Un réseau de neurones simple

Cyril Charignon

Table des matières

1 Présentation **1**

2 Programmation **2**
2.1 Lecture d'une image 2
2.2 Correction des poids 2
2.3 Programme final 2
2.4 Amélioration et variantes 2

3 Version matricielle **3**
3.1 Notations 3
3.2 Algèbre linéaire 3
3.3 Limites d'un perceptron 3

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, 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.

On notera (n, p) le format des images à lire (n lignes et p colonnes).

Notre réseau de neurones aura un neurone d'entrée pour chaque pixel de l'image lue, ce qui fait np neurones d'entrée. Il aura un neurone de sortie pour chaque chiffre possible, ce qui fait a priori 10 neurones de sortie. Toutefois, on préférera noter N_s le nombre de neurones de sortie, à la fois par clarté et pour permettre de rajouter éventuellement d'autres sorties ultérieurement (des lettres, des symboles...)

Chaque neurone d'entrée sera relié à chaque neurone de sortie, ce qui fait $n \times p \times N_s$ 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. Le but est que le neurone de sortie correspondant au chiffre représenté par l'image lue reoive un signal proche de 1, et les autres un signal proche de -1.

Tous les coefficients synaptiques seront enregistrés dans une matrice `P`. Le but de l'algo est de déterminer les valeurs judicieuses pour ces coefficients 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 (n, p)). On utilisera une matrice `P` de format (N_s, n, p). Pour tout $i \in [\![0, n[\![$, $j \in [\![0, n[\![$, et $k \in [\![0, N_s[\![$, `P[k][i][j]` représentera la connectivité de la synapse qui relie le pixel (i, j) de l'entrée à la sortie k.

On pose alors, pour tout $k \in [\![0, N_s[\![$:

$$\mathcal{A}(k, Im, P) = \sum_{i=0}^{n} \sum_{j=0}^{p} \texttt{P[k][i][j]} \times \texttt{Im[i][j]}.$$

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 fournit :

- 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.

2 Programmation

2.1 Lecture d'une image

1. Écrire une fonction `A` prenant en entrée Im, k, et P et renvoyant $\mathcal{A}(k, Im, P)$.
2. Nous dirons qu'un neurone de sortie est activé lorsque le signal qu'il reçoit est supérieur à $\frac{1}{2}$. Écrire une fonction `sorties_activées` prenant en entrée Im et P, et renvoyant le tableau des $k \in [\![0, N_s[\![$ pour lesquels la sortie k est activée en lisant Im.

2.2 Correction des poids

Lors de l'entraînement de notre réseau, nous voudrons faire en sorte que lors de la lecture d'une image Im qui représente le chiffre k_0, la sortie k_0 reçoive une quantité de signal proche de 1, et que les autres sorties reçoivent une quantité de signal proche de -1.

Pour tout $k \in [\![0, Ns[\![$, nous appellerons « erreur à la sortie k en lisant l'image Im » et noterons $\text{err}(Im, P, k_0, k)$ le nombre : « valeur voulue $- \mathcal{A}(k, Im, P)$ », où « valeur voulue » vaut 1 si $k = k_0$ et 0 sinon.

On fixe encore en coefficient η qui nous permettra de régler à quelle vitesse on corrigera les coefficient. On décide alors que lors de la lecture de Im, pour tout $(k, i, j) \in [\![0, Ns[\![\times [\![0, n[\![\times [\![0, p[\![$, le cofficient `P[k][i][j]` sera augmenté de

$$\eta \times Im[i][j] \times \text{err}(Im, P, k_0, k).$$

Le fait de multiplier par $Im[i][j]$ permet de faire en sorte que seules les synapses intervenant dans le calcul de $\mathcal{A}(Im, P, k)$ soient modifiées.

