MIAM (ICM1A-ICM2A), UP1, TP3 : Réseaux de Neurones

Ce travail pratique numérique est la suite de celui portant sur la régression. Les mêmes données seront utilisées, et le but recherché est le même, à savoir la construction d'un modèle de régression permettant de prédire la conductivité d'un milieu fibreux homogénéisé en fonction de quatre paramètres de sa micro-structure.

Ce TP est évalué. Le travail est à faire individuellement. **Si vous choisissez ce TP** : il faudra déposer, au plus tard deux semaines après la fin de la séance ≪ réseaux de neurones ≫ (dernière séance de TP), le code que vous aurez fait, ainsi qu'un compte-rendu sous format `pdf`. Il est également possible de rendre les deux simultanément dans un notebook jupyter. Dans ce cas, il faut rendre à la fois la version `ipynb` et `pdf`. **Si vous ne choisissez pas ce TP** : vous devrez faire apparaître en préambule du compte-rendu du TP que vous choisissez un paragraphe expliquant la démarche de ce TP.

La forme est importante. Dans un contexte ≪ ingénieur ≫, un code mal commenté, mal organisé ou avec des noms de fonctions ou de variables mal choisis a vocation à être une source de stress et d'erreurs pour les contributeurs qui s'y pencheront après vous. Dans le contexte présent, un tel code a simplement vocation à être une source de points en moins pour vous. Il en va de même pour le compte-rendu.

Forme attendue pour le compte-rendu : Il doit pouvoir se lire indépendamment du sujet. Il comprendra une rapide introduction expliquant l'objet d'étude, et surtout présentera les résultats de l'exécution de votre code (images ou matrices si elles sont petites), ainsi qu'une description de ces résultats (paramètres utilisés etc) et un commentaire sur ces résultats. Le compte-rendu est organisé en sections (les mêmes que dans le sujet) et se finit par une conclusion, comme rappelé en fin de document. Si le compte-rendu n'est pas un notebook jupyter, le code ne doit pas y apparaître. Il faut également faire apparaître un court résumé des TPs non-choisis dans le compte-rendu du TP choisi.

Mise en place

En plus de ce sujet, vous aurez besoin du support de cours pour pouvoir mener à bien le travail demandé. Vous avez également à disposition une bibliothèque de fonctions regroupées dans le fichier `nn_ressources`. La bibliothèque du TP précédent, `regression_ressources` sera elle-aussi utilisée. Ces bibliothèques devront être importées dans le code que vous créerez, de préférence sous un diminutif bien choisi.

Comme on l'a vu à l'occasion du TP numérique précédent, la normalisation, qui consiste à transformer les données de façon affine pour qu'elles soient comprises entre 0 et 1, a une grande importance en régression. Dans le cas des réseaux de neurones, cette étape est même en pratique indispensable. La fonction `normalize_data` effectue automatiquement cette transformation.

Par ailleurs, comme dans le TP précédent, la fonction `crossValSplit` peut être utilisée pour séparer une base de donnée de construction du modèle de la base de validation.

À faire : Préparer les données à l'aide des deux fonctions pré-mentionnées.

1 Implémentation du réseau de Neurones

Dans cette partie, on implémente (partiellement) un réseau de neurones ≪ maison ≫, dont on va pouvoir comparer le comportement avec `tensorflow`. Pour faciliter l'implémentation, la fonction `BackPropagateNN` est donnée. Elle calcule le gradient de la fonction-coût des moindres carrés à l'aide de la méthode de rétro-propagation. On ignorera son paramètre facultatif `lastLayerLinear` jusqu'à la dernière question bonus de ce sujet.

Remarquons que la fonction donnée n'est pas fondamentalement plus difficile à implémenter que les deux qui vous sont demandées, mais est légèrement plus technique, notamment en ce qui concerne la gestion du terme de biais.

On donne dans la section `main` de `nn_ressources.py` la fonction « d'emballage » (wrapper) `FwdPropagateNN`, qui gère l'agencement des différents vecteurs de la base de donnée, et qui fait appel à une fonction non-implémentée, `FwdPropagateNN_single` pour évaluer le réseau de neurones pour un seul vecteur $\mathtt{x}_n$ de la base de données.

On rappelle dans l'algorithme 1 la procédure de propagation directe dans un réseau de neurones.

Algorithm 1: Évaluation d'un réseau de neurones : EvaluateNN($\{w_{sij}\}_{s=1..S,i=1..u_{s-1},j=1..u_s}$, $\mathtt{x}$)

$\{w_{sij}\}_{s=1..S,i=1..u_{s-1},j=1..u_s}$ fixés
$\mathtt{x}$ donné
$\beta_0 = \mathtt{x}$
$\beta_0^{u_0+1} = 1$ (terme de biais)
for $s = 1..S$ **do**
  $\alpha_s = W_s \beta_{s-1}$
  $\forall i = 1..u_s,\ [\beta_s]_i = g_s\left([\alpha_s]_i\right)$
  $[\beta_0]_{u_s+1} = 1$ (terme de biais)
end
$\mathtt{z} = \beta_S$

À faire : Construire, sur le modèle de `BackPropagateNN_single`, une fonction de propagation directe implémentant l'algorithme 1. Il y a quelques choix d'implémentation imposés par la façon dont `FwdPropagateNN` est implémentée :

- Pour que le reste du code fonctionne correctement, `x` doit être le premier argument de la fonction.
- dans `BackPropagateNN`, les multi-vecteurs de données sont transposés par-rapport à ce qui a été fait jusqu'ici. Il est conseillé d'utiliser le même format dans la fonction demandée.
- Lors du stockage de la liste des vecteurs β, il ne faut pas stocker le terme de biais
- Le vecteur `x` n'est pas le premier élément de la liste `betalist`. On ne le stocke pas dans cette liste.

