MPSI, premier devoir surveillé d'informatique

La difficulté est dans l'ensemble croissante, et il n'est pas nécessaire, ni suffisant d'avoir traité la totalité du sujet pour obtenir la note maximale.

Les différents identificateurs sont notés à l'aide d'une police italique (par exemple $t[i]$) lorsqu'ils sont considérés du point de vue mathématique, et avec une police romaine à chasse fixe lorsqu'ils sont plutôt considérés du point de vue de leur implémentation sur machine (par exemple `t[i]`).

1 Cours

1. (a) Écrire une fonction `appartient` prenant en entrée un tableau `t` et un élément `x` et renvoyant un booléen qui indique si $x \in t$.
   (b) Soit `t` un tableau et n sa longueur. Combien de cases de `t` sont-elles lues par votre programme ?
2. On écrit à présent une fonction pour tester si une chaîne de caractères est inclue dans une autre. On fixe deux chaînes de caractères `m` et `t`. On note `lm = len(m)` et `lt = len(t)`. A priori, `lm < lt`.
   (a) Soit $i \in [\![0, lt - lm + 1[\![$. Écrire une formule mathématique qui exprime précisément le fait que « `m` est présent dans `t` en démarrant en position i ».
   (b) Écrire une fonction prenant en entrée les chaînes de caractères `m` et `t`, un indice $i \in [\![0, lt - lm + 1[\![$ et renvoyant un booléen indiquant si `m` est présent dans `t` en démarrant en position i.
   (c) Écrire une formule mathématique qui exprime précisément l'idée que « `m` est présent quelque part dans `t` ».
   (d) Écrire finalement une fonction prenant en entrée les chaînes de caractères `m` et `t` et renvoyant un booléen qui indique si `m` est inclu dans `t`.
   (e) Combien de comparaisons entre deux lettres sont effectuées par votre programme ? Donner une réponse en fonction de lm et lt.

2 Problème : réseau de neurones multicouche

2.1 Introduction et notations

Le but de ce probème est de construire un réseau de neurone un peu plus général que celui vu en DM. Restons sur l'idée de reconnaître différents types d'images. Cependant, cette fois les images ne représenteront pas forcément des chiffres. Nous prenons un ensemble $\mathcal{C}$ que nous appelons ensemble des « catégories » (par exemple $\mathcal{C}$ contient les chiffres comme dans le DM mais aussi des lettres et autres symboles), et chaque image est supposée appartenir à une des catégories. Comme dans le DM, les images seront en noir et blanc, et seront représentées par des tableaux de 0 et de 1.

Notations :

- n le nombre de lignes et p le nombre de colonnes des images.
- $N_e = n \times p$ le nombre de pixels des images. Ce sera aussi le nombre de neurones d'entrée de nos réseaux. Pour toute image `Im` et tout $(i, j) \in [\![0, n[\![\times [\![0, p[\![$, nous dirons que le neurone d'entrée (i, j) est activé en lisant `Im` lorsque `Im[i][j]==1`.
- $N_s = \text{Card}(\mathcal{C})$: c'est le nombre de catégories à reconnaître. Ce sera aussi le nombre de neurones de sorties de nos réseaux.
- L'ensemble des catégories sera enregistré dans un tableau `catégories` de longueur N_s. Par exemple `catégories = ["0", "1", "2","+", "-", "a",...]`.

2.2 Perceptron simple

Dans ce premier réseau, nous relions chaque neurone d'entrée à chaque neurone de sortie par une synapse, ce qui nous fait $N_e \times N_s$ synapses. Pour tout $(i, j, k) \in [\![0, n[\![\times [\![0, p[\![\times [\![0, N_s[\![$, nous noterons $P^k_{i,j}$ un nombre qui représente la «connectivité» de la synapse qui relie l'entrée liée au pixel (i, j) à la sortie liée à la catégorie k. Dans les programmes, tous ces coefficients seront rangés dans une matrice `P` de format (n, p, N_s) de telle sorte que $P^k_{i,j}$ s'obtiendra par `P[i][j][k]`.

Pour toute image `Im`, et pour tout $k \in [\![0, N_s[\![$, nous posons :

$$\mathcal{A}(k, \texttt{Im}, \texttt{P}) = \sum_{i=0}^{n-1} \sum_{j=0}^{p-1} \texttt{P[i][j][k]} \times \texttt{Im[i][j]}$$

c'est la quantité de signal reçue par le neurone de sortie k. Nous dirons que le neurone k est « activé » lors de la lecture de `Im` lorsque $\mathcal{A}(k, \texttt{Im}, \texttt{P}) \geqslant 1$.

