Task 2

Lucas Magnana

June 2020

Ce rapport décrit le projet trouvable à l'adresse suivante : https://github.com/LucasMagnana/Task_2
Les packages à installer sont dans le fichier "requirements.txt" et le fichier "main.py" est à exécuter avec python 3.7. Pour tester le réseau contenu dans le fichier "network.pt", il faut rajouter l'argument "test".
Exemple : "python main.py test".

1 Les données

Pour la "Task 2", j'ai du implémenter une solution de machine learning au problème posé lors de la "Task 1". J'avais le choix entre implémenter ma solution au problème, où la solution donnée dans l'énoncé. Les deux proposaient d'entraîner un réseau de neurones, mais ma solution utilisait des données "path-based" (c'est à dire des données décrivant les chemins entre deux noeuds) tandis que la solution donnée dan l'énoncé utilisait des données "link-based" (c'est à dire des informations sur tous les liens du graphe). J'ai pris la décision d'implémenter la solution "link-based", car je voulais voir les performances d'un réseau de neurone sur ce type de données (que je pensais trop complexe). J'ai cependant supprimé des données la durée totale de transaction.

2 Le réseau de neurones

J'ai utilisé le framework pyTorch pour développer mon réseau de neurones. Ce réseau est un fully connected, composé d'une couche d'entrée, d'une couche cachée et d'une couche de sortie. La fonction d'activation RElu est utilisée entre chaque couche, et la sortie passe par une fonction LogSoftMax. La sortie est donc un vecteur de quatre probabilités, et il suffit de prendre la plus haute pour savoir dans quelle classe le réseau à classée l'entrée. J'aurais pu utiliser un encodage one hot (c'est d'ailleurs comme cela qu'étaient classées les données) mais j'ai préféré utiliser LogSoftMax que je trouve plus adapté lorsque une entrée ne peut pas appartenir à plusieurs classes en même temps. J'ai utilisé la fonction d'erreur CrossEntropyLoss qui m'a donnée de meilleur résultats que la fonction NLLLoss (negative log likelihood) sûrement car le problème n'a que

quatre classes différentes. Enfin, j'ai utilisé l'algorithme Adam pour la descente du gradient, qui s'est avéré meilleur que SGD.

3 Les résultats

La cross-validation m'a permis d'évaluer la performance de mon réseau : 20% des données sont prise aléatoirement et enlevées des données d'entraînement. Une fois le réseau entraîné, on lui envoie en entrée ces 20%, et on vérifie qu'il arrive bien à les classées. Les résultats sont assez aléatoire, allant de 65% à plus de 85% de prédictions correctes. J'ai enregistré un réseau ayant plus de 88% de prédictions correctes, que ce soit sur les données d'entraînement ou les 20% de données utilisées pour le test. Ce réseau est testable sur l'ensemble des données en rajoutant l'argument "test" lors de l'exécution du programme.