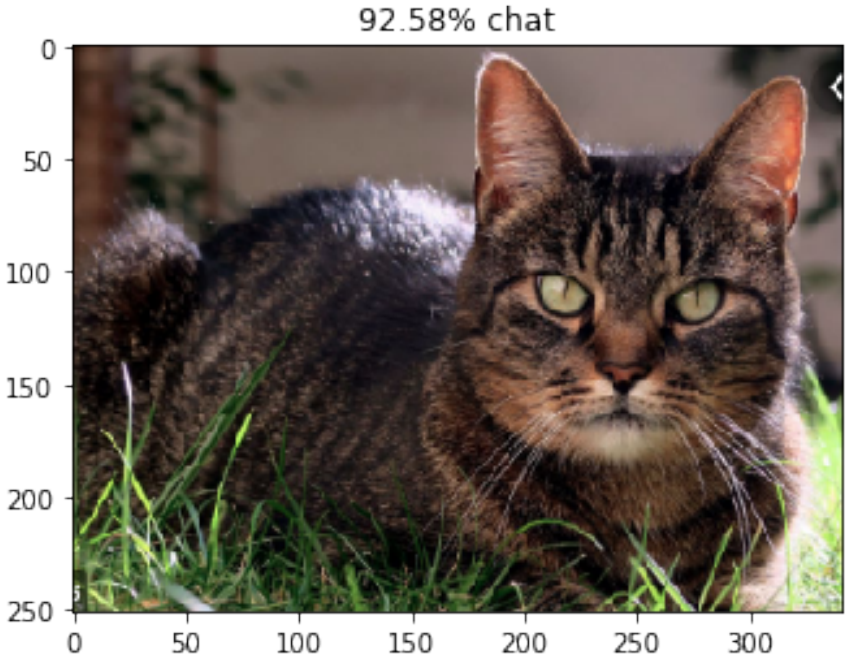
[14]: `<keras.callbacks.callbacks.History at 0x13ff60990>`

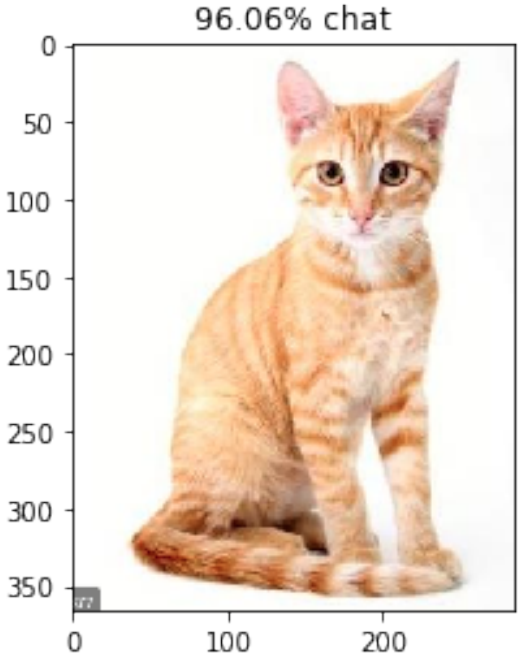
La phase de la prédiction: Dans la base single_prédiction, on 8 images de chats et 5 images de chiens à chaque fois on présente au modéle une image et il nous dit si c'est un chat ou chien