2.2.1 Lecture d'une image

1. Écrire une fonction `sigal_reçu` prenant en entrée $k \in [\![0, N_s[\![$, une image `Im`, une matrice `P` de format (n, p, N_s) et renvoyant $\mathcal{A}(k, \texttt{Im}, \texttt{P})$.
2. Écrire une fonction `sorties_activée` prenant en entrée une image `Im`, le nombre N_s de catégories d'image, une matrice `P` de format (n, p, N_s) et renvoyant la liste des $k \in [\![0, N_s[\![$ qui sont activé lors de la lecture de `Im`.
3. Écrire une fonction `sortie_dominante` qui prend en entrée une image `Im` et une matrice `P` de format (n, p, N_s) et qui renvoie l'indice $k \in [\![0, N_s[\![$ pour lequel $\mathcal{A}(k, \texttt{Im}, \texttt{P})$ est maximal.
4. Enfin écrire une fonction semblable à la précédente mais prenant en outre en entrée le tableau `catégorie` et renvoyant non plus l'indice k mais la catégorie correspondante.

2.2.2 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 $\mathcal{A}(k, \texttt{Im}, \texttt{P})$.

On utilise cette fois une formule un peu plus fine que celle du DM : lorsque nous corrigeons un coefficient, nous le corrigeons d'une valeur proportionnelle à l'écart entre la valeur de $\mathcal{A}(k, Im, P)$ souhaitée, et celle obtenue. En prenant comme valeur souhaitée 2 si k est la catégorie de `Im` et 0 sinon, ceci donne la règle suivante : pour tout $(i, j, k) \in [\![0, n[\![\times [\![0, p[\![\times [\![0, N_s[\![$, lors de la lecture d'une image `Im` appartenant à la catégorie d'indice k_0 :

- Si `Im[i][j]==1` et $k = k_0$, nous augmentons $P[i][j][k]$ de la quantité $0.1 \times (2 - \mathcal{A}(k, \texttt{Im}, \texttt{P})$. (Le coefficient 0.1 est choisi plus ou moins arbitrairement.)
- Si `Im[i][j]==1` et $k \neq k_0$, nous augmentons $P[i][j][k]$ de la quantité $-0.1 \times \mathcal{A}(k, \texttt{Im}, \texttt{P})$.
- Si `Im[i][j]==0`, le coefficient n'est pas modifié.

Pour ceux qui se posent la question : oui, augmenter d'une quantité négative, cela revient à diminuer.

Remarque : On peut rassembler ces trois cas en un seul : dans tous les cas, le coefficient est modifié de :

$$0.1 \times \big(Im[i][j]\big) \times (\text{Valeur de } \mathcal{A} \text{ souhaitée} - \text{Valeur de } \mathcal{A} \text{ obtenue}) .$$

1. Écrire une fonction `litUneImage` qui prend en entrée une image `Im`, une matrice `P` de format (n, p, N_s), et l'indice k_0 de la catégorie de `Im`, et qui applique la règle ci-dessus à chaque coefficient de `P`.
2. Si votre fonction calcule plusieurs fois certaines valeurs de $\mathcal{A}$, corrigez-la pour que ça ne soit plus le cas. (Si votre fonction était déjà optimale, vous aurez le point sans rien faire de plus que de le signaler en une phrase.)
3. À présent écrire une fonction qui applique la fonction précédente à un grand nombre d'images. Cette nouvelle fonction prendra en entrée un tableau contenant des couples de type `(image, numéro de catégorie)`. Ainsi ce tableau ressemblera à : `[ (Im0, k0), (Im1, k1), ...]`.

   En outre, cette fonction lancera la fonction `sortie_dominante` sur chaque image, et elle renverra un booléen qui indique si, pour au moins une image, la sortie ayant reçu le plus signal n'était pas celle qui correspondait à sa catégorie. Attention : on utilisera `sortie_dominante` *avant* `litUneImage`.
4. On suppose connue une fonction d'entête `init_P(n,p,s)` qui crée une matrice aléatoire de format `(n, p, s)`. Écrire la fonction finale, qui prend en entrée un tableaux d'images comme à la question précédente, ainse que le tableau des catégories, qui crée une matrice `P` puis la modifie grâce à la fonction précédente jusqu'à ce que la sortie renvoyée soit la bonne pour chaque image.
5. Dans le cas où le nombre d'images est très grand, il ne semble pas réaliste que notre réseau soit capable de les reconnaître toutes.

