Comprendre les principes de la programmation des réseaux de neurones

Cette activité propose de créer à partir d'un jeu de données un modèle linéaire de neurone artificiel.

Le but est d'illustrer la notion d'apprentissage et quelques fonctions nécessaire à celui-ci.

Pour approfondir et comprendre certaines fonctions proposées, on pourra lire les cours présents sur le web et parmi eux les cours présents sur le site du LABRI (université de Bordeaux https://www.labri.fr/). On pourra aussi regarder un très bon document vidéo: https://www.youtube.com/watch?v=tdelUss-5hY&t=2s

Code capytale de l'activité : 850c-6776709

Notions travaillées :

- Fonctions de coût
- Fonctions d'activation (dérivabilité, non linéarité, adaptation aux classifications etc...),
- Notion d'hyper paramètre (epoch et learning rate...)
- Approche intuitive de la rétropropagation.

Déroulement de l'activité :

Pour effectuer le réglage, on utilise une méthode par descente de gradient.

Le principe est simple. Nous allons calculer les coordonnées de la réponse du modèle (z=a1x+a2y+b en deux dimensions et un calcul matriciel dans l'exemple donné ci-dessous qui peut prendre en charge des modèles en dimension n). Le résultat de ce calcul est passé par une fonction d'activation. Cette sortie permet le calcul d'une fonction de coût (ici une Log loss). Le système va effectuer le réglage des paramètres par la méthode de la descente de gradient sur le principe d'une rétroaction où les coefficients du gradient sont calculés à partir de l'écart entre la valeur calculée et la valeur réelle. Plus les valeurs sont proches, plus les coefficients du gradient sont faibles et moins les évolutions des coefficients sont grandes. L'approximation linéaire génère de faibles variations puisque elle est proche de la sortie réelle.

Pour effectuer ce travail, nous avons fait des choix de **paramètres** (structure (1 neurone)...), et nous allons faire des choix d'**hyperparamètres** (learning rate (ici nommé "pas"), epoch, batches ...).

Etape 1 : **Création des fonctions nécessaires à l'implantation du neurone**

1. Création du jeu de données
2. Initialisation du modèle
3. Création du modèle
4. Création de la fonction coût
5. Création de la fonction gradient en vue du réglage des poids
6. Descente de gradient

Etape 2 : Implémentation du neurone artificiel et apprentissage

1. Test de l'apprentissage à travers la visualisation des valeurs d'erreur
2. Le modèle est construit, test sa fiabilité et utilisation pour effectuer des prédictions
3. Construction et visualisation de la ligne de décision construite à partir du modèle créé

IA - approche de classification et prédiction

Le seul algorithme d'IA présent en NSI est l'algorithme des k-plus proches voisins. Malgré cette limitation, il est possible de sensibiliser aux enjeux propres à l'IA. Le but de cette activité est d'apporter des informations sur le prétraitement de données et le code présenté n'a pas vocation à être enseigné directement en NSI.

Code capytale de l'activité : d404-6776801

Déroulement de l'activité :

1. **Présentation du problème et traitement des données :**

On considère ici le problème suivant : étant donné des mesures médicales, peut-on déterminer si un patient est atteint du diabète ?

Le jeu de données est le Pima Indians Diabetes Database qui fournit 768 mesures associées à des femmes issues de la communauté nord-amérindienne des Pima dont la présence de diabète et d'obésité a été un sujet d'études.

Dans ce jeu de données, il y a 8 champs associés à des valeurs numériques et un champ Outcome valant 1 ou 0 selon que la personne soit atteinte de diabète ou non.

Pour traiter les données, on va

- Passer d'un tableau de tuples nommés à deux tableaux :
  - carac composé de tableaux de valeurs numériques observées ;
  - etiq composé des étiquettes de classification, ici 0 ou 1 selon le fait d'être atteint ou non du diabète ;
- Sélectionner une partie des données pour l'entrainement et une partie pour la vérification de l'efficacité, on parle de jeu d'entraînement et de jeu de tests.

2. **Les classificateurs**

Un classificateur (*classifier* en anglais) va partir d'un couple (carac, etiq) de données d'entrainement, s'initialiser pour s'aligner à ces données (on parle de *fit*) et permettre ensuite de prédire l'étiquette d'une valeur.

Certains classificateurs sont dits *sans modèle*, car ils se contentent d'utiliser le couple de données d'entrainement pour prédire. Pour avoir une ligne de comparaison, on va considérer deux classificateurs sans modèle *naïfs* :

- le classificateur Aléatoire qui renvoie une étiquette aléatoire parmi les étiquettes des données d'entrainement ;

- le classificateur Majoritaire qui renvoie l'étiquette la plus présente.

L'algorithme des **k-plus proches voisins, ou kNN,** est un classificateur sans modèle qui prédit l'étiquette en prenant l'étiquette majoritaire parmi les k données les plus proches. Il y a donc deux paramètres le nombre k, impair pour avoir une étiquette majoritaire dans le cas il y a deux classes, et la distance.

Pour estimer la précision d'un classificateur, on va prédire l'ensemble du jeu de tests et calculer le pourcentage de bonnes prédictions.

Les **réseaux de neurones** sont un classificateur avec modèle. `scikit-learn` propose un tel classificateur `MLPClassifier`, pour *multi-layer perceptron classifier`. Ici, la phase de `fit` va réaliser l'entrainement du réseau. Pour le faible nombre de données, on effectue un nombre conséquent d'itérations (epoch) pour garantir la convergence du modèle.

3. Nettoyage des données :

Si on affiche les données, on se rend compte que la notion de distance n'est pas forcément pertinente, car les échelles des distances sont différentes. Cela installe un biais dans l'expérimentation.

Une manière simple de résoudre ce problème est de redimensionner chaque donnée entre 0 et 1. Cela ne change pas la *forme* de la distribution, notamment cela ne l'étale pas, mais cela lui permet d'être comparée avec les autres.