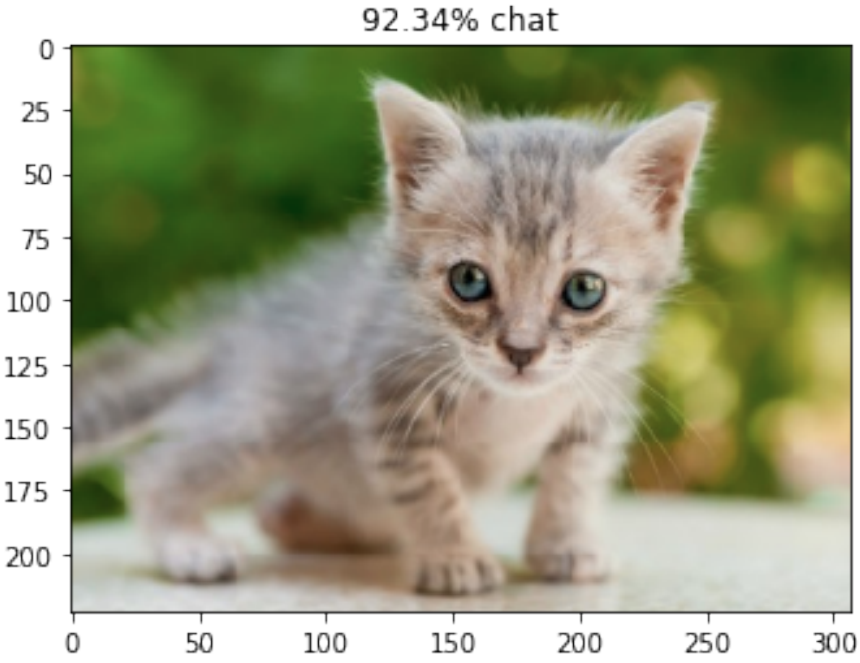
[25]:
```
probabilities = classifier.predict_generator(prediction_set,␣
 ↪len(prediction_set))
for index, probability in enumerate(probabilities):
    image_path = 'dataset/prediction' + "/" +prediction_set.filenames[index]
    img = mpimg.imread(image_path)
    plt.imshow(img)
    if probability > 0.5:
        plt.title("%.2f" % (probability[0]*100) + "% chien")
    else:
        plt.title("%.2f" % ((1-probability[0])*100) + "% chat")
    plt.show()
```


