Le génie pour l'industrie

Laboratoire 2 ELE-767

Apprentissage machine en intelligence artificielle

Perceptron multicouches (MLP)

Saddat Mohammad, Cavero-Linares Gabriella et Frija-Altarac Liam

MOHS27119206
CAVG08599602
FRIL04049603

Rapport Présenté à
Chakib Tadj

17/03/2019

Introduction **3**

Étapes de développement **3**

Fonctions d'activation utilisées **4**

Contraintes complémentaires ajoutées **4**

Contrainte complémentaire 1: Interface graphique 4

Contrainte complémentaire 2: Nombre de couche cachée 4

Contrainte complémentaire 3: Améliorations apportées 5

Amélioration **5**

Différentes simulations effectuées **5**

Évolution de l'erreur **7**

a) L'apprentissage 7

b) La validation croisée 8

c) La généralisation 9

Analyse du comportement du réseau de neurones **10**

10

Recommandations **10**

Conclusions **10**

Introduction

Lors de ce laboratoire, nous avons fait une application, à l'aide du langage de programmation Python, permettant de construire un réseau de neurone pouvant reconnaître la parole. L'apprentissage est fait par rétro-propagation du gradient de l'erreur.

Étapes de développement

1. Prérequis :
La première étape de développement a été de lister tous les prérequis, les concepts à utiliser et les contraintes du laboratoire.

2. Choix de Structure:
Ensuite, nous avons décidé comment allions-nous implémenter la structure d'un MLP. Dans notre implémentation, le MLP a été décomposé de sorte que la plus petite unité soit une couche. Les structures sont faites de sorte avoir une certaine modularité et permettre ainsi à l'utilisateur de créer un réseau ayant les caractéristiques voulues.

3. Modélisation de l'apprentissage:
Lors de la modélisation de l'apprentissage, nous avons implémenté la phase d'activation, de calcul d'erreur, de correction et d'actualisation. Lors de cette étapes, il n'a suffit que d'intégrer les fonctions vu durant le cours dans le code sous-formes de fonctions.

4. Confirmation de calculs:
Afin de confirmer les résultats de calculs faits par notre application, nous avons refait les calculs de chaque phase à main et comparer les résultats avec ceux de l'application.

5. Validation croisée et test de généralisation:
Lors de l'implémentation de la validation croisée, nous avons intégré une fonction de calcul de performance afin de confirmer le bon fonctionnement de notre réseau. Ensuite, nous avons essayé l'application avec certains problèmes de MPL vues en classe et confirmer les résultats. Pour finir nous, nous avons fait un test de généralisation avec notre réseau et confirmer son bon fonctionnement.

6.Interface:
La dernière étape a été de de créer un interface GUI permettant à l'utilisateur de choisir les caractéristiques principales de son réseau de neurone sans s'aventurer dans le code.

Fonctions d'activation utilisées

Les trois fonctions d'activation que nous avons intégrés dans notre MLP sont le sigmoïde, la tangente hyperbolique et le sinusoïdale.

- Sigmoïde : $S(x) = \frac{1}{1+e^{-x}}$
- Tangente Hyperbolique : $tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- Sinus : sin(x)

Contraintes complémentaires ajoutées

Contrainte complémentaire 1: Interface graphique

Nous avons implémenté une interface graphique sur web. Elle nous permet de sélectionner les fichiers d'entraînement, tests, validations et configurations. On peut sélectionner les configurations voulues tel que le nombre de couches cachées, le nombre neurones par couche, les sets de données, l'eta, les fonctions d'activations, le nombre d'époques, eta adaptatif et ajout de bruit.
Nous pouvons aussi visualiser la performance de notre MLP pendant l'apprentissage.

Figure 1. Interface web

Contrainte complémentaire 2: Nombre de couche cachée

Nous avons implémenté un choix de nombre de couche cachée variable. L'utilisateur peut choisir le nombre de couche ainsi que le nombre de neurones pour chacune de ses couches.

Contrainte complémentaire 3: Améliorations apportées

Lors de l'apprentissage, nous avons implémenté le "**adaptive learning rate**" . Cette méthode consiste à faire diminuer le facteur de performance après chaque époque. Les poids vont donc avoir de grandes variations lors des premières époques ,mais les variations diminuent linéairement. Il y aura donc une diminution de la vitesse d'apprentissage. Un réseau a besoin d'apprendre rapidement au début, mais, plus il se rapproche de sa version optimal moins il a besoin d'apprendre. Cette méthode permet d'avoir une meilleure performance.

