Introduction à l'apprentissage automatique
Séance 5
Exercice 1: *Playground Tensorflow* – CORRECTION

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 :
  - réinialiser 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) ;
  - 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 donnée 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* : cf apprentissage par lot dans le cours), 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

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 (cependant, l'apprentissage ne se fait pas avec l'algorithme du perceptron vu en cours mais bien par minimisation du *training loss*). 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 ?

   La sortie est : $H(w_1x^1 + w_2x^2 + b)$ où H vaut -1 si $w_1x^1 + w_2x^2 + b < 0$ et 1 si $w_1x^1 + w_2x^2 + b > 0$. Ici, peu importe la fonction d'activation choisie : elle ne joue aucun rôle car il n'y a pas de couche cachée, elle n'agit pas sur la couche de sortie. On lit les poids w^1 et w^2 en passant la souris au dessus des arcs correspondants, mais on n'a pas accès au biais b.

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.

   On voit que la séparation est toujours linéaire. Ce réseau calcule :

   $$w_s\sigma(w_1x^1 + w_2x^2 + b) + b_s$$

   où w_1 et w_2 sont les poids entre les entrées et le neurone de la couche cachée, b le biais, w_s le poids entre le neurone de la couche cachée et la sortie, b_s le biais correspondant, et σ l'activation du neurone caché.

   Si le résultat est positif, on affecte à une classe, sinon à l'autre. La surface de séparation est telle que $\sigma(w_1x^1 + w_2x^2 + b) = -b_s/w_s$. Comme σ est croissante comme toute activation, la séparation est toujours une droite, d'équation $w_1x^1 + w_2x^2 + C = 0$ (où $C = b - \sigma^{-1}(-b_s/w_s)$ si σ est inversible).

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 donnés.

   On converge vers des séparations qui ne sont généralement pas suffisamment complexes pour séparer les deux classes (sauf pour *Gaussian* : les deux clusters bien séparés, pour lesquels zéro neurone caché suffisait).

   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*.

   Avec trois neurones, on arrive à séparer les deux cercles concentriques (*circles*), avec quatre ou cinq on commence à séparer correctement *Xor*. Avec huit neurones, on ne s'en sort toujours pas avec *spiral* (faire au moins 2000 epoch). Cf théorème d'approximation universelle : théoriquement, pour tout problème il existe une solution à une couche cachée, mais le nombre de neurones de cette couche peut être grand.

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 ?

   Les séparations ont un aspect polygonal avec ReLU. On peut se contenter de ce constat qualitatif.

   Si on souhaite aller plus loin, on peut aussi développer comme suit (facultatif).

   Rappelons que $\text{ReLU}(x) = \max\{x, 0\}$.

   Avec K neurones dans la couche cachée, un point de coordonnées (x^1, x^2) est classé en « bleu » si :

$$\sum_{k=1}^{K} w_s^k \text{ReLU}(w_1^k x^1 + w_2^k x^2 + b_k) + b_s > 0$$

   On considère la partition du plan formée par les zones délimitées par les k droites d'équation $w_1^k x^1 + w_2^k x^2 + b_k = 0$. Pour tout k, si $w_1^k x^1 + w_2^k x^2 + b_k < 0$, la contribution du k-ème neurone caché est nulle, et si $w_1^k x^1 + w_2^k x^2 + b_k > 0$, sa contribution est $w_1^k x^1 + w_2^k x^2 + b_k$. À l'intérieur d'une des « zones » précédemment définies, la classe bleue est donc délimitée par une droite. On se rend compte que la classe bleue est l'union de l'intersection de demi-plans, d'où l'aspect polygonal de la frontière de séparation (une classe peut être l'union de plusieurs polygones disjoints).

   Pour une preuve complète et d'autres résultats, consultez :
   X. Pan, V. Srikumar, *Expressiveness of rectifier networks*, Proceedings of ICML 2016.
   `http://proceedings.mlr.press/v48/panb16.pdf`

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

   ici $\sigma(x) = x$, et la sortie du réseau, quelle que soit l'architecture, est une fonction affine de l'entrée. La séparation est donc toujours une droite. C'est bien la non-linéarité de l'activation qui rend possible l'approximation de fonctions d'aussi près que l'on souhaite.

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

   Par exemple pour *Xor*, le perceptron de Rosenblatt (0 couche cachée) peut classifier, car $x^1 x^2 > 0$ (resp. <0) donne le carré correspondant aux points bleus (resp. rouges). On voit que le poids appris pour le feature produit $x^1 x^2$ est beaucoup plus grand que les poids pour x^1 et x^2. Idem pour les *circle* lorsqu'on utilise les features x^1, x^2, $(x^1)^2$, $(x^2)^2$ et $x^1 x^2$, avec aucune couche cachée. C'est logique, la surface de séparation est solution de $w_1 x^1 + w_2 x^2 + w_3 (x^1)^2 + w_4 (x^2)^2 + w_5 x^1 x^2 + b$ (une ellipse). En activant tous les features pour les *spiral*, on converge assez rapidement vers une classification raisonnable avec deux couches cachées à quatre neurones, ou une couche cachée à huit neurones.

   En faisant cela on introduit la non-linéarité en entrée du réseau, on a donc besoin de moins de neurones dans les couches cachées. C'est de l'ingénierie de features classique, mais bien entendu c'est assez heuristique : on ne sait pas a priori quels features introduire (cela dépend du problème). L'intérêt des réseaux de neurones multicouches est justement que les premières couches du réseau sont censées calculer des features adaptés à la base d'observations...

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, et fonctions d'activation ReLU. Constatez que l'entraînement du réseau mène au surapprentissage (vous pouvez commencer avec un *learning rate* à 0.3). Comment l'éviter ?

   Le *test loss* finit par remonter (alors que le *training loss* continue à descendre, ce qui est logique car c'est celui que l'apprentissage minimise), on colle trop aux données d'apprentissage,

ce qui n'est pas une bonne idée lorsqu'elles sont « bruitées ». Il y a un grand écart entre *test loss* et *training loss* (qui est bien plus faible), ce qui montre que le classifieur ne généralise pas correctement face à des nouvelles observations. Solution : on arrête l'apprentissage si le *test loss* remonte (*early stopping*) ou bien on régularise (activez L^2 et jouez avec le taux de régularisation, vers 0.001).

7. **À faire en fin de séance après les autres exercices si le temps disponible le permet (le temps de calcul est assez important, et la question est purement illustrative).** 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).