Rapport de Projet:
Projet Annuel 3 Big Data

RHERMINI Idriss

31/08/2020

Table des matières

Introduction

Sujet

L'objectif de ce projet est d'implémenter et utiliser des modèles et algorithmes simples relatifs au Machine Learning, combiner le tout dans un cas pratique réel. Dans mon cas, j'ai choisi la classification d'images.

Le but est de chercher à classer une image et dire si celle-ci appartient au genre CHAT, CHIEN ou OISEAU.

Ce rapport a pour but de rendre compte de l'avancée du projet sur la création d'un dataset, puis implémenter un neurone unique.

Pour la classification, je vais utiliser la méthode de Rosenblatt pour l'apprentissage et essayer de voir comment les paramètres influent sur mes résultats.

Analyse d'application

Cette section a pour but de présenter les structures et ainsi les fonctions principales de ce projet. Vous y trouverez aussi le modèle et les choix d'implémentation.

Structures principales

Structure Neuron

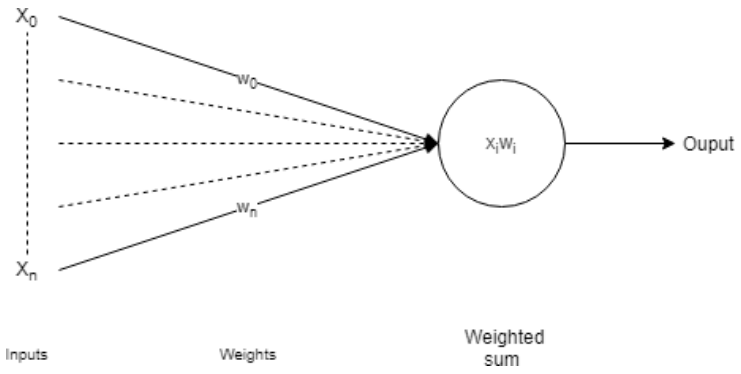
Ci-dessous la structure qui est le cœur de notre projet, le neurone.

Le neurone permet de faire de simples calculs, Il va faire une somme pondérée de ses entrées. Dans cette forme basique, nous avons un neurone dans sa version linéaire. Pour cesser ce calcul linéaire, il faut faire passer la sortie du neurone dans une fonction d'activation non linéaire tel que $tanh(X)$.

```
Type def  struct neuron
{
   double * inputs ;
```

```
    double * weights ;
    double output ;
    int nbInputs ;
    int type Activation ;
    double bias ;
} Neuron ;
```

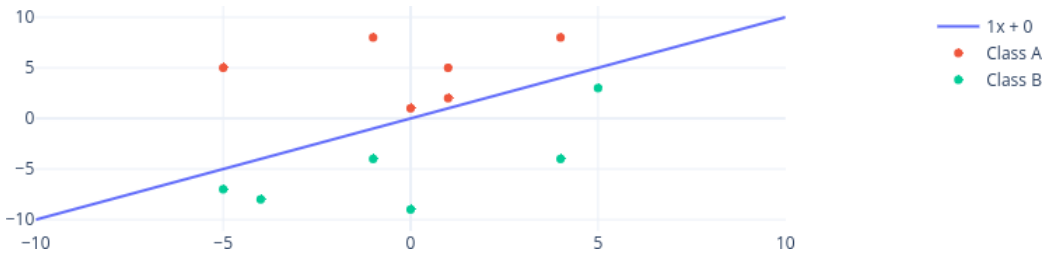
Les *inputs* sont les entrées. Pour une position donnée on l'écrit x_i.
Les *weights* sont les poids. Pour une position donnée on l'écrit w_i.



