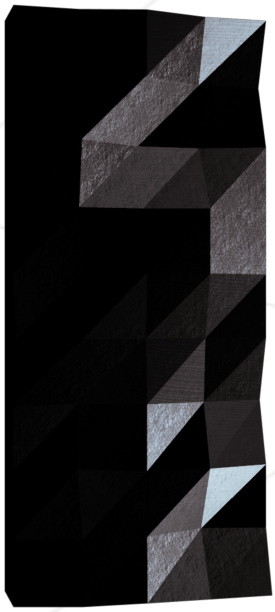


Projet Machine Learning

Multilayer Perceptron

42AI contact@42ai.fr
William Watkins wwatkins@student.42.fr

Résumé: Ce projet est une introduction aux réseaux de neurones artificiels grâce à l'implémentation d'un multilayer perceptron.

42 ARTIFICIAL INTELLIGENCE

Table des matières

Chapitre I

Introduction

Dans le langage de votre choix vous allez mettre en place un `multilayer perceptron` pour déceler si un cancer est malin ou bénin sur un dataset de diagnostics de cancer du sein réalisé au Wisconsin.

I.1 Un peu d'histoire

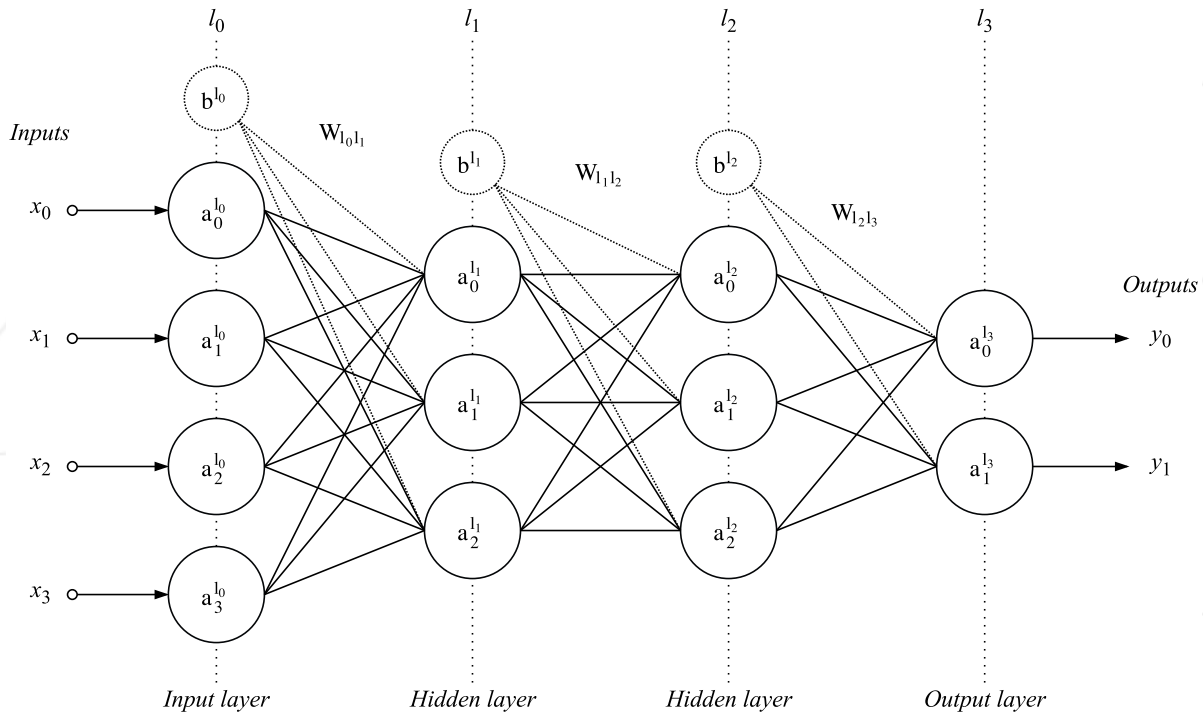
L'apprentissage automatique est un vaste domaine dont les réseaux de neurones ne constituent qu'un petit sous-ensemble. Nous allons néanmoins l'aborder car c'est un outil très puissant qui a ressurgi ces dernières années.

Contrairement à ce que l'on pourrait penser, les réseaux de neurones artificiels existent depuis un certain temps, en effet dès 1948 Alan Turing introduit dans son papier `intelligent machinery` un type de réseau de neurones nommé `B-type unorganised machine` qu'il considère comme le modèle de système nerveux le plus simple possible.