   Expliquer comment modifier vos fonctions prédentes pour prendre en outre un flottant $p \in [0, 1]$ et arrêter l'apprentissage dès que la proportion d'erreurs est inférieure à p.

2.3 Limites d'un perceptron à une couche

On étudie ici un cas particulier extrêmement simple : les images lues n'auront que deux pixels (il n'y a donc que quatre images différentes !), et nous voulons reconnaître un seul type d'objets. Autrement dit, nous prenons $N_s = 1$, $N_e = 2$, $n = 1$, $p = 2$. Il n'a plus que deux coefficients synaptiques ; nous noterons α celui qui relie le premier pixel à la sortie, et β celui qui relie le deuxième pixel à l'unique sortie.

Soit `Im` une image. Notons x le contenu du premier pixel (donc $x \in \{0, 1\}$) et y celui du deuxième. Nous associons alors à `Im` le point du plan de coordonnées (x, y).

1. Représeter dans le plan les quatre images possibles.
2. Écrire une formule donnant $\mathcal{A}(Im, P, 0)$ en fonction de x, y, α, et β.

   On constate ainsi que la courbe d'équation $\mathcal{A}(Im, P, 0)$ est une droite. Donc l'ensemble des images pour lesquelles la sortie 0 (qui est la seule sortie) s'active est un demi-plan délimité par cette droite.

3. Trouver un ensemble E d'image qui n'est pas reconnaissable par notre réseau de neurone. Autrement dit tel que quels que soient les coefficients α et β, il y aura toujours soit une image de E pour laquelle la sortie ne s'active pas, soit une image qui n'est pas dans E pour laquelle la sortie ne s'active pas.

Ceci se généralise pour n'importe quelles dimemsions. Pour des images à trois pixels, qu'on pourrait représenter par des points de l'espace, l'ensemble des points tels que $\mathcal{A}(Im, P, k)$ vaut 1 est un plan. Et dans le cas général, c'est un « hyperplan dans un espace de dimension N_e » [1]*. On dit que notre réseau de neurones est « linéaire ». Et il existera toujours des ensembles d'images qu'il sera impossible de reconnaître.*

2.4 Perceptron multicouches

Un perceptron à plusieurs couches n'a pas le défaut du monocouche : il a été démontré qu'il peut reconnaître presque n'importe quel ensemble d'images. Par contre, il peut nécessiter un grand nombre de neurones, et partant un très grand nombre d'images pour son apprentissage.

Nous conservons nos $n \times p$ neurones d'entrée, reliés aux pixels de l'image lue, ainsi que nos N_s neurones de sortie, un pour chaque catégorie d'image. Mais nous ne relions pas directement : nous intercalons une couche de neurones entre les deux. Notons N_i le nombre de neurones de cette couche intermédiaire. Chaque neurone d'entrée sera relié à chaque neurone intermédiaire (cela nous fait donc $n \times p \times N_i$ synapses pour le premier étage), et chaque neurone intermédiaire sera relié à un neurome de sortie (ce qui nous donne $N_i \times N_s$ synapses pour le deuxième étage). Une matrice `P1` de format (n, p, N_i) contiendra les coefficients de synapses reliant les neurones d'entrée aux neurones intermédiaires, et une matrice `P2` de format (N_i, N_s) contiendra les coefficients des synapses reliant les neurones intermédiaires aux neurones de sortie.

La formule donnant la quantité de signal reçue par un neurone intermédiaire est la même que celle utilisée pour les neurones de sortie précédemment : nous posons pour tout $k \in [\![0, N_i[\![$,

$$\mathcal{A}(P1, Im, k) = \sum_{i=0}^{n-1} \sum_{j=0}^{p-1} Im[i][j] \times P1[i][j][k].$$

Pour alléger les notations dans les formules à suivre, je ne noterai plus les matrices des poids en entrée des différentes fonctions : je noterai par exemple $\mathcal{A}(Im, k)$ au lieu de $\mathcal{A}(P1, Im, k)$. Il faudra avoir à l'esprit que ce calcul dépend néammoins de `P1`, et mettre `P1` (et par la suite `P2`) en entrée des programmes à venir lorsque nécessaire.

Il faut ensuite appliquer à $\mathcal{A}(Im, k)$ une fonction non linéaire (c'est-à-dire utilisant autre chose que des additions et produits par des constantes) sans quoi les défauts mis en valeur à la partie précédente sont conservés. Un choix fréquent est d'utiliser une tangente hyperbolique. On appliquera donc th à chaque valeur de $\mathcal{A}$ obtenue.