La somme pondérée des entrées du neurone s'écrit :

- $out = Sign(\sum_{i=0}^{n} w_i x_i)$

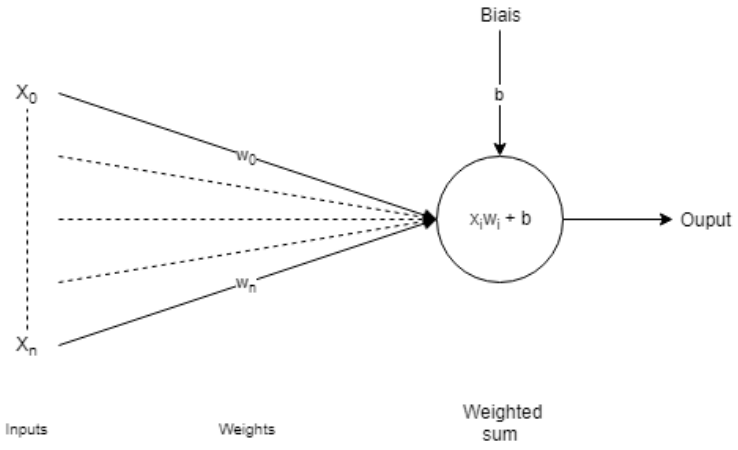
On se retrouve avec fonction linéaire de forme $f(X) = aX + b$ avec $b = 0$. On est donc capable de tracer une droite passant par l'origine.



Si mes données sont linéairement séparables par l'origine, alors ce modèle est suffisant. Cependant, si nous avons un point de la classe A qui passe sous la droite $f(X) = 1X + 0$, alors je ne suis plus en mesure de classer mes données correctement. Il faut donc modifier l'ordonnée à l'origine b. Une autre solution à

ce problème aurait été de normaliser les données pour les centrer afin de se retrouver avec $b = 0$.

On rajoute donc un biais à ma somme pondérée afin de générer l'ordonnée à l'origine de ma droite qui va séparer nos données.

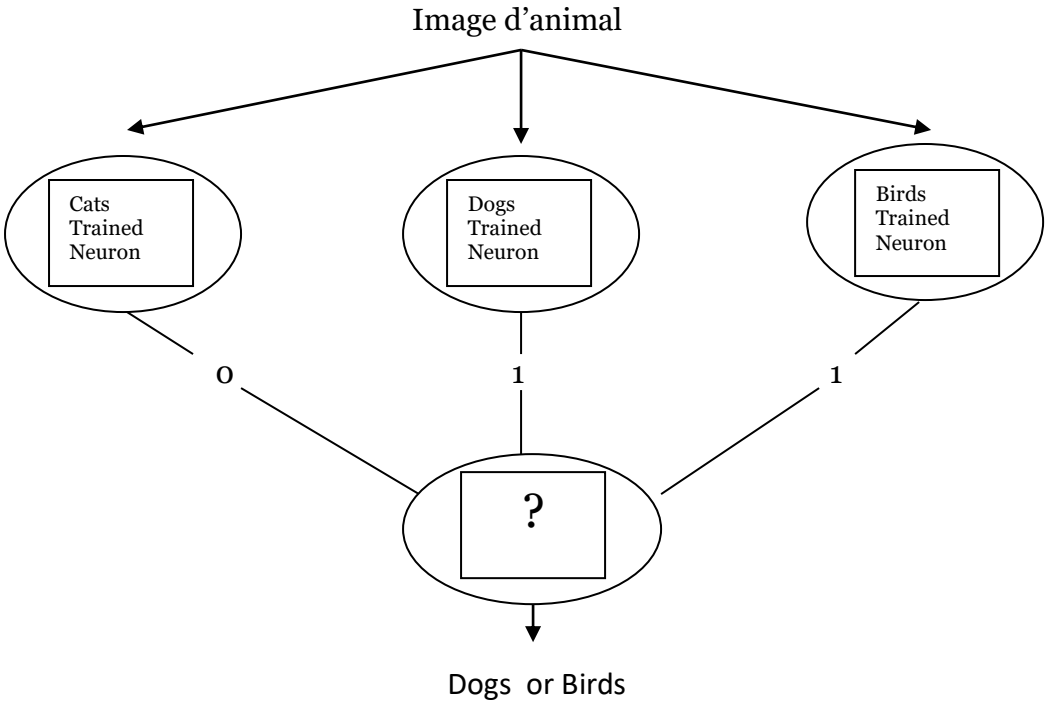


Dans le cas de mon projet, j'ai choisi de réduire le nombre de pixels de l'image en prenant que 3 canaux d'un pixel tout en conservant sa valeur en niveau de gris. J'expérimenterai, plus tard, les couleurs, une fois convaincu du bon fonctionnement du neurone pour des tâches simples. Les pixels ont une valeur entre 0 et 255, imaginons une image totalement blanche (R :255 G :255 B :255). Que se passerait-il si je fais la somme pondérée de la valeur 255 ? Je risquerais d'avoir une valeur qui va exploser en infini. Pour remédier donc à ce problème, je normalise mes valeurs d'entrées, je répartis mes valeurs entre 0 et 1, c'est une normalisation min-max.

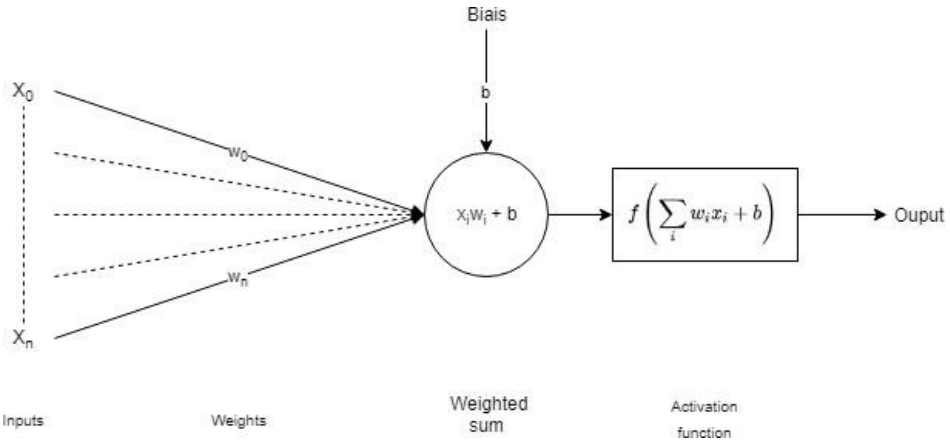
$$z = \frac{x - min(x)}{max(x) - min(x)}$$

On se retrouve avec des valeurs plus petites sans pour autant perdre d'informations, cependant je n'ai toujours pas résolu la possibilité de tendre vers l'infini lors de ma somme pondérée. En effet, je ne connais pas les valeurs possibles des poids. Il est possible qu'un pixel ait une importance infime dans le rôle de la détermination d'une image ce qui risque de créer un calcul qui tend vers l'infiniment petit. De même pour une valeur importante qui risque de produire une sortie très grande. Pour pallier ce problème, je veux borner cette information afin de ne pas minimiser l'importance des valeurs extrêmes.

Le rôle de cette fonction d'activation a pour but de casser cette linéarité. La première fonction d'activation utilisée est celle par palier. On dit que si la sortie du neurone est positive alors on peut affirmer que l'image appartient à la classe recherchée. C'est donc une classification binaire, "vrai" ou "faux". Mais est-ce applicable pour une classification de 3 classes ? Impossible pour le neurone de me dire son avis sur 3 classes alors qu'il est binaire. Même si nous faisons 3 neurones indépendants que j'entraine sur 3 classes séparément, que se passe-t-il si 2 neurones sortent "vrais" ? Impossible de prendre une décision sur des valeurs aussi brutes. Il me faut un intervalle de valeur plus large.



C'est pour cela que j'introduis la fonction sigmoïde (1 /(1-ex)) ou la tangente hyperbolique (tanh(x)). La première solution va nous garantir une sortie comprise entre [0, 1] alors que la seconde entre [−1, 1]. Dans mon cas j'ai choisi la fonction tanh(x) pour conserver les valeurs négatives afin d'être plus sévère sur mon futur apprentissage sur les entrées peu importantes.



Règle de Rosenblat

Maintenant que je sais comment fonctionne un neurone, je peux modifier ses paramètres pour avoir une sortie qui reflète ce qu'on attend de lui. C'est à dire, prédire l'appartenance d'une image à une classe. Pour avoir la bonne sortie, il faut jouer avec les poids du neurone. On verra par la suite que d'autres paramètres rentrent en jeux, mais risquent de complexifier la tâche.

Pour modifier les poids du neurone il faut regarder la marge d'erreur ou la distance entre ce qu'on s'attendait à avoir et le résultat obtenu avec certains poids.

Je dois créer un dataset d'image *XTrain* ainsi que *YTrain* les valeurs espérées suite à la prédiction du neurone. On modifie le poids du neurone si la sortie n'est pas identique à celle espérée grâce à cette formule :

$$w_i \leftarrow w_i + a(y^k - g(x^k))x_i^k$$

- w_i : le poids d'un pixel x à la position i.
- a : le pas d'apprentissage fixé par l'utilisateur.
- y_i^k : la sortie attendue pour l'image k au pixel i.
- $g(x_i^k)$: la sortie calculée par le neurone pour l'image k au pixel i.
- x_i^k : la valeur du pixel de l'image k à la position i.

Je suis la règle de Rosenblatt qui permet de mettre à jour les poids d'un neurone en itérant sur les images qu'on lui fournit.

Maintenant que la théorie est posée, il serait intéressant de modifier les paramètres pour observer leur comportement sur un neurone.

Structure NeuralNet

```
typedef struct layer
{
    Neuron** neurons;
    int nbNeurons;
} Layer;

typedef struct neural Net
{
    double * inputs;
    Layer ** Layers; // Matrix of neurons
    int nb Layers;
    int* sizeLayers;
} Neural Net;
```

Pour plus tard, j'ai déjà entamé la structure possible qui servira à la construction d'un réseau de neurone. A l'instant présent cette structure semble robuste, du moins pour le réseau de neuron à plusieurs couches. Le principe est simple, un réseau de neurone possède des couches de neurones, cette couche est appelée **Layers**. Ce qui n'est finalement qu'une simple liste avec un int pour pouvoir itérer sur les élements de cette liste.

Lors de la construction d'un NeuralNet, je définis ses nombres de couches et de neurones par couche. Une fois les paramètres remplis, la liaison des couches est automatique. En effet, chaque couche prend une liste neurone et chaque neurone prend une liste de inputs, quand je construis le réseau, je lie les output de tous les neurones de la couche précédente pour en faire les inputs de la couche suivante.

Lors de la construction, les poids sont aussi aléatoires entre [−1, 1] pour ne pas influencer les résultats. Une fois le réseau construit, je fais une première prédiction pour alimenter les inputs et output de chaque neurone de chaque couche, sinon j'aurais des pointeurs nuls.

Les biais et améliorations

Construction du data set

Ma première tâche a été de se créer mon propre jeu de données pour nourrir mon neurone. J'ai donc 2 possibilités pour trouver des images des animaux :

- Trouver les images sur Google.
- Regarder des vidéos sur Youtube.

J'ai fait un choix sur le type d'image, le format et son contenu. En premier lieux je me suis focalisé sur maximum 3 types d'animaux par classe (CATS, DOGS, BIRDS).

Paramètre d'apprentissage

La taille du data set

J'ai pris la décision de commencer par environ 1.000 images par classe soit un total de 3.000 images. Si je dois respecter la proportion images par nombre de pixel, je dois utiliser des images avec 50 pixels. Tous ces chiffres sont strictement spéculatifs et ne sont pas déterministes, ils peuvent changer. En effet, pour un humain, il est assez compliqué de différencier un type d'animaux sur une image ne contenant que 45 pixels. Peut-être que la machine, oui ? Mais à quel prix ?