Remarque. *La fonction `FwdPropagateNN` est donnée dans la section `main` du fichier de ressources, ce qui signifie qu'elle n'est pas définie lorsque ce script est appelé via la commande `import`. Pour avoir accès à cette fonction, il faut la copier dans votre propre script. Remarquons au passage que dans la plupart des cas, il est aberrant de définir des fonctions dans la section `main`. Cette fonction est ici dans l'unique but d'être copiée dans votre script.*

À faire : Implémenter l'algorithme du gradient pour optimiser les poids $\{W_s\}_{s=1..S}$ du réseau de neurones. L'initialisation sera faite en utilisant l'une des fonctions données `InitializeNN_ones` ou `InitializeNN_rand`.

Bonus : L'une des fonctions qui sont données s'appelle `BackPropagateNN_Block`. Il s'agit d'une version optimisée de `BackPropagateNN`, utilisant notamment la notion d'opérations par blocs. Sur son modèle, implémenter la fonction de propagation directe par blocs, et utiliser ces deux fonctions optimisées dans l'algorithme d'optimisation. Comparer les temps de calculs à l'aide de la fonction `time` du module du même nom.

2 Utilisation de `tensorflow`

Dans le document de ressources, quelques lignes de code entre les commentaires `BEGIN TENSORFLOW` et `END TENSORFLOW` réalisent l'implémentation d'un réseau de neurones à une seule couche interne à l'aide de la bibliothèque `tensorflow`.

À faire : Installer `tensorflow` sur votre machine à l'aide de `pip3` (ou tout autre moyen) et l'importer dans votre script.

À faire : Dans le compte-rendu, afficher les quelques lignes de code sus-mentionnées en expliquant, en lien avec le cours, ce que fait chacune de ces lignes. *Cette question vous demande de faire une exception à la règle générale selon laquelle il ne faut pas reproduire de code dans un compte-rendu.*

À faire : Copier ces lignes de code dans votre script. À l'aide de la fonction fournie `modelPlotter`, confronter le modèle à l'ensemble des données (groupe de validation y-compris).

Si le code a été copié tel quel, le résultat n'est probablement pas très bon.
À faire : Sur la base donnée, modifier le modèle afin de le rendre plus précis. On s'intéressera notamment au nombre d'itérations de l'algorithme de minimisation, à la largeur des couches de neurones, à leur nombre, et à la fonction d'activation de la couche de sortie (ligne commentée à la fin de la définition du modèle).

Bonus : `tensorflow` permet de choisir parmi une bibliothèque impressionnante de fonctions d'activation, fonctions-coûts et méthodes d'optimisation. Explorer ces possibilités et voir leur impact sur la précision du modèle.

3 Test et comparaison des modèles

À faire : Évaluer le modèle construit à l'aide de la fonction `modelPlotter` (du TP de régression), ainsi que de la validation croisée. Observer également la décroissance de la fonction-coût. Si le même format de données d'entrée que pour `BackPropagateNN` a été utilisé pour la propagation directe, le vecteur d'entrée doit être transposé pour être lu : si nécessaire, on définira une fonction « wrapper » permettant de transposer ce vecteur d'entrée, et c'est cette fonction wrapper qui sera envoyée dans `modelPlotter`.

À faire : Visualiser l'impact du nombre et de l'épaisseur des couches sur la vitesse de convergence et sur l'erreur de validation croisée à convergence. On rappelle qu'il est possible de fixer le germe aléatoire dans la fonction `crossValSplit`.

À faire : Obtient-on les mêmes résultats (ou des résultats proches) de ceux obtenus avec `tensorflow` ?

Bonus : En pratique, on constate que la régression est de meilleure qualité quand la fonction d'activation de la dernière couche est linéaire, et non-pas sigmoïde. Ajouter une option permettant de construire un tel réseau de neurones, et vérifier que son comportement est conforme avec celui de `tensorflow`. On pourra à-nouveau prendre modèle sur ce qui a été fait pour la fonction `BackPropagateNN`.

Conclusion

À faire : Proposer une conclusion synthétique, claire et pertinente à ce travail. On s'interrogera notamment (mais pas nécessairement exclusivement) sur les questions suivantes :

- L'impact des différents hyper-paramètres d'un réseau de neurones
- Les moyens de choisir ces hyper-paramètres
- L'interprétabilité d'un modèle par réseaux de neurones : par-rapport à un modèle de régression affine, nous donne-t-il autant d'information qualitative sur l'impact des différents paramètres ?