Amélioration

Nous avons l'option de rajouter du bruit à nos données. Cela nous permet,donc, de doubler le nombre de donnée ,avoir un meilleur apprentissage et ainsi avoir une meilleure performance.

Différentes simulations effectuées

Nombre d'entrées:
Nous avons simulé avec 40, 50 et 60 sets de base de données. Nous avons comparé la performance lors de la validation croisée et constaté que la performance est meilleur avec un set de 60 données. Voici le tableau résumant les performances obtenues.

ETA	Set de donnée	Neurones / CC	Fct D'activation	nb_epoche	Eta adaptif	Ajout de bruit	Performance data_train	Performace data_vc	Performance data_test
0.1	40	50	Sigmoid	10	Faux	Faux	99%	75.83%	76.25%
0.1	50	50	Sigmoid	10	Faux	Faux	99.50%	74.16%	73.20%
0.1	60	50	Sigmoid	10	Faux	Faux	98%	80%	73%

Tableau 1. Comparaison du nombre d'entrée

Nombre de couches cachées:
Nous avons simulé notre réseau avec 60 paquets d'entrées avec une à 3 couches cachées. La meilleure performance de validation croisée est obtenue avec une seule couche cachée.

ETA	Set de donnée	Neurones / CC	Fct D'activation	nb_epoche	Eta adaptif	Ajout de bruit	Performance data_train	Performace data_vc	Performance data_test
0.1	60	50	Sigmoid	10	Faux	Faux	98%	80%	73%
0.1	60	50,50	Sigmoid	10	Faux	Faux	73%	63%	63%
0.1	60	50,50,50	Sigmoid	10	Faux	Faux	10%	11%	12%

Tableau 2. Comparaison du nombre de couches cachées

Nombre de neurones par couches cachées:
Nous avons fait varier le nombre de neurones par couches cachées entre 40 et 100 .Nous constatons que la performance lors de la validation croisée est la meilleur lorsque l'on a 50 neurones dans la couche cachée.

ETA	Set de donnée	Neurones / CC	Fct D'activation	nb_epoche	Eta adaptif	Ajout de bruit	Performance data_train	Performace data_vc	Performance data_test
0.1	60	40	Sigmoid	10	Faux	Faux	99%	78%	74%
0.1	60	50	Sigmoid	10	Faux	Faux	98%	80%	73%
0.1	60	60	Sigmoid	10	Faux	Faux	99%	74%	77%
0.1	60	70	Sigmoid	10	Faux	Faux	98%	78%	75%
79	60	100	Sigmoid	10	Faux	Faux	99%	79%	76%

Tableau 3. Comparaison du nombre de neurone par couches cachées

Nombre de sorties: Nous avons fixé le nombre de sorties à 10, puisque nous avons 10 sorties différentes. Chaque valeur possède son bit. Par exemple, si le réseau détecte 8 on obtiendra en sortie 0000000100. Cette méthode est la méthode la plus répandue dans le web.

Fonctions d'activations: Nous avons aussi testé nos 2 fonctions d'activations pour tous les cas. Nous avons conclu que la fonction sigmoïde est la plus performante lors de la validation croisée.

ETA	Set de donnée	Neurones / CC	Fct D'activation	nb_epoche	Eta adaptif	Ajout de bruit	Performance data_train	Performace data_vc	Performance data_test
0.1	60	50	Sigmoid	10	Faux	Faux	99%	78%	74%
0.1	60	50	Tanh	10	Faux	Faux	98%	76%	77%

Tableau 4. Comparaison fonctions d'activations

Nombre d'époques: Lors de ce test, nous avons utilisé un taux d'apprentissage adaptatif et observé qu'il y avait un plateau lorsque l'on atteint 15 époques .Dans la figure suivante, nous pouvons constater celui-ci.