L'époque

L'époque (Epoch) est le nombre d'itération que je vais effectuer sur chaque images dans mon dataset. Il joue un rôle crucial sur l'apprentissage de l'algorithme. Une Epoch trop faible rendrait l'algorithme peu performant, alors qu'un trop grand mènerait à un apprentissage par coeur du dataset et risquerait de mal représenter la réalité lors des tests inconnus.

Le pas d'apprentissage

Le pas d'apprentissage nommé α est une valeur qui va nous permettre de gérer la finesse de notre apprentissage. Si l'on prend une petite valeur, je vais me rapprocher de façon lente vers notre objectif inconnu. Il faudra donc augmenter le nombre d'époque pour être sûr d'atteindre mon objectif. Si le pas d'apprentissage est trop petit, certes je serais précis dans ma recherche des poids, mais cela me demanderait beaucoup plus de temps pour trouver l'objectif.

Si je prends un pas d'apprentissage trop élevé, je me rapproche plus rapidement de mon objectif, par contre la probabilité d'arriver proche de mon objectif est faible. Il serait donc impossible d'être précis.

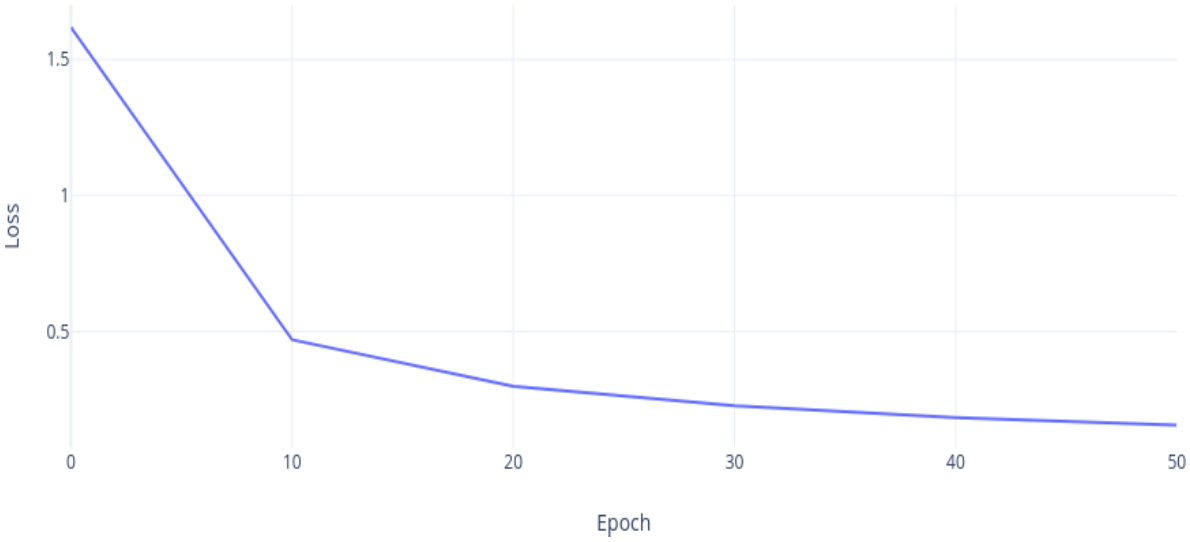
L'objectif ici est de jouer avec le pas d'apprentissage selon le nombre d'époque ou d'une autre métrique. C'est à dire avoir un pas d'apprentissage élevé pour les premiers cycles d'apprentissage et de diminuer le pas au fur et à mesure afin d'affiner la précision sur la recherche des poids.

Test d'apprentissage

Nombre d'époques

Le test à été de charger des images de chaque classe. Un taux d'apprentissage fin et un nombre d'époques très élevé. le but est de vérifier le fonctionnement du neurone et de comprendre le rôle de ses paramètres.

On obtient ce graphique ci dessous pour avoir une bonne représentation de nos résultats.

