Réseaux de Neurones : reconnaître des chiffres

Les réseaux de neurones peuvent être utilisés pour faire de la reconnaissance de forme. L'objet de ce projet est d'apprendre à reconnaître des images de chiffres. Pour traiter de manière automatique des documents papiers structurés comme des chèques ou des lettres, il est utile d'avoir un outil pour reconnaître les chiffres écrits à la main.

Pour ce projet, nous allons utiliser une base d'exemples appelée MNIST qui est couramment utilisée pour tester des algorithmes d'apprentissage automatique. Pour plus de détails, veuillez consulter la page yann.lecun.com/exdb/mnist/. La base contient 60,000 exemples, chaque exemple est constitué d'une image d'un chiffre écrit à la main, et d'une étiquette qui correspond au chiffre écrit. Les images sont de basse résolution (28x28 pixels), mais cela suffit pour la reconnaissance. La base contient aussi un autre ensemble de 10,000 exemples avec leur étiquette et qui nous servira de base pour tester les performances de notre réseau (l'idée est donc d'apprendre sur 60,000 exemples, et de tester sur 10,000 qui n'ont pas été vu pendant l'apprentissage).

Le code pour lire les données ainsi que l'architecture basique est fournie. Vous devez réaliser les tâches suivantes :

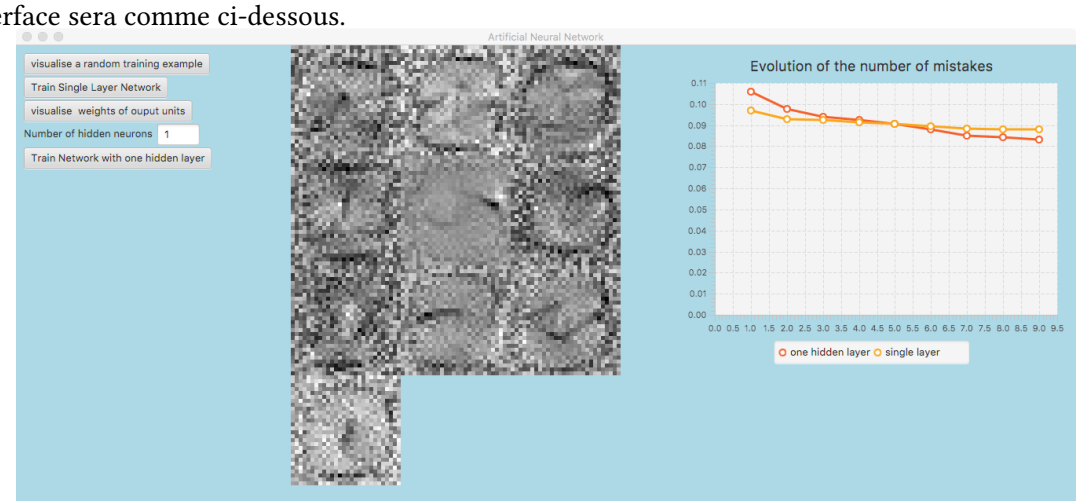
1. Implémenter un réseau mono couche pour apprendre à reconnaître les chiffres. Votre réseau aura donc une couche de neurones d'entrée qui vont simplement accéder aux données des images, et une couche de 10 neurones de sortie : si le neurone de sortie numéro i est activé, le chiffre est i. Chaque neurone de sortie est connecté à chaque neurone d'entrée.
2. Implémenter un réseau avec une couche de neurones cachée. Votre réseau aura donc trois couches : la couche de neurones d'entrée, un couche cachée de k neurones, et une couche de 10 neurones de sortie. Chaque neurone de sortie est connecté à chacun des k neurones de la couche cachée, et chaque neuronne de la couche cachée est connecté avec tous les neurones d'entrée.
3. Comparer les performances de ces deux versions en terme du nombre d'erreurs.

Code existant

Nous livrons un code qui va lire les données MNIST et qui va lancer une interface graphique. A l'aide de cette interface, vous pourrez :

- visualiser une image de la base MNIST au hasard (à chaque nouvel appel, une nouvelle image sera tirée au hasard).
- lancer l'apprentissage du réseau à une seule couche.
- visualiser le poids des neurones de sortie. Ce bouton est surtout destiné à l'apprentissage du réseau à une couche : chaque pixel représentera la valeur relative du poids associé à chaque pixel de l'image d'entrée (plus la case est foncée, plus le poids est grand). Attention, la méthode utilise une normalisation pour calculer l'intensité. La méthode marchera aussi pour d'autres types de réseau, mais si la dernière couche cachée contient k neurones, il n'y aura que k poids réprésentés !
- spécifier le nombre de neurones dans la couche cachée
- lancer l'apprentissage d'un réseau à une couche cachée.

Après chaque apprentissage, un graphique montrant l'évolution du nombre d'erreurs commises sur l'ensemble de tests sera automatiquement mis à jour. Si vous entrainez un nouveau réseau, une nouvelle courbe viendra s'ajouter au graphique. L'interface graphique se trouve dans la classe `MnistData`, et l'interface sera comme ci-dessous.



Les classes `Input` et `Output` modélise les données :

- `Input` modélise l'entrée, pour notre application, il s'agira donc des valeurs des 28×28 pixels.
- `Output` modélise la sortie. Pour notre application, il s'agit de chiffre (de zéro à neuf). Cependant, comme on a choisit comme couche de sortie 10 neurones, chaque neurone i indiquant si le chiffre est i, une `Output` sera en fait un tableau de 10 valeurs.

La classe `Neuron` modélise un neurone. Chaque neurone n aura deux listes : la liste des neurones "sources", i.e. les neurones qui enverront les entrées à n et la liste des neurones "destinataires", i.e. les neurones qui utilisent la valeur de sortie de n. Chaque neurone a aussi sa fonction d'activation (On vous donne les classes codant les fonctions linéaire (`Linear`), logistique (`Sigmoid`) et tangeante hyperbolique (`Tanh`).

Les poids associés à chacune des entrées du neurone sont représentés par un `Map` qui à chaque neurone d'entrée associe la valeur du poids associé. Ce n'est pas la meilleure structure pour avoir un code performant au niveau du temps d'exécution, mais cela permettra peut-être un développement un peu plus simple (pour faire plus rapide, on peut utiliser un tableau de double).

Chaque neurone possèdera aussi la valeur courante de sa sortie et de son erreur.

On a choisi de modéliser les entrées par un type de neurones particulier dans la classe `InputNeuron`. Ces neurones n'ont pas de poids et on s'en sert simplement pour mettre les données dans le réseau de neurones (en fait, le choix que `InputNeuron` hérite de `Neuron` n'est peut être pas très judicieux).

La classe abstraite `ANN` (pour Artificial Neural Network) contient la description du réseau de neurones. Elle décrit les différentes couches de neurones. En particulier, tout réseau doit avoir la couche d'entrée (`inLayer`) et la couche de sortie (`outLayer`). La classe contient aussi les données pour entrainer le réseau (`trainingData`) et pour tester le réseau (`testingData`). Pour vos réseaux, il faudra implémenter les méthodes

- `Output feed(Input in)` qui donne la donnée `in` en entrée et qui calcule la valeur de sortie du réseau stockée dans une instance de `Output`.
- `Map<Integer,Double> train(int numIterations)` qui va entrainer votre réseau sur les données contenues dans `trainingData`. La sortie de cette méthode est un map qui va associer

le numéro d'une itération et le nombre d'erreurs commises sur l'ensemble de données de test. Pour ce faire, la méthode `double test(Map<Input,Output> data, int it)` est implémentée et calcule ce nombre d'erreurs.

Travail à effectuer

Le projet est à effectuer en binôme. Ce projet compte pour 40% de la note de l'UE. Il est donc souhaitable que la note corresponde au travail de votre binôme, et non aux conseils d'autres groupes, d'autres étudiants ou d'internet. Si j'ai des soupçons de plagiat, je n'hésiterai pas à saisir le conseil de discipline.

Complétez les classes `Neuron`, `SingleLayer` et `OneHiddenLayer`. Vous pouvez créer des sous classes si vous avez besoin. Vous ne pouvez pas modifier les classes `MnistData`, `ANN`, `Input`, `InputNeuron`, `Output`, `Activation`, `Linear`, `Sigmoid`, `Tanh`.

Rédiger un rapport (deux pages maximum) qui contient

- La présentation des résultats que vous avez obtenus ainsi que la présentation des paramètres du meilleur réseau que vous avez construit.
- Une discussion sur ce qui a été le plus dur à implémenter pour vous et sur le niveau de difficulté du projet.

Soumission

1. Le projet est à rendre le **18 décembre 2017** sur la plateforme `myCourse`
2. le projet est à faire en binôme. Il y aura une soutenance organisée après la semaine d'examen. Si le nombre d'étudiants est impair, soit un étudiant sera seul ou bien au plus un seul groupe de trois étudiants est autorisé. C'est votre responsabilité de vous organiser.
3. Vous devez soumettre deux fichers :
   - une archive `zip` dont le nom est la concaténation des noms des auteurs et qui contiendra l'ensemble des fichiers sources. Si les auteurs sont Astérix et Obélix, le fichier zip sera donc nommé `asterix_obelix.zip`.
   - un document au format pdf contenant votre rapport. N'oubliez pas d'indiquez le nom des auteurs du travail.