Intelligence Artificielle - Deep Learning
Feuille 1

Réseaux de neurones (I)

Présentation

Dans ce TP, nous allons commencer par construire un réseau de neurones "from scratch" en Python. Nous allons travailler sur un squelette de programme python qu'il faudra compléter. Quelques instructions sont à utiliser telles quelle. Nous avons mis des commentaires dans le code pour expliquer les instructions.

Nous allons travailler sur le corpus `mnist` (voir `http://yann.lecun.com/exdb/mnist/` pour plus de détails), un des corpus les plus utilisés en machine learning et en deep learning. Il est composé de 70000 images de chiffres manuscrits. Les images sont de taille 28×28 pixels. Chaque pixel est codé par un nombre entre 0 et 255 correspondant au niveau de gris.

Dans un second temps, à partir de l'exercice 2, nous utiliserons la bibliothèque `keras`.

Commencez donc par télécharger le fichier `lab1_skeleton.py` disponible à l'adresse du cours : `https://www.labri.fr/~zemmari/ia-m1info`.

Si vous travaillez au CREMI, via `ssh` par exemple, vous devez également activer l'environnement `tensorflow` en exécutant l'instruction

```
source /net/ens/DeepLearning/tensorflow/bin/activate
```

Quelques rappels.

Rappelons qu'un réseau de neurones est composé d'une suite de couches successives. Chaque suite est un ensemble de neurones formels. Les architectures de base considèrent que les signaux ne circulent que d'une couche vers la suivante.

Chaque neurone dans la $k^{ème}$ couche est connecté à tous les neurones de la $(k-1)^{ème}$ couche et les neurones d'une couche forment un ensemble indépendant.

Un réseau de neurones est composé de :

- une couche d'entrée x
- un (éventuel) nombre arbitraire de couches cachées
- une couche de sortie y
- un ensemble de poids W et de biais b
- un ensemble de fonctions d'activation, une par couche.

Exercice 1. Un réseau à un seul neurone, faisons les calculs ... juste une fois, c'est promis

Dans un premier temps, nous allons considérer un réseau composé d'un seul neurone. Ce neurone est connecté à 28×28 entrées (les pixels de l'image) et utilise comme fonction d'activation, le fonction

sigmoid σ :

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

Notre neurone va juste "apprendre" à distinguer entre le chiffre 5 et les autres chiffres (dans le corpus `mnist`). Ainsi, une fois entraîné, le neurone donnera en sortie la valeur 1 si il voit un 5 et 0 sinon.

1. Compléter le fichier en définissant la fonction sigmoid.
2. Donner les expressions mathématiques de la sortie y du neurone en fonction de x, W, b et σ.
3. Donner l'expression mathématique de le fonction de perte $\mathcal{L}$. Utiliser pour cela la `cross_entropy` telle que définie dans le cours. Ecrivez la définition Python de cette fonction.

Pour réaliser la rétro-propagation, nous aurons besoin de connaitre les changement de la fonction de perte en fonction des changements de chaque paramètre w de W et du biais b.

1. Ecrire l'expression de la dérivée de $\mathcal{L}$ en par rapport à chaque paramètre w.
2. Compléter le fichier pour implémenter tous les éléments nécessaires à l'entrainement de votre modèle.
3. Entrainer le modéle.

Exercice 2. Un réseau de neurones avec une couche cachée

Dans cet exercice, et dans le suivant, nous allons utiliser `keras`. Notre réseau sera constitué des coûches suivantes : une couche d'entrée, une couche cachée avec 64 neurones et une couche de sortie avec, pour fonction d'activation, la fonction `softmax`.

Récupérer le fichier `lab1-2_skeleton.py` disponible à l'adresse du cours.

1. Modifier le fichier pour implémenter ce réseau de neurones.
2. Entrainer le réseau et mesurer son "accuracy".

Exercice 3. Un réseau pour reconnaitre tous les chiffres

À présent, nous allons adapter le reseau précédent pour reconnaire tous les 10 chiffres.

1. Expliquer, avec une figure, l'architecture du nouveau réseau.
2. Redéfinir, dans le programme, la nouvelle fonction de perte.
3. Entrainer et tester le nouveau réseau.