67.36% chat
0
100
200
300
400
500
600
700
0
200
400
600
800


79.43% chat
0
200
400
600
800
1000
1200
1400
0
500
1000
1500
2000
2500

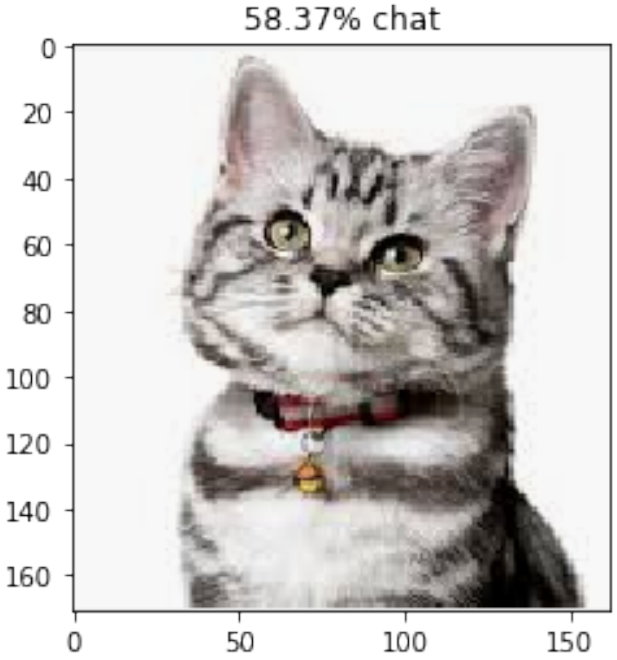

92.58% chat
0
50
100
150
200
250
0
50
100
150
200
250
300

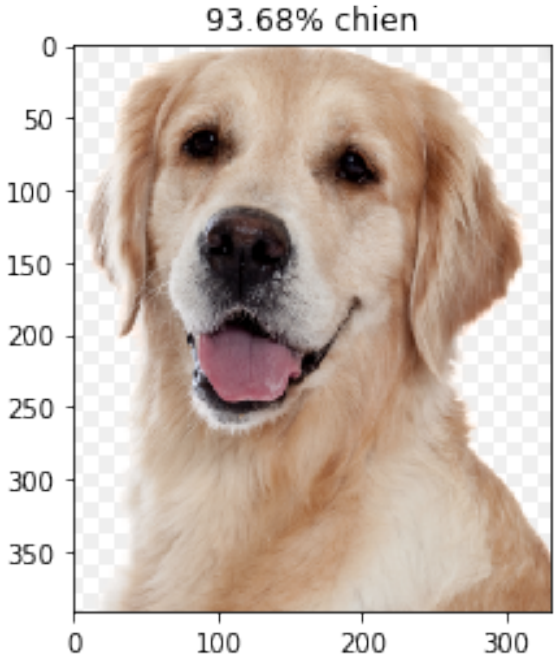

96.06% chat
0
50
100
150
200
250
300
350
0
100
200

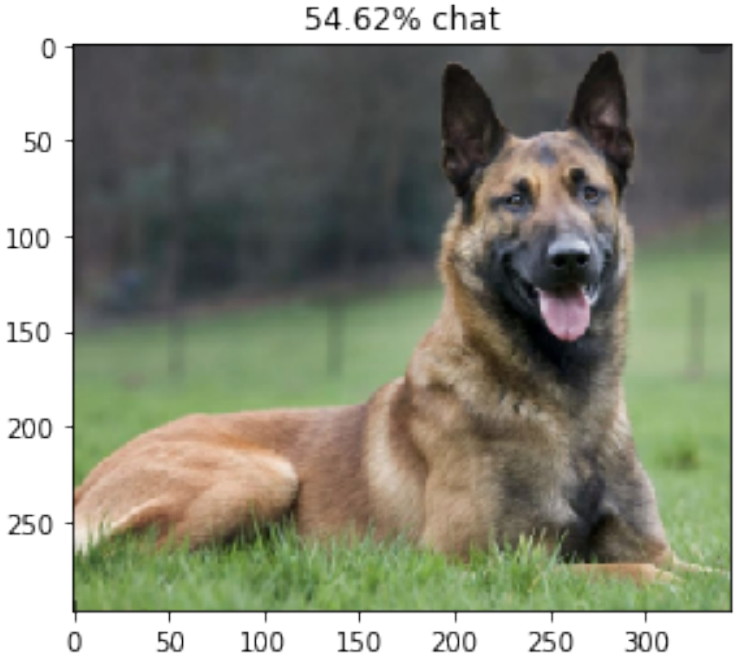

92.34% chat
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300

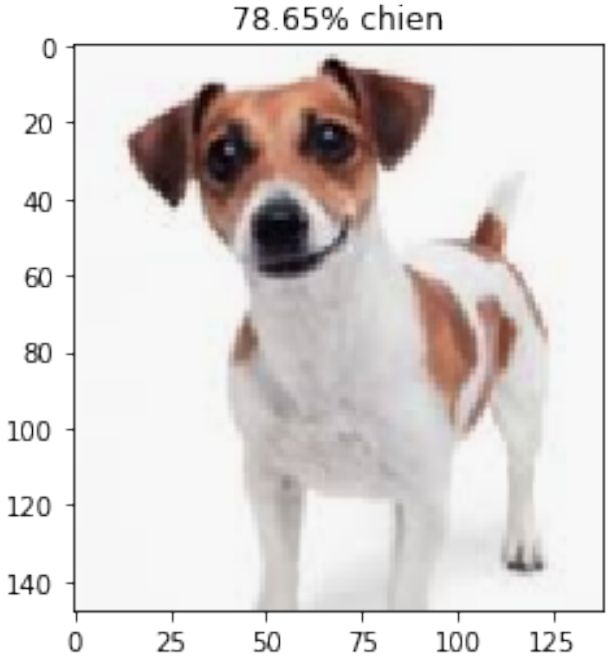

85.08% chien
0
25
50
75
100
125
150
175