1. Programmer la fonction `err`.
2. Programmer une procédure `lecture_image` prenant la matrice P, le coefficient η, une image Im, et le chiffre k_0 qu'elle représente et modifiant chaque coefficient de P selon la formule ci-dessus.
3. On suppose qu'on dispose d'une banque d'images, qui se présente sous la forme d'un tableau de couples : `[ (Im0, k0), (IM1, k1) , ... ]` Chaque couple est formé d'une image, et du chiffre qu'elle représente. Un exemple est le tableau `TOUT` du fichier `bib_neurones.py`.

   Écrire une fonction `lecture_banque` prenant en entrée P et une telle banque d'images, et appliquant le programme précédent pour chaque image de la banque.

2.3 Programme final

1. Écrire un prédicat `tout_juste` prenant en entrée P et une banque d'images comme à la question précédente et indiquant si `sorties_activées` renvoie le résultat attendu pour chaque image de la banque.
2. En déduire le programme final `entraînement` qui prend en entrée une banque d'image et le nombre de sorties N_s, qui initialise une matrice de poids P et qui applique `lecture_banque` jusqu'à ce que toutes les images soient reconnues correctement.

2.4 Amélioration et variantes

1. Déjà on peut augmenter le nombre d'images, soit en mettant plusieurs images qui représentent le même chiffre, ou en introduisant de nouveaux symboles à reconnaître. On peut aussi augmenter le nombre de pixels.
   Afin d'avoir une idée des limites de notre programme, nous pouvons compter le nombre de multiplications effectuées par un tour le la boucle principale.
2. Dans le programme `lecture_image`, vérifiez si vous n'avez pas exécuté plusieurs fois le programme `err` sur les mêmes entrées...
3. Les nombres $\mathcal{A}(im, k, P)$ sont calculés deux fois à chaque itération de la boucle principale : une fois par `lecture_banque` et une fois par `tout_juste`. Le programme serait environ deux fois plus rapide si on évitait ça...
4. En pratique, il n'est souvent pas réaliste d'espérer que le réseau finisse par reconnaître toutes les images de la banque. On peut mettre en entrée un flottant $q \in [0, 1]$ et d'arrêter l'entraînement une fois qu'une proportion q (au moins) d'images est correctement reconnue.

3 Version matricielle

3.1 Notations

Le but de cette partie est de prendre un peu de recul sur les formules utilisées et de s'apercevoir des limites d'un perceptron.

Changeons légèrement les notations soit $N_e = n \times p$ le nombre de neurones d'entrée. Numérotons les pixels d'entrée de 0 à $N_e - 1$ (au lieu de les indexer par un couple $(i, j) \in [\![0, n[\![\times [\![0, p[\![$.

La matrice P devient de format (N_s, N_e). Et la formule pour A devient : pour toute image Im et tout $k \in [\![0, N_s[\![$,

$$\mathcal{A}(Im, k, P) = \sum_{i=0}^{N_e-1} P[k][i] \times Im[i].$$

3.2 Algèbre linéaire

On reconnaît alors un simple produit matriciel :

$$\mathcal{A}(Im, k, P) = P[k] \times Im.$$

Ainsi, $\mathcal{A}(\cdot, k, P)$ est une forme linéaire de matrice $P[k]$ (matrice ligne : la ligne k de P).

Dès lors l'ensemble des images Im pour lesquelles $\mathcal{A}(Im, k, P) = 1$ est un hyperplan. Et l'ensemble des images qui activent le neurone k est un demi-plan.

3.3 Limites d'un perceptron

Ceci nous permet d'imaginer des ensembles qui ne peuvent *pas* être discriminés par un réseau de neurone.

D'après le théorème de Hahn-Banach : soient E et F deux ensembles d'images. Il existe une matrice P tel que le neurone k s'active pour toutes les images de E et ne s'active pour aucune image de F si et seulement si $\mathrm{Conv}(E) \cap \mathrm{Conv}(F) = \emptyset$.

Pour obtenir un réseau de neurones capable de discrimer des ensembles plus complexes, il faut introduire de la non-linéarité : c'est le principe des réseaux à plusieurs couches. Le terme « deep learning » est utilisé lorsque le nombre de couche devient grand.