université PARIS-SACLAY

FACULTÉ DES SCIENCES D'ORSAY

Apprentissage supervisé pour les réseaux de neurones artificiels

Méthodes d'apprentissage

Dans la littérature, les méthodes d'apprentissage peuvent se structurer en trois catégories : i) l'apprentissage supervisé, ii) l'apprentissage par renforcement, et iii) l'apprentissage non supervisé. Les travaux pratiques proposés dans ce texte sont orientés vers **l'apprentissage supervisé**. Les techniques utilisées permettent l'identification automatique des hyperparamètres d'un réseau de neurones (ex. poids et biais) à partir des données en entrée et des sorties attendues. Le processus d'apprentissage vise donc à réduire l'erreur en sortie du réseau par rapport aux valeurs attendues.

Les problèmes abordés par l'apprentissage supervisé peuvent se concentrer sur divers objectifs. Ces objectifs peuvent être la régression, la classification, la labellisation (classification multi-étiquettes), la recherche et le classement. Dans ce cadre, nous limiterons le travail à des problèmes de régression et de classification.

Afin de rendre le travail d'implémentation accessible et efficace, nous utiliserons une bibliothèque Python (scikit-learn) intégrant les méthodes d'apprentissage, des objets de type perceptron et perceptron multi-couches, ainsi que des méthodes d'évaluation. Pour commencer, il est nécessaire de procéder à l'installation de la bibliothèque.

```
pip install sklearn
```

Le perceptron

Dans un premier temps, nous nous pencherons sur l'entraînement d'un perceptron pour l'implémentation des opérateurs logiques OR, AND et XOR. Ces opérateurs ont été paramétrés manuellement au début du cours. Vous pourrez ainsi comparer les paramètres obtenus par apprentissage automatique à ceux définis manuellement.

Pour initier l'apprentissage automatique, veuillez suivre la procédure détaillée ci-dessous :

- Dans un script python, commencez par importer la définition du perceptron de la bibliothèque scikit-learn

```
from sklearn.linear_model import Perceptron
```

- Définissez toutes les valeurs possibles pour les entrées de l'opérateur logique

```
#liste d'entrées possibles pour l'opérateur logique
data = [[0,0], [0,1], [1,0], [1,1]]
```

- Définissez les labels en sortie attendus selon l'opérateur concerné (ex. pour l'opérateur OR) :

```
#labels correspondants selon liste d'entrées
ORlabels = [0, 1, 1, 1]
```

- Créez une instance du perceptron et indiquez le nombre d'itérations max. admis pour l'apprentissage ainsi que le seuil de d'erreur toléré (paramètres de convergence).

```
#Création du perceptron
operateurOR = Perceptron(max_iter=40, tol=1e-3)
```

- Exécutez la procédure d'apprentissage en utilisant la fonction **fit** de la variable perceptron et en indiquant en paramètres les entrées et les labels préalablement définis.
- La fonction **predict** renvoie la valeur en sortie du réseau pour une entrée spécifiée.

Exercices de classification

1. Concevez et entraînez les perceptrons pour les opérateurs logiques OU, ET et OU-Exclusif.
2. Vérifiez les prédictions du perceptron à l'aide de la fonction **predict**. Est-ce que toutes les prédictions sont correctes ? Oui, non, pourquoi ?
3. Utilisez la fonction score pour estimer la précision moyenne du perceptron après l'entraînement. Rapportez les résultats. Le score reflète-t-il la qualité des résultats du perceptron ? Quelle est la plage de valeurs du score rapporté par la bibliothèque ?

Exercice d'OU-Exclusif comme un classifieur

4. Comme constaté dans l'exercice précédent, l'opérateur OU-Exclusif ne peut pas être satisfait par un classifieur linéaire. Il est donc nécessaire d'utiliser une topologie de perceptron multi-couche. Pour ce faire, utilisez le MLPClassifier de la bibliothèque scikit-learn.

```
#Importation de la définition du perceptron multi-couche
from sklearn.neural_network import MLPClassifier
```

5. Choisissiez une fonction d'activation type **tangente hyperbolique** ('tanh'), **une couche** cachée, deux neurones et une **limite d'itérations** de 10000. Référez-vous à la documentation : lien.

   Est-ce que le modèle converge lors de l'étape d'apprentissage ? Comparez les valeurs prédites par le réseau et celles attendues. Est-ce que son score atteint la valeur de 1 ? Est-ce que les résultats de l'entraînement du réseau sont les mêmes à chaque cycle d'apprentissage ? Faites plusieurs essais.

6. A partir des résultats obtenus dans l'exo. 5, indiquez les poids obtenus dans le modèle topologique de la Fig. 1. Les poids du réseau correspondent à l'attribut **coefs_** de l'objet ($w_{1a}, w_{1b}, w_{2a}, w_{2b}, w_3, w_4$). Les biais du réseau sont définis dans l'attribut **intercepts_**(b_1, b_2, b_3).