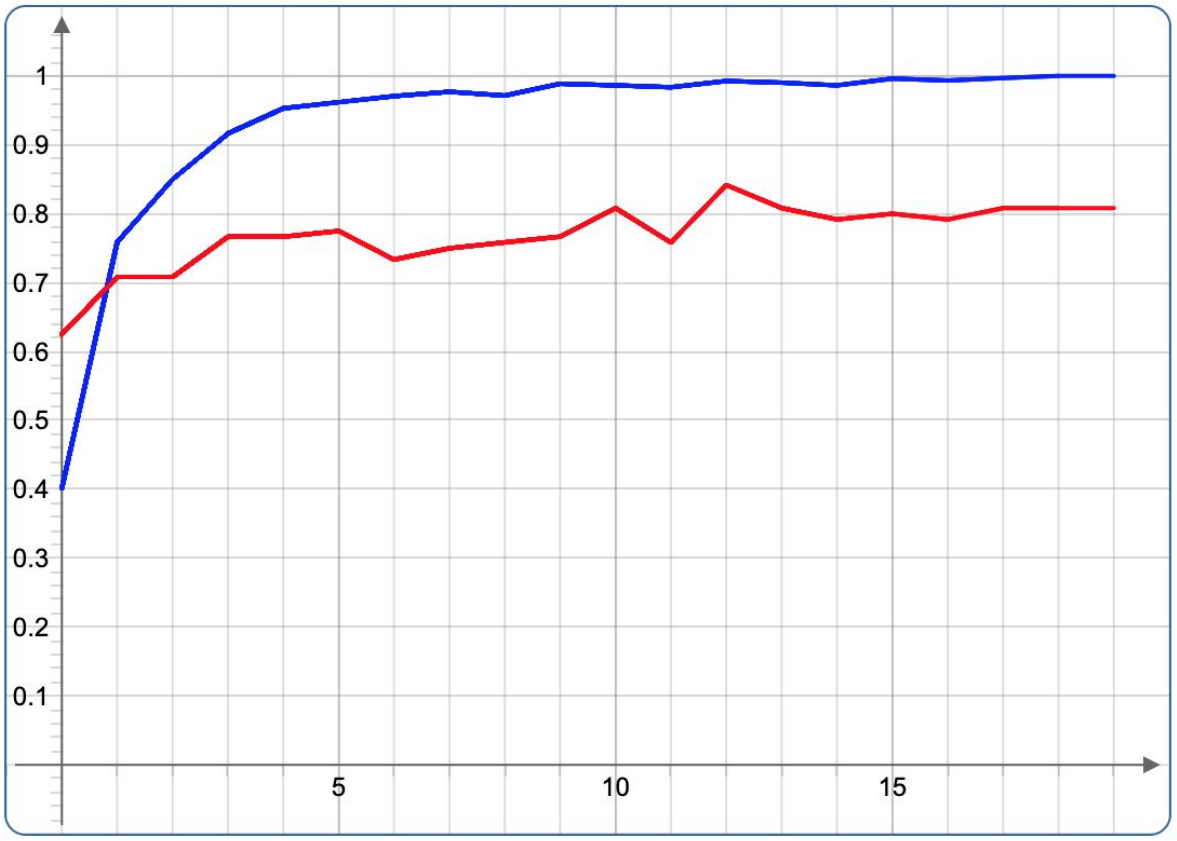

Figure 2. En rouge: performance de la validation croisée et en bleu celle de l'apprentissage

Évolution de l'erreur

a) L'apprentissage

ETA	Set de donnée	Neurones / CC	Fct D'activation	nb_epoche	Eta adaptif	Ajout de bruit	Performance data_train
0.1	60	50,50,50	Sigmoid	10	Faux	Faux	10%
0.1	60	50,50	Sigmoid	10	Faux	Faux	73%
0.1	60	50	Tanh	10	Faux	Faux	98%
0.1	60	70	Sigmoid	10	Faux	Faux	98%
0.1	60	50	Sigmoid	10	Faux	Faux	98%
0.1	60	50	Sigmoid	10	Faux	Faux	98%
0.1	60	50	Sigmoid	10	Faux	Faux	98%
0.1	60	60	Sigmoid	10	Faux	Faux	99%
0.1	40	50	Sigmoid	10	Faux	Faux	99%
0.1	60	40	Sigmoid	10	Faux	Faux	99%
0.1	60	50	Sigmoid	10	Faux	Faux	99%
0.1	60	100	Sigmoid	10	Faux	Faux	99%
0.1	50	50	Sigmoid	10	Faux	Faux	99.50%

Nous constatons que premièrement, augmenter le nombre de couche cachées diminuent la performance de l'entraînement. Ensuite, peu importe la fonction d'activation utilisée la performance d'entraînement n'est pas affectée. Changer le nombre de neurones dans la couche cachée n'affecte également pas la performance d'entraînement. Le set de 50 donnée permet d'avoir la meilleur performance.