0
50
100

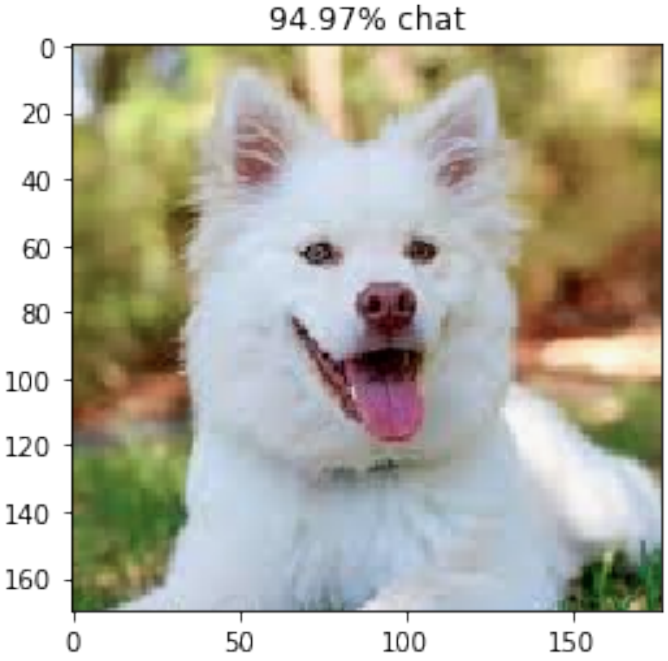

96.05% chat
0
20
40
60
80
100
120
140
0
50
100
150
200

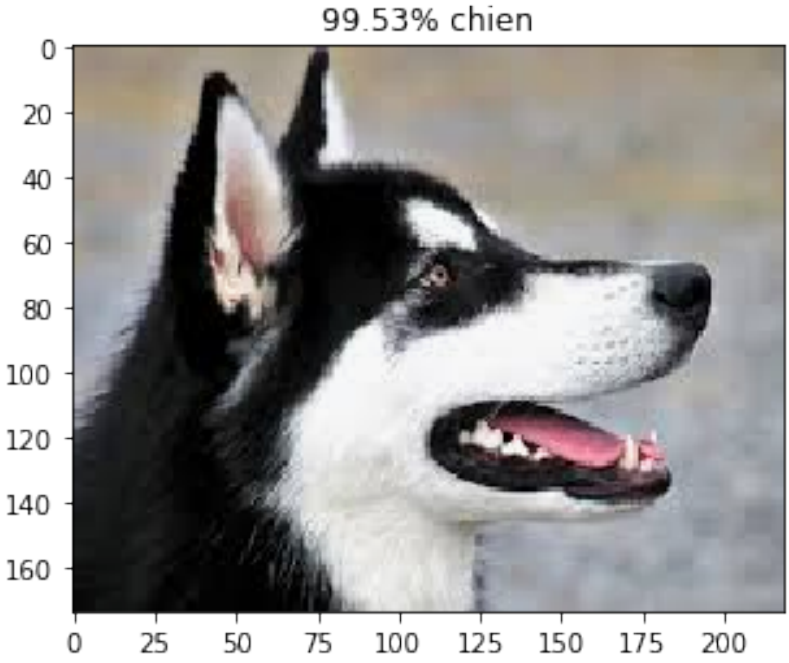

58.37% chat
0
20
40
60
80
100
120
140
160
0
50
100
150


93.68% chien
0
50
100
150
200
250
300
350
0
100
200
300


54.62% chat
0
50
100
150
200
250
0
50
100
150
200
250
300


78.65% chien
0
20
40
60
80
100
120
140
0
25
50
75
100
125


94.97% chat
0
20
40
60
80
100
120
140
160
0
50
100
150


99.53% chien
0
20
40
60
80
100
120
140
160
0
25
50
75
100
125
150
175
200

[ ]: