TP de Machine Learning

Exercice 1 : *Playground Tensorflow*

Dans ce premier exercice, nous discutons et illustrons différentes propriétés fondamentales des perceptrons multi-couches à l'aide du « bac à sable » de TensorFlow disponible ici :

http://playground.tensorflow.org

Voici une brève description de l'interface.

— Le bandeau du haut permet les réglages suivants, de gauche à droite :

- — réinitialiser le réseau *reset*, lancer l'apprentissage *run / pause*, exécuter un pas (*step :* une *epoch*) d'apprentissage (Epoch indique l'avancée de l'apprentissage);
- — changer le *learning rate* (il s'agit du pas dans la descente de gradient);
- — la fonction d'activation (vous avez le choix);
- — le type et le taux de régularisation (cf cours S1);
- — le type de problème avec lequel on joue (restez sur Classification).

— le panneau du bas donne les indications suivantes, de gauche à droite :

- — **Data** : choix d'une base de données dans le plan (le nom de la base s'affiche en survolant avec la souris), proportion entre données d'apprentissage et de test, bruit sur les données (*noise*), taille du lot (*batch size*), bouton pour régénérer les données.
- — **Features**, il s'agit du choix des neurones d'entrée : par défaut les deux coordonnées (x_1 ,x_2) de points du plan, mais on peut ajouter des caractéristiques comme le carré de chaque coordonnée, leur produit, ou leur sinus. Chaque carré représente les valeurs des entrées pour les points du domaine couvrant la base de données : en orange une valeur négative, en bleu positive.
- — **Architecture du réseau** : on peut changer le nombre de couches cachées et le nombre de neurones dans chaque couche. Les poids des liaisons sont représentés par différentes épaisseur et code couleur, un survol avec la souris permet d'afficher la valeur courante, et même de la changer. Même chose pour les biais dont on peut accéder à la valeur par le petit carré noir à gauche de chaque neurone caché (attention, il semble que le biais de la sortie ne soit

pas visualisable). Le carré en chaque neurone représente la sortie du neurone pour chaque valeur représentée par les carrés des neurones de la couche précédente : dans la première couche cachée, il s'agit des sorties pour les valeurs d'entrée lorsque celles-ci parcourent le domaine de la base de données.

— **Output** : représentation graphique du *test loss* (calculé sur la base de test) et du *training loss* (calculé sur la base d'apprentissage), représentation de la sortie du réseau en chaque point du domaine, superposé aux données d'apprentissage; échelle de couleurs représentant les différentes valeurs; et possibilité de montrer les données test et de discrétiser la sortie (*discretize output* : il s'agit d'un seuil par rapport à 0, c'est la fonction de décision pour 1 une classification bi-classe). Rappelons que l'apprentissage adapte les poids en minimisant le *learning loss* par descente de gradient.

Vous reviendrez aux paramètres par défaut (rechargez la page) après chaque question.

Remarque préliminaire : on peut avoir envie d'utiliser une valeur plus grande du *learning rate* pour accélérer la convergence de l'algorithme de descente de gradient à pas fixe. Néanmoins, lorsque le graphique du *training loss* se met à osciller en restant globalement au même niveau, il faut diminuer le *learning rate* car on est dans un cas où l'algorithme de descente « saute » de part et d'autre d'un minimum local sans parvenir à l'atteindre.

Faites les expériences suivantes (et n'hésitez pas à expérimenter librement) :

1. Commencez par une architecture à 0 couche cachée, et cochez *discretize output* : il s'agit du perceptron de Rosenblatt. Constatez que le perceptron ne peut apprendre que des séparations linéaires, ce qui est utile pour une seule des bases de données proposées. Pour cette base, quelle est l'équation de la droite séparatrice ?

2. Utilisez à présent une couche cachée composée d'un seul neurone. Lancez l'apprentissage, et visualisez la forme de la séparation. Justifiez qu'elle est toujours linéaire, quelle que soit la fonction d'activation de ce neurone caché.

3. Utilisez à présent une couche cachée composée de deux neurones. Lancez l'apprentissage, et visualisez la forme de la séparation sur les différentes bases de données.

   Combien de neurones sont-ils nécessaires dans la couche cachée pour séparer les deux classes dans les quatre exemples ? Vous pouvez augmenter la valeur du *learning rate*.

4. Toujours avec une couche cachée, utilisez l'activation ReLU, et visualisez les séparations entre classes dans les différents exemples. Quelle différence remarquez-vous par rapport à une activation sigmoïde ou tanh ?

   Que se passe-t-il si on choisit une activation *linear* ?

5. Constatez que si on enrichit les *features* en entrée, on peut se contenter d'un réseau bien plus simple.

6. On revient aux paramètres par défaut (rechargez la page) et au jeu de données *circle*. Augmentez la valeur de *noise* à 50. Les deux classes sont alors assez « entremêlées ». On considère un réseau à quatre couches cachées, chacune étant formée de quatre neurones. Constatez que l'entraînement du réseau mène au surapprentissage (vous pouvez commencer avec un *learning rate* à 0.3). Comment l'éviter ?

7. On considère la base de données formée des deux spirales entremêlées. Plutôt qu'ajouter des *features* non linéaire et utiliser un réseau simple, on revient aux *features* X_1 et X_2, et on considère un réseau de 6 couches cachées à 8 neurones chacune, fonction d'activation ReLU (plus rapide à calculer que tanh ou sigmoïde), *learning rate* de 0.03 (au départ), régularisation L_2, et *regularization rate* de 0.003. Utilisez une taille de lot de 20. Laissez l'apprentissage se dérouler pendant quelques centaines d'*epochs* (en diminuant le *learning rate* en cours d'apprentissage, comme expliqué dans la remarque préliminaire).