Le perceptron est inventé par Frank Rosenblatt en 1957, c'est un classifieur linéaire d'une seule couche, et c'est aussi un des premiers réseau de neurones implémenté, cependant les résultats ne sont pas concluants et l'idée reste à l'abandon pendant plusieurs années. Il faut attendre les années 70 pour que l'algorithme évolue en `multilayer perceptron` et soit de nouveau utilisé.

I.2 Multilayer perceptron

Le `multilayer perceptron` est un réseau de type `feedforward`, c'est-à-dire que le flot d'informations circule de la couche d'entrée (`input layer`) vers la couche de sortie (`output layer`) défini par la présence d'une ou plusieurs couches cachées (`hidden layers`) ainsi qu'une interconnexion de tous les neurones d'une couche à la suivante.

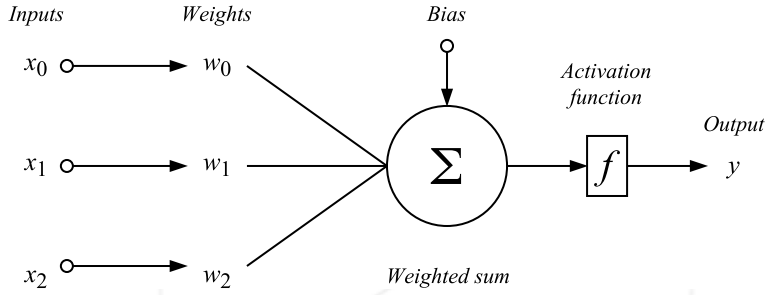


L'exemple ci-dessus est un réseau contenant 4 couches denses. Son entrée prend 4 valeurs et sa sortie retourne 2 valeurs (parfait pour de la classification binaire). Les poids d'une couche à l'autre sont représentés par des matrices à deux dimensions notées $W_{l_j l_{j+1}}$. La matrice $W_{l_0 l_1}$ est de taille $(3, 4)$ par exemple, comme elle contient les poids sur les connexions entre la couche l_0 et la couche l_1.

Le biais est souvent représenté par un neurone spécial qui n'a pas d'entrée et dont la sortie vaut toujours 1, comme un perceptron il est connecté à tous les neurones de la couche suivante (les neurones de biais sont notés b^{l_j} sur le schéma ci-dessus). Il est très souvent utile car il permet de "contrôler le comportement" d'une couche.

I.3 Perceptron

Le perceptron est le type de neurone qui compose le `multilayer perceptron`. Il est défini par la présence d'une ou plusieurs connexions d'entrées, d'une fonction d'activation et d'une unique sortie. Les connexions contiennent un poids (aussi appelé paramètre) qui est appris durant la phase d'entrainement.



Pour obtenir la sortie d'un perceptron deux étapes sont nécessaires, la première consiste à calculer la somme pondérée (`weighted sum`) des sorties de la couche précédente et des poids des connexions entrantes du neurone, ce qui donne

$$weighted\ sum = \sum_{k=0}^{N-1} (x_k \cdot w_k) + bias$$

La seconde étape est d'appliquer une fonction d'activation sur cette somme pondérée, la sortie de cette fonction est la sortie du perceptron, et peut être interprété comme le seuil à partir duquel un neurone est activé (les fonctions d'activations peuvent prendre plusieurs formes, libre à vous de la choisir selon le modèle à entrainer, en voici quelques-unes fréquemment utilisées pour vous donner une idée : sigmoid, hyperboloid tangent, rectified linear unit).

Chapitre II

Objectifs

Ce projet a pour but de vous faire découvrir les réseaux de neurones artificiels, et d'implémenter les algorithmes au coeur du processus d'apprentissage de ces derniers. Vous allez par la même occasion être amené à vous (re)familiariser avec la manipulation de dérivées et de calculs matriciels car c'est des outils mathématiques indispensables à la bonne réalisation du sujet.

Chapitre III

Consignes générales

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.
- Le choix du langage est vôtre, vous n'avez aucune restriction dessus.
- Aucune bibliothèque gérant l'implémentation des réseaux de neurones ou des algorithmes sous-jacents n'est autorisée, vous devez tout coder à partir de zéro, vous avez néanmoins l'autorisation d'utiliser des bibliothèques pour gérer les calculs matriciels et le tracé des courbes d'apprentissage.
- Dans le cas d'un langage compilé, vous devrez rendre un Makefile. Ce Makefile doit compiler le projet, et doit contenir les règles habituelles de compilation présentes. Il ne doit recompiler et relinker le programme qu'en cas de nécessité. Les dépendances doivent aussi être téléchargées/installées grâce au Makefile si nécessaire.
- La norme n'est pas appliquée sur ce projet. Toutefois, nous vous demandons d'être clair et verbeux sur la conception de vos codes sources.

