Rapport

November 5, 2022

1 Classifying Handwritten Digits (IFT3395 2022 automne)

1.1 Auteurs du rapport

Equipe MS

20161780 Matei Hulubas Barnia
20245469 Stéphane Combo

1.2 Introduction

Le problème est informellement de prédire la somme des chiffres de deux images de MNIST concaténées. Il s'agit d'un problème de classification avec 19 classes (de 9 à 18 inclus).
Nous avons implémenté un modèle de régression logistique de zero et deux réseaux de neurones, un classique et un avec convolutions.
Le premier modèle a un taux de succès maximum de 9%, le réseau de neurone classique d'environ 70% et le réseau de neurones convolutif de plus de 95%.

1.3 Feature Design

Nous avons utilisé la commutativité de l'addition pour générer de nouvelles données.
Concrètement, l'addition des chiffres de l'image 1 et 2 est la même que celle de l'image 2 et 1. Nous avons donc codé un programme python qui génère un nouveau jeu de données avec deux fois plus de données. Cela nous a permis d'améliorer légèrement la performance du réseau de neurones. Nous avons également modifié la normalisation des données qui était de [-1,1] à [0,1] car cela améliorait les performances de notre réseau de neurones.

1.4 Algorithmes

1.4.1 Régression Logistique

Nous avons implémenté une régression logistique avec régularisation l2. Nous nous sommes inspirés du travail de Sophia Yang (1) pour comprendre et implémenter le calcul du gradient.

1.4.2 Réseau de neurones

Nous avons décidé d'utiliser Keras, une bibliothèque utile pour coder des réseaux de neurones simplement et efficacement. Nous avons choisi le modèle *Sequential* qui est le *building block* de cette bibliothèque (2).

Nous avons d'abord essayé un premier modèle de réseau de neurones contenant uniquement des couches classiques *Dense*, puis un deuxième contenant un mélange de couches convolutives *Conv2D* et classiques.

1.5 Méthodologie

1.5.1 Régression Logistique

Nous avons d'abord essayé plusieurs combinaisons de taux d'apprentissages, régularisation et nombre d'itérations "à la main" sur le modèle pour voir ses réactions.
Nous avons ensuite cherché les hyperparamètres les plus performants en divisant les données en dApp et dValid mais des problèmes évoqués en détail dans l'analyse des résultats (notamment la tendance à prédire une des classes majoritaires) font que les taux d'erreur étaient identiques sur dValid pour les valeurs d'hyperparamètres essayées. Nous avons donc été obligé de chercher nos valeurs d'hyperparamètres à la main jusqu'à trouver un modèle qui n'a pas appris à uniquement prédire une des classes les plus présentes.

1.5.2 Réseau de neurones

Il est important de préciser que pour tester ces modèles avant de faire des soumissions, nous avons divisé le jeu de données *train*: 80% des données pour l'entrainement et 20% des données pour valider et avoir une estimation du taux d'erreur du modèle sur de nouveaux exemples.

Au debut, nous ne savions pas vraiment combien de neurones ou de couches cachées étaient nécessaires. Plusieurs documents en ligne annonçaient que MNIST est une tâche simple qui ne nécessite pas plus de 2 couches cachées (3). Nous avons donc décidé de faire un premier modèle avec 2 couches cachées