Département de génie informatique et de génie logiciel

INF4215

Introduction à l'intelligence artificielle

Laboratoire #3

Travail personnel

Éric Morissette, #1631103

Sacha Licatese-Roussel, #1635849

13 avril 2016

École Polytechnique de Montréal

1 - Introduction

Lors de ce troisième et dernier travail pratique, nous avions le choix du sujet afin de réaliser une application impliquant l'intelligence artificielle. Nous avons donc opté pour un travail concernant l'apprentissage machine par réseau neuronal. Et donc, nous nous sommes dirigé vers un problème classique de ce domaine de recherche; La reconnaissance de nombres manuscrits. De plus, afin de rendre le travail un peu plus intéressant, nous avons ajouté une partie interactive, permettant de tester l'apprentissage réalisé à partir d'une entrée de données de la part de l'utilisateur.

2 - Présentation du travail

Dans cette seconde section, nous présenterons les différents aspects du travail réalisé. Pour ce faire, nous aborderons le projet en effectuant une description de la conception des deux principales parties du travail, c'est-à-dire la partie d'apprentissage et d'interaction avec l'utilisateur à l'aide d'une interface usager. Nous enchainerons ensuite avec les difficultés rencontrées, les limites du projet ainsi que les améliorations possibles du travail.

2.1 - Description de l'approche utilisée

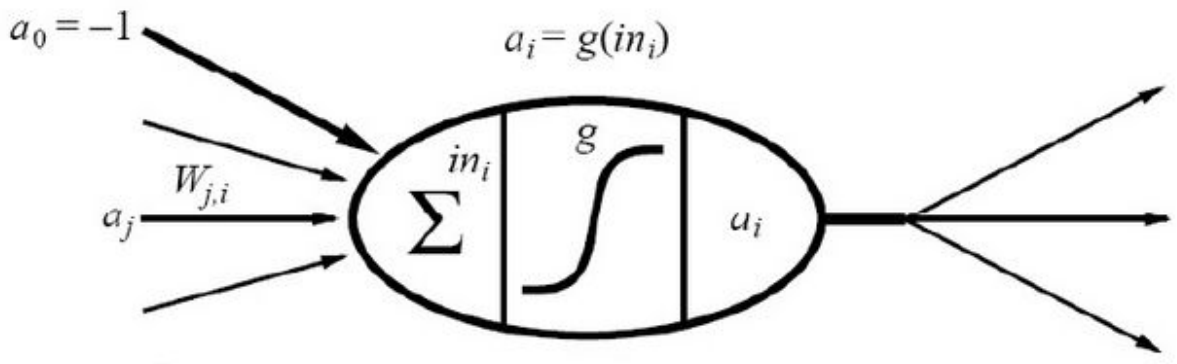
Commençons donc avec la description des deux différentes parties du travail réalisé en commençant par le coeur du sujet; L'apprentissage machine.

2.1.1 - Description de la partie IA

Pour la section sur l'apprentissage machine, elle est effectuée avec un réseau neuronal classique. C'est-à-dire qu'il y a un niveau d'entrée, un nombre de niveaux cachés, déterminés par le programmeur et un niveau de sortie. Le niveau d'entrée doit être assez grand pour prendre toute l'information d'une vérification. Dans notre cas, nous

avons des images de 28 pixels par 28 pixels, nous avons donc 784 entrées. Pour ce qui est du niveau de sortie, il doit corresspondre au nombre de réponses possibles, les chiffres de 0 à 9 dans notre cas, il y a donc 10 neurones de sortie. Dans notre travail, afin de simplifier l'apprentissage, il n'y a qu'un seul niveau caché, de taille arbitrairement choisie, de 500 neurones.

Maintenant que nous avons expliqué la structure de notre réseau, nous pouvons passer au coeur de la matière, comment est-ce que ce réseau fonctionne. Commencons par une image représentant un seul neurone:



Un neurone comporte un ensemble de connexions en entrée, ici nommées a_j avec j = 1...n ou n est le nombre de neurones dans le niveau précédent et une seule entrée a_0 représentant un biais. Ces connexions ont un poids $W_{j,i}$ associées, qui correspond au poids entre la connexion du neurone j au neurone visé i. Dans notre réseau, les neurones d'un niveau sont connectés à tous les neurones du niveau précédent. La variable in_i représente simplement la somme des entrées $in_i = \sum_{k=0}^{n} a_k * W_{k,i}$, ici, nous utilisons la variable k pour la sommation car on prend a_0 ainsi que tous les a_j.

La fonction d'activation $g(in_i)$, qui est un softmax dans notre cas ($\max(0, in_i)$), correspond au coeur de notre neurone. La valeur de retour de cette fonction sera la valeur a_i, qui est envoyée aux neurones du niveau suivant. Si le neurone est dans la dernière couche, alors la valeur a_i correspond à la probabilité, sur 1, que la réponse soit associée à ce noeud.

Pour ce qui est d'une explication globale du réseau, comme dit précédamment, tous les neurones d'un noeud sont connectés avec tous les neurones du niveau précédent. Lors de l'exécution, nous prenons les données du MNIST et les séparons en 3 groupes, les valeurs d'apprentissage, de validation et de test. Nous montrons les images d'apprentissage au réseau et regardons les valeurs de sorties. Par la suite, nous modifions les paramètres $W_{j,i}$ de tous les niveaux ainsi que les poids de biais, afin de modifier le comportement du réseau. Cet apprentissage est dit "apprentissage surveillé" car à chaque étape d'apprentissage, nous connaissons la réponse exacte au problème. Lorsque l'apprentissage est fini, nous validons le réseau en lui passant les valeurs de validation. Si la validation est bonne, c'est-à-dire que le pourcentage d'erreur est moindre que celui de la dernière validation, alors nous prenons les nouvelles valeurs des poids W afin de continuer à entrainer le réseau. Les valeurs de test sont utilisés indépendamment des autres et servent à démontrer s'il y a un sur-apprentissage ou non lors de l'exécution du réseau.

Aussi, nous utilisons deux facteurs permettant d'empêcher le sur-apprentissage de l'ensemble de validation lors de l'apprentissage. Premièrement, tel que mentionné lors de la description d'un seul neurone, le facteur de biais a0 est toujours à une valeur constante, pour tous les noeuds. C'est-à-dire qu'il y a un facteur plus ou moins aléatoire d'ajouté à chaque noeud. Donc, ce facteur permet d'empêcher un apprentissage au pixel près des valeurs d'apprentissage et de validation et permet d'avoir un réseau plus robuste. Deuxièmenent, nous utilisons une régularisation L1 L2, ce qui veut dire que lorsqu'on ajuste les valeurs des poids W*j,i*, on les ajuste selon l'ajustement "parfait" provenant du calcul ainsi que deux facteurs, L1 et L2, qui varient en fonction de la somme de tous les $W_{j,i}$ du niveau visé. Cette régularisation est effectuée lors d'une étape de propagation arrière, étape qui ajuste les poids $W_{j,i}$ afin d'avoir de meilleurs résultats et qui est le concept de base d'un réseau neuronal.

2.1.2 - Description de la partie interactive

Maintenant que notre projet est en mesure d'apprendre à reconnaitre les nombres manuscrits sous certaines conditions et à l'aide de certains paramètres, nous trouvions intéressant de pouvoir expérimenter avec cet apprentissage et d'intéragir avec le travail réalisé jusqu'à présent. Pour ce faire, nous avons mis au point une interface usager simple composée d'un canvas (Avec le module Python Tkinter) et de deux boutons. Le canvas permet donc de dessiner un chiffre à main levé, ensuite, le premier bouton permet de soumettre le dessin à notre module, qui lui retournera le chiffre le plus probable détecté (selon l'entrainement réalisé) par notre module d'apprentissage, et le dernier bouton permet de réinitialiser le canvas.

Au final, ce petit ajout à ce problème typique d'apprentissage machine nous semble plutôt intéressant et permet de mettre en contexte l'apprentissage réellement effectué tout en permettant de valider les différents paramètres utilisés lors de l'apprentissage à partir du Dataset du MINST.

2.2 - Difficultés rencontrées

Pour ce qui est de la construction du réseau de neurones, nous n'avons pas eu beaucoup de problèmes. Étant donné la relative simplicité du problème ainsi que son utilisation comme étant un des problèmes les plus classiques d'apprentissage machine, il y avait beaucoup d'information et de structures de code disponibles sur internet afin de nous aider à développer notre propre projet. Par contre, nous avons eu à comprendre plusieurs concepts clés lors de la programmation en Python d'un réseau neuronal. Premièrement, nous avons utilisé la librairie *Theano*, qui permet de construire des fonctions optimisées, en C, directement dans l'environnement Python, qui est lui-aussi développé en C si l'on creuse assez profond. *Theano* utilise certaines fonctionnalités et syntaxes complexes qui ont pris un certain temps à comprendre.

Deuxièmement, il a fallu faire un gros travail de restructuration du code suite à l'ajout de la fonctionnalité graphique, car il fallait pouvoir accéder à certaines variables que nous n'avions pas initialement besoin lors de l'exécution initiale du réseau de neurones.
Pour ce qui est de l'interface usager et l'interaction entre ce qui est dessiné par l'utilisateur ainsi que le réseau neuronal, nous avons eu un problème avec l'utilisation de la librairie Tkinter. Le canvas qui est créé par la librairie permet de dessiner à l'intérieur mais il n'y a pas de fonctionnalités pour aller prendre ce canvas et le transformer en tableau de pixels. Il a donc fallu aller insérer du code lors des appels de mouvements de souris lorsque le bouton de la souris est appuyé. Avec cette fonction, qui contient les valeurs de positions de la souris présentement ainsi que lors du dernier appel de cette même fonction, nous avons pu remplir nous-même un tableau qui contient environ les mêmes valeurs que le canvas affiché à l'écran. Cette tactique aura fonctionnée uniquement avec des valeurs de blanc et de noir, autrement, il aurait été necéssaire de trouver une autre solution.

2.3 - Aptitudes et limites du projet

Passons maintenant aux aptitudes du projet ainsi que ses limitations. Premièrement, un réseau de neurones comporte beaucoup de paramètres "internes", c'est-à-dire les poids $W_{j,i}$, mais il comporte aussi plusieurs paramètres "externes" et indépendants du processus. On peut donc modifier ces valeurs afin d'obtenir une exécution plus rapide mais moins précise, ou plus lente mais plus robuste face au sur-apprentissage, etc. Ces paramètres peuvent donc compter comme une aptitude de notre travail.
Aussi, notre réseau, comme la plupart des problèmes d'apprentissage machine, arrive rapidement à un état relativement stable, c'est à dire ~5% d'erreur dans notre cas et ensuite ralenti rapidement. Ce phénomène est un effet qui arrive tout le temps lors de travail avec de l'apprentissage machine. Il n'y a aucun moyen d'avoir une courbe d'apprentissage linéaire qui arrive rapidement à 100% de précision, valeur qui est, jusqu'à présent, impossible à atteindre même avec les meilleurs constitutions de réseaux modernes.

Pour ce qui est de notre travail en soit, l'utilisation d'un canvas n'est pas parfaite et il faudrait donner beaucoup de temps au réseau afin de bien apprendre avant de lui demander ce qu'il voit dans les exemples de l'utilisateur. Tout de même, vous pourrez observer vous-même qu'avec un apprentissage relativement court, on remarque la capacité du réseau de neurone de deviner le chiffre qu'on trace à l'écran.

2.4 - Améliorations possibles

Tel que mentionné plus haut, le concept de réseau de neurone possède une grande quantité de paramètres et d'approches différentes. On pourrait par exemple, étendre le nombre de couches du réseau afin d'obtenir un réseau plus robuste et plus efficace. Qui plus est, il nous serait même possible de rendre ce nombre de couches paramétrable et déterminer l'équilibre idéal entre performance et robustesse d'apprentissage pour ce problème donné. Comme vous le savez, ce n'est pas les techniques et optimisations qui manquent en apprentissage machine et réseaux de neurones, avec plus de temps on pourrait ajouter un bon nombre d'éléments, tel que le calcul sur GPU et diverses techniques permettant d'éviter le sur-apprentissage plus efficacement, permettant d'améliorer le projet. Cependant, dans le cadre de ce cours d'introduction à l'intelligence artificielle et de ce troisième travail pratique, nous trouvions ce travail de conception et analyse d'ampleur suffisante.

3 - Conclusion

En conclusion, nous avons réalisé dans ce troisième travail, un réseau neuronal classique afin de s'attaquer au problème classique du MNIST. Ceci nous a donc permis d'en apprendre d'avantage, et par nous-même, sur l'apprentissage machine et les réseaux de neurones en général. De plus, puisque nous trouvions intéressant de pouvoir tester de manière interactive les résultats de cet apprentissage, nous avons créé une interface usager permettant de mettre à profit les apprentissages de notre réseau et d'en visualiser les résultats.