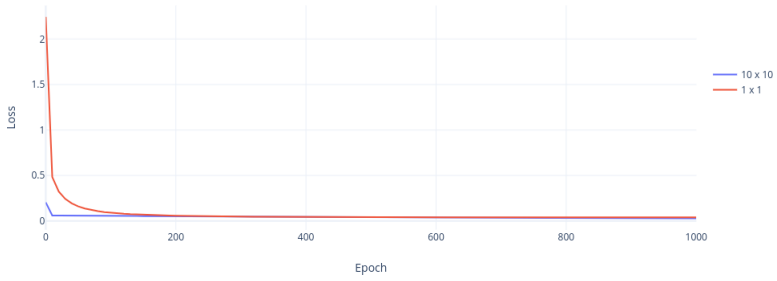


Pour 50 époques c'est en effet bien trop. En effet le test avec 1000 époques on se retrouve presque un bon résultat. On obtient 0.156150 Loss contre 0.039148 avec 1000 époques.

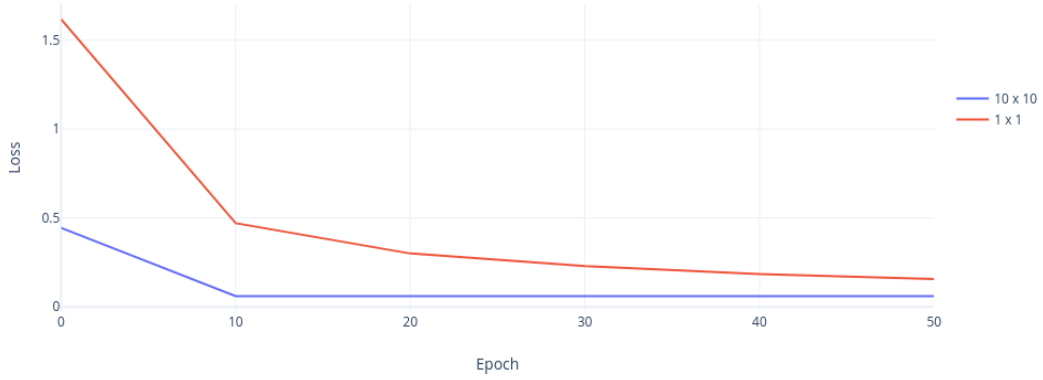
On se rend compte que le nombre d'époque est important pour définir la précision de mon résultat.

Nombre de pixels et d'images

Mon second test est basé sur le nombre de mes arguments du neurone. J'ai seulement augmenté le nombre de pixels dans les images sans toucher au nombre d'époque.



Plus le nombre de paramètres augmente plus l'apprentissage semble efficace. Maintenant une combinaison du nombre d'époques et de paramètres d'entrée.



Avec ce graphique plus zoomé on se rend compte que la première itération d'apprentissage 3 fois plus efficace que l'apprentissage sur une image de 1 pixel. On pourrait donc encore diminuer le nombre d'époques vu que je suis sur une ligne droite, pour la perdre à partir de 10 itérations ! Attention, augmenter le nombre de paramètres augmente la charge des calculs et donc directement du temps d'apprentissage. Le nombre d'images donne environs les mêmes résultats que l'augmentation du nombre de pixels.

Pas d'apprentissage

D'après mes tests, plus le pas d'apprentissage est petit plus notre recherche de minimisation de l'erreur sera précise, mais plus longue. Généralement si on diminue le pas d'apprentissage on augmente le nombre d'époques, sinon le neurone n'aura pas assez appris.

Test sur Dataset

Les tests effectués sur ma dataset contenant mes 3 genres d'animaux sont plus aléatoires. En effet, il est rare que le neurone arrive à déterminer à lui seul l'appartenance d'une image à l'une des trois classes. A chaque nouvel apprentissage, j'ai des résultats différents de ceux précédents, même en conservant les mêmes paramètres. J'ai essayé de modifier les hyper-paramètres pour trouver une stabilité dans mes résultats. Mais sur des images de $1090 * 1080$ pixels en niveau de gris avec un unique neurone, on ne sait pas ce qu'il cherche réellement à apprendre. Ce qu'il semble apprendre c'est le poids du niveau de gris sur énorme images. Ce qui rend la tâche très complexe.