Chapitre IV

Partie obligatoire

IV.1 Avant-propos

Une partie non négligeable de la correction sera basé sur votre compréhension de la phase d'apprentissage, il vous sera demandé d'expliquer à votre correcteur les notions de `feedforward`, `backpropagation` et `gradient descent`. Des points vous seront accordés en fonction de la clarté de vos explications. Je souligne cet aspect car il est important pour la suite de la branche et vous offrira un réel avantage si vous souhaitez continuer dans le domaine.

IV.2 Dataset

Le dataset est fourni en ressources. Il s'agit d'un fichier `csv` de 32 colonnes, la colonne `diagnosis` est le diagnostic associé à toutes les `features` de la ligne actuelle, elle peut avoir comme valeur `M` ou `B` (pour malin ou bénin).

Les features du dataset décrivent les caractéristiques du noyau cellulaire de masse mammaire extraites par aspiration à l'aiguille fine. (pour des informations plus détaillées, rendez-vous ici).

Comme vous pourrez le voir il y a un premier travail de compréhension des données avant de se lancer dans la mise en place de l'algorithme pour pouvoir les classifier. Il est bon de prendre l'habitude de jouer avec ces données en les affichant sur des graphs pour ensuite les manipuler plus aisément.

`Les données du dataset sont brutes et doivent être prétraitées avant d'être utilisées pour l'apprentissage.`

IV.3 Implémentation

Votre implémentation du réseau de neurones doit contenir au moins deux couches cachées (cela vous incite à écrire votre programme de manière un minimum modulaire, bien que ce point ne soit pas évalué pendant la correction c'est une bonne habitude à prendre). Vous devrez aussi mettre en place la fonction `softmax` sur la couche de sortie pour obtenir une distribution probabiliste.

Afin d'évaluer les performances de votre modèle de manière robuste pendant l'entrainement vous allez devoir séparer votre `dataset` en deux parties, une pour l'apprentissage et une pour la validation (les données de validation sont celles sur lesquelles on effectue une prédiction et qui nous permettent de déterminer les performances du modèle face à des exemples inconnus).

Dans le but de visualiser les performances de votre modèle pendant l'entrainement, vous allez devoir afficher à chaque époque la métrique d'entrainement et de validation, exemple :

```
epoch 39/70 - loss: 0.0750 - val_loss: 0.0406
epoch 40/70 - loss: 0.0749 - val_loss: 0.0404
epoch 41/70 - loss: 0.0747 - val_loss: 0.0403
```

Vous allez aussi devoir implémenter l'affichage des courbes d'apprentissage à la fin de l'entrainement (vous êtes libres d'utiliser n'importe quelle bibliothèque).

IV.4 Rendu

Vous allez devoir rendre deux programmes, le premier pour la phase d'entrainement, et le second pour la phase de prédiction (ou vous pouvez aussi en rendre un seul qui prendra un paramètre pour passer d'une phase a l'autre) :

- Le programme d'entrainement utilisera la `backpropagation` et la descente de gradient pour apprendre sur les données d'entrainement et devra sauvegarder le modèle (la topologie et les poids du réseau) à la fin de son exécution.
- Le programme de prédiction devra charger les poids du réseau entrainé, faire une prédiction sur un set donné puis l'évaluer en utilisant la fonction d'erreur d'entropie croisée binaire :

$$E = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log p_n + (1-y_n)\log(1-p_n)]$$

Une partie de la notation va s'effectuer sur la phase d'apprentissage et les performances du modèle entrainé. Comme beaucoup de facteurs aléatoires entrent en jeu (l'initialisation des poids et des biais par exemple), vous avez l'autorisation d'utiliser un seed pour obtenir un résultat répétable.

Chapitre V

Partie bonus

La partie bonus ne sera évalué que si la partie obligatoire a été parfaitement réalisée. Vous êtes libres d'implémenter à votre programme les fonctionnalités que vous jugez intéressantes, voici néanmoins une liste non exhaustive de bonus :

- Une fonction d'optimisation plus complexe (par exemple : nesterov momentum, RMSprop, Adam, ...).
- Un affichage de plusieurs courbes d'entrainements sur le même graph (très pratique pour comparer plusieurs modèles).
- Une sauvegarde de l'historique des métriques obtenues durant l'entrainement.
- De l'early stopping.
- Évaluer l'apprentissage avec plusieurs métriques.

Chapitre VI

Rendu et peer-évaluation

Rendez-votre travail sur votre dépot `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance.