Un réseau de neurones simple

Consignes générales pour un DM : Le devoir est facultatif et non noté. Les copies présentant un effort de recherche ou d'originalité recevront un bonus à valoir sur le prochain DS. Par ailleurs, ce prochain DS reprendra tel quel ou légèrement modifié une partie de ce DM.

Chaque fonction écrite devra être accompagnée de tests, et les résultats des tests devront figurer sur la copie. Si une fonction ne fonctionne pas, indiquez le problème, et le cas échéant l'erreur reçue, et expliquez autant que possible ce qu'elle signifie.

1 Présentation

Nous allons construire un réseau de neurones à une seule couche dont le but est de reconnaître des images de chiffres. Ce type de réseau de neurones s'appelle un « perceptron » et a été proposé par Frank Rosenblatt en 1957.

Les images que nous traiterons seront des images noir et blanc de format 3×5, représenté par des matrices de 0 et de 1 : chaque case correspond à un pixel, 1 signifie noir, et 0 signifie blanc. Par exemple :

```
un = [ [0, 1, 0],
       [1, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0]]
```

La matrice `un` représente une image qui représente le chiffre 1.

Notre réseau de neurones aura 15 neurones d'entrée : un pour chaque pixel de l'image lue, et 10 neurones de sortie : un pour chaque chiffre possible. Chaque neurone d'entrée sera relié à chaque neurone de sortie, ce qui fait 150 connexions (qu'on pourra appeler « synapses »). On décide que le neurone d'entrée correspondant au pixel (i, j) s'active dès que le pixel (i, j) de l'image lue vaut 1. Il envoie alors un signal vers les neurones de sortie. Mais à chaque synapse sera associé un flottant, appelé « coefficient synaptique », qui indique à quelle point la connexion passe facilement : plus ce flottant est grand et plus le signal passe facilement (voire est amplifié) dans cette synapse. Donc les neurones de sortie ne recevront pas tous la même quantité de signal. Nous décidons que ceux qui reçoient une quantité de signal supérieure à 1 s'activeront.

Tous les coefficients synaptiques seront enregistrés dans une matrice `P`. Le but de l'algo est de déterminer les valeurs judicieuses pour faire en sorte que lorsqu'une image représentant le chiffre k est lue, le neurone de sortie d'indice k, et seulement celui-ci s'active.

Rentrons dans les détails. Soit `Im` l'image lue (donc une matrice de format $(5, 3)$). On utilisera une matrice `P` de format $(5, 3, 10)$. Pour tout $i \in [\![0, 5[\![$, $j \in [\![0, 3[\![$, et $k \in [\![0, 10[\![$, `P[i][j][k]` représentera la connectivité de la synapse qui relie le pixel (i, j) de l'entrée à la sortie k.

On calcule alors, pour tout $k \in [\![0, 10[\![$:

$$\sum_{i=0}^{4} \sum_{j=0}^{2} \texttt{P[i][j][k]} \times \texttt{Im[i][j]}.$$

Ce nombre sera noté $\mathcal{A}(k, \texttt{Im}, \texttt{P})$. Il représente la quantité de signal reçu par le neurone de sortie k. Nous dirons que le neurone k est activé si ce nombre est $\geqslant 1$.

Notre but est de trouver une matrice `P` pour que pour tout k, le neurone k s'active si et seulement si une image représentant le chiffre k est lue.

Le fichier `bib_neurones.py` disponible sur le site fourni :

- Des images des dix chiffres (mais vous pouvez créer les votres),

- Une fonction pour créer une matrice `P` aléatoire ;
- Une ou deux variables globales qui peuvent être utile.

Il est inutile d'imprimer ce fichier avec le reste de votre travail ! Le plus simple est d'écrire un `from bib_neurones import *` au début de votre fichier.

2 Lecture d'une image

On pourra tester les fonctions de cette partie en prenant une matrice `P` aléatoire telle que définie dans `bib-neurones`. Les résultats seront naturellement aléatoires eux aussi...

1. Écrire une fonction `estActivé` prenant en entrée un entier k, une matrice `Im` de format $(5, 3)$, une matrice `P` de format $(5, 3, 10)$ et renvoyant un booléen qui indique si $\mathcal{A}(k, \texttt{Im}, \texttt{P}) \geqslant 1$.
2. Écrire une fonction `sorties_activée` prenant en entrée une matrice `Im` de format $(5, 3)$, une matrice `P` de format $(5, 3, 10)$ et renvoyant la liste des $k \in [\![0, 10[\![$ qui sont activé lors de la lecture de `Im`.

3 Réglage des poids

On va maintenant modifier les coefficients de `P` jusqu'à ce que notre réseau de neurone reconnaissent effectivement les images souhaitées. L'idée est qu'à chaque fois qu'un neurone de sortie s'active alors qu'il n'aurait pas du, nous diminuons les poids des coefficients de `P` qui sont intervenus dans le calcul de $\mathcal{A}(k, \texttt{Im}, \texttt{P})$. Et au contraire, lorsqu'un neurone de sortie qui aurait du s'activer ne l'a pas fait, nous augmentons les coefficients qui sont intervenus dans le calcul de $\mathcal{A}(k, \texttt{Im}, \texttt{P})$.

Précisément, lorsque la lecture d'une image `Im` censée représenter le chiffre k_0 :

- Si k_0 n'a pas été activé, nous augmentons `P[i][j][k0]` pour tout (i, j) tel que `Im[i][j]==1` ;
- Pour tout $k \in [\![0, 10[\![$, si $k \neq k_0$ mais que le neurone k s'est activé, nous diminuons `P[i][j][k]` pour tout (i, j) tel que `Im[i][j]==1`.

Nous décidons plus ou moins arbitrairement d'augmenter et de diminuer les coefficients de 0.1.

1. Écrire une fonction `augmenteCoeffConcernés` prenant en entrée une image `Im`, la matrice des poids `P`, et un entier $k_0 \in [\![0, 10[\![$ tel que le neurone k_0 ne s'est pas activé alors qu'il aurait du, et qui augmente `P[i][j][k0]` de 0.1 pour tout (i, j) tel que `Im[i][j]==1`.
2. Écrire de manière analogue une fonction `diminueCoeffConcernés`.
3. Écrire une fonction `litUneImage` prenant en entrée une image `Im`, l'entier k_0 qu'elle est censée représenter, et la matrice des poids `Q` et qui appelle les fonctions précédentes pour toutes valeurs de k pour lesquelles c'est nécessaire.

   Cette fonction devra en outre renvoyer un booléen qui indique si il y a eu au moins une erreur dans la lecture : un neurone activé qui n'aurait pas du, ou un neurone non activé qui aurait du l'être.

   À ce stade, vous pouvez déjà lancer plusieurs fois de suite la lecture d'une même image, puis voir grâce à `sorties_activées` si votre réseau reconnaît cette image.
4. Écrire une fonction qui se charge de lire toutes les images de chiffre dont vous disposez et qui appelle la précédente à chaque fois. Cette fonction devra en outre renvoyer un booléen qui indique si il y a eu au moins une erreur dans la lecture d'une de ces images.
5. Écrire une fonction `apprentissage` qui commence par initialiser aléatoirement une matrice `P` puis qui appelle la fonction précédente tant que la lecture d'au moins une image donne au moins une erreur. On renverra alors la matrice `P` finale : c'est votre réseau de neurones qui a fini son apprentissage !

4 Bonus

Vous pouvez maintenant imaginer de nombreuses améliorations, par exemple :

- Augmenter la taille des images ;
- Mettre plusieurs versions de chaque chiffre ;
- Augmenter le nombre de symboles différents.