Tendance générale de la courbe de performance d'apprentissage:
Nous constatons que la courbe de performance d'apprentissage évolue de façon logarithmique et tant vers 98% pour les configurations a une seule couche.

b) La validation croisée

ETA	Set de donnée	Neurones / CC	Fct D'activation	nb_epoche	Eta adaptif	Ajout de bruit	Performace data_vc
0.1	60	50,50,50	Sigmoid	10	Faux	Faux	11%
0.1	60	50,50	Sigmoid	10	Faux	Faux	63%
0.1	60	60	Sigmoid	10	Faux	Faux	74%
0.1	50	50	Sigmoid	10	Faux	Faux	74.16%
0.1	40	50	Sigmoid	10	Faux	Faux	75.83%
0.1	60	50	Tanh	10	Faux	Faux	76%
0.1	60	70	Sigmoid	10	Faux	Faux	78%
0.1	60	40	Sigmoid	10	Faux	Faux	78%
0.1	60	50	Sigmoid	10	Faux	Faux	78%
0.1	60	100	Sigmoid	10	Faux	Faux	79%
0.1	60	50	Sigmoid	10	Faux	Faux	80%
0.1	60	50	Sigmoid	10	Faux	Faux	80%
0.1	60	50	Sigmoid	10	Faux	Faux	80%

Nous constatons que premièrement, augmenter le nombre de couche cachées diminuent la performance de la validation croisée. Avoir 60 comme set de données performe mieux qu'en avoir 40 ou 50. Si l'on analyse la configuration du réseau contenant une couche cachée de 50 neurones et un set de 60 données, on constate qu'utiliser la fonction d'activation tangente hyperbolique au lieu du sigmoïde diminue un peu la performance. Changer le nombre de neurones par couche donne des performances variés mais la meilleure performance est avec 50. L'analyse plus détaillée de la validation croisée a été effectué dans la section "**différentes simulations effectuées**".

Tendance générale de la courbe de performance VC:
Nous constatons que la courbe veut suivre celle de la performance d'apprentissage .Ensuite, la courbe tendra vers une certaine valeur avec de petites variations.

c) La généralisation

ETA	Set de donnée	Neurones / CC	Fct D'activation	nb_epoche	Eta adaptif	Ajout de bruit	Performance data_test
0.1	60	50,50,50	Sigmoid	10	Faux	Faux	12%
0.1	60	50,50	Sigmoid	10	Faux	Faux	63%
0.1	60	50	Sigmoid	10	Faux	Faux	73%
0.1	60	50	Sigmoid	10	Faux	Faux	73%
0.1	60	50	Sigmoid	10	Faux	Faux	73%
0.1	50	50	Sigmoid	10	Faux	Faux	73.20%
0.1	60	40	Sigmoid	10	Faux	Faux	74%
0.1	60	50	Sigmoid	10	Faux	Faux	74%
0.1	60	70	Sigmoid	10	Faux	Faux	75%
0.1	60	100	Sigmoid	10	Faux	Faux	76%
0.1	40	50	Sigmoid	10	Faux	Faux	76.25%
0.1	60	50	Tanh	10	Faux	Faux	77%
0.1	60	60	Sigmoid	10	Faux	Faux	77%

Nous constatons que premièrement, augmenter le nombre de couche cachées diminuent la performance de la généralisation. Avoir 60 comme set de données performe mieux que les autres cas et les données 40 performent mieux que 50. Utiliser la fonction d'activation tangente hyperbolique au lieu du sigmoïde augmente un peu la performance. Changer le nombre de neurones par couche donne des performances variés, mais la meilleure performance est avec 60.

Tendance générale du test de généralisation:
La performance de notre réseau variera entre 70% et 80%, ce qui selon nous est une performance assez bonne.

Analyse du comportement du réseau de neurones

Si nous récapitulons, notre réseau le plus performant contient 60 sets de données en entrée , une couche cachée de 50 neurones,utilise 15 époques, utilise la fonction d'activation sigmoïde et utilise l'apprentissage adaptatif. On constate que lorsque l'on fait l'apprentissage avec des données non-bruitées et ensuite avec des données bruitées la performance augmente , mais stagne à une certaine performance. La performance de VC avec un entraînement uniquement avec les données non-bruitées est de 73%. Elle passe à 78% lorsque l'on l'entraîne aussi avec les données bruitées.

Recommandations

- Apprentissage incrémentale.
- Garder les couches qui nous donnent la meilleure performance.
- D'autres fonctions d'activation, (eg.: RELU, LeakuRELU etc)

Conclusions

En conclusion, à travers ce laboratoire nous avons réalisé que l'entraînement d'un réseau de neurones est un jeu de devinette pour obtenir les paramètres optimales pour ensuite correctement classifier des données. Cependant, il existe des méthodes pour faciliter la tâche tel que le taux d'apprentissage adaptatif ou bien l'ajout de bruit pour en créer des nouvelles données d'entraînement.

Annexe

mlps_sauvgarde/mlpBest_Sigmoid_et_LeakyRELU.txt

Figure3. Configuration de la meilleure performance