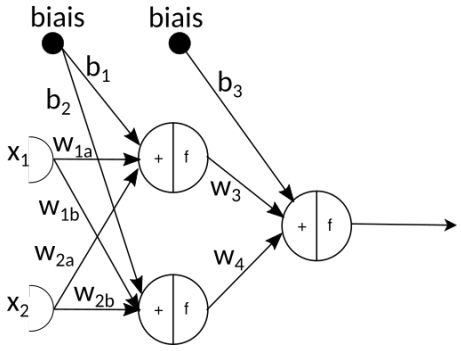

Fig. 1 : Topologie du perceptron multi-couche - Opérateur logique OU-Exclusif

Exercice d'OU-Exclusif comme une régression

Une approche alternative dans l'implémentation de l'opérateur logique OU-Exclusif consiste à le représenter sous la forme d'une régression. Dans ce cas, le réseau devra prédire les valeurs de sortie souhaitées plutôt que la classe, contrairement au cas du classifieur.

7. - L'entité disponible à cet objet dans la bibliothèque Sklearn est celle du MLPRegressor :

```
#Importation de la définition du perceptron multi-couche pour la régression
from sklearn.neural_network import MLPRegressor
```

   - Définissez une instance en indiquant un minimum de 2 couches cachées, une fonction d'activation 'tanh' et l'utilisation de la méthode d'optimisation 'lbfgs'.
   - Entraînez le réseau (fit) en indiquant les entrées et la valeur de sortie correspondante.
   - Estimez le score et vérifiez les prédictions.

Exercice d'application robotique

L'objectif est de développer une solution basée sur un réseau de neurones du type MLP capable d'assurer une navigation autonome du robot Thymio évoluant dans un labyrinthe. Cet exercice a pour but de valider le processus d'apprentissage supervisé en utilisant l'ensemble des capteurs de proximité.

8. À l'aide d'un contrôle manuel du robot en utilisant les touches du clavier ou d'un réseau travaillé en cours d'évitement d'obstacles, faites évoluer le robot dans le labyrinthe et collectez des données provenant des sept capteurs de proximité ainsi que les consignes de vitesse des moteurs.

   (a) Stockez l'ensemble des données dans des listes, puis enregistrez-les dans un fichier HDF (Hierarchical Data File) à l'aide de l'utilitaire h5py, comme illustré dans l'exemple ci-dessous :

```
# Open an HDF5 file for writing
with h5py.File("dataset_webots.hdf5", "w") as f:
    array1  =  np.array(trial_commands)
    array2 =   np.array(trial_scans)
    array3 =   np.array(trial_proximeters)
    # Create the main dataset (can be empty or contain additional data)
    commands_dataset    = f.create_dataset('thymio_commands', data=array1)
    scans_dataset       = f.create_dataset('thymio_scans',    data=array2)
    proximeters_dataset = f.create_dataset('thymio_prox',     data=array3)
# Close the HDF5 file
f.close()
```

9. La phase d'apprentissage supervisée commence par la lecture de la base de données crée (fichier hdf5).

   (a) Avant d'initier la fonction d'entraînement, assurez-vous que les données en entrée et en sortie du réseau ont été normalisées. Par exemple, les mesures des capteurs de proximité doivent être comprises dans l'intervalle $[0, 1]$, et les consignes des moteurs dans l'intervalle $[-1, 1]$.

   (b) A l'aide de la fonction Train-Test-Split de l'utilitaire Scikit-learn obtenez un partitionnement de données (voir options de la fonction) :

```
#%% Proximeters Split dataset
from sklearn.model_selection import
train_test_split prox_x_train, prox_x_test, prox_y_train, prox_y_test = train_test_split(thymio_proximeters, thymio_commands)
```

   (c) En utilisant le partitionnement des données, effectuez l'entraînement du MLP en définissant un réseau à deux couches cachées. Cette topologie pourra évoluer en fonction des résultats obtenus (phase de réglage - tunning).

   (d) A la convergence, évaluez le réseau (fonction score) sur la partition de données prévue à cet effet.

   (e) Si le processus d'entraînement est satisfaisant (score > 0.97) sauvegardez le modèle entraîné. Utilisez l'utilitaire pickle pour l'enregistrement du MLP dans une fichier binaire.

```
import pickle
filename = 'ai_controller_model_hyper_prox.model'
pickle.dump(controller, open(filename, 'wb'))
```

10. Enfin, chargez le modèle et implémentez le perceptron multicouche (MLP) de régression pour inférer les consignes de vitesse en fonction des mesures de proximité. Validez la navigation autonome du robot Thymio.