Remarque : La fonction utilisée ici, th dans notre cas, s'appelle la « fonction d'activation ». Lorsque $\mathcal{A}(Im, k)$ est grand, $\text{th}(\mathcal{A}(Im, k))$ est proche de 1, et nous dirons que le neurone est activé. Au contraire, lorsque $\mathcal{A}(Im, k)$ tend vers $-\infty$, $\text{th}(\mathcal{A}(Im, k))$ tend vers -1. Dans ce cas, nous dirons que le neurone n'est pas activé. Mais contrairement à la partie précédente où le neurone final était complétement activé ou complétement non activé, nous autorisons ici des états

1. Vivement le cours de maths d'algèbre linéaire !

intermédiaires.

Au final, pour tout $l \in [\![0, N_s[\![$, le signal reçu par le neurone de sortie, qui sera notée $\mathcal{B}(P1, P2, Im, l)$ est définie par :

$$\mathcal{B}(P1, P2, Im, l) = \sum_{k=0}^{N_i-1} \text{th}\left(\mathcal{A}(P1, Im, k)\right) \times P2[k][l].$$

Qu'on notera souvent juste $\mathcal{B}(Im, l)$ pour simplifier.

2.4.1 Lecture d'une image

1. Écrire une fonction pour calculer $\mathcal{A}$.
   Indication : Vous avez toujours le droit d'utiliser les fonctions écrites lors des questions précédentes...
2. Rappeler la définition et la dérivée de la fonction th.
3. Définir pour Python la fonction tangente hyperbolique ainsi que sa dérivée. On pourra utiliser la fonction exponentielle de numpy, obtenue via `np.exp` après avoir chargé numpy au moyen de `import numpy as np`.
4. Écrire une fonction pour calculer $\mathcal{B}$.
5. À présent écrire une fonction prenant en entrée `P1`, `P2` et `Im` et renvoyant un tableau contenant les valeurs de $\mathcal{B}(P1, P2, Im, l)$ pour tout $l \in [\![0, N_s[\![$ ainsi qu'un autre tableau contenant toutes les valeurs de $\mathcal{A}(P1, Im, k)$, pour tout $k \in [\![0, N_i[\![$. On prendra soin de ne pas calculer plusieurs fois une même valeur de $\mathcal{A}$...

2.4.2 Correction des poids

Cette fois, pour tout $l \in [\![0, N_s[\![$, et toute image `Im`, nous décidons que la valeur $\mathcal{B}(Im, l)$ devrait être 1 si l'image `Im` appartient à la catégorie l, -1 sinon. On notera $d_l(Im)$ cette valeur souhaitée, et $e_l(Im) = \mathcal{B}(Im, l) - d_l(Im)$ la différence entre la valeur obtenue et la valeur souhaitée.

Les synapses de la deuxième couche sont corrigées ainsi : pour tout $(l, k) \in [\![0, N_s[\![\times [\![0, N_i[\![$, le coefficient `P2[k][l]` est diminué de

$$0.1 \times e_l(Im) \times \text{th}\left(\mathcal{A}(k, Im)\right).$$

Le dernier facteur permet de corriger principalement les synapses provenant des neurones qui ont le plus contribué au calcul de $\mathcal{B}(Im, l)$, et le facteur $e_l(Im)$ permet de corriger proportionellement à l'erreur commise.

La formule est un peu plus complexe pour les synapses de la première couche : pour tout $(i, j, k) \in [\![0, n[\![\times [\![0, p[\![\times [\![0, N_i[\![$, le coefficient `P1[i][j][k]` est diminué de

$$0.1 \times \text{th}'\left(\mathcal{A}(k, Im)\right) \times Im[i][j] \times \sum_{l=0}^{N_s-1} e_l(Im) \times P2[k][l].$$

(La somme à droite peut être interprétée comme l'erreur totale à daquelle a contribué la synapse.)
Écrire une ou plusieurs fonctions pour corriger les synapses après lecture d'une image, puis programmer comme précédemment la lecture de plusieurs images et l'éducation de votre réseau.

Remarques finales :

- *Lorsque le nombre de couches devient grand, on dit qu'on utilise un réseau « profond » et qu'on fait du « deep learning ».*
- *Le paramètre* 0.1 *utilisé dans les formules de correction s'appelle la « vitesse d'apprentissage ». Si elle est trop petite, l'apprentissage sera trop lent, et si elle est trop grande, on risque de rater (sauter par dessus si vous voulez) le bon point d'équilibre.*
- *Souvent, on rajoute à chaque couche un neurone d'entrée toujours activé, qui permet donc de décaler d'une constante le signal. Ce neurone s'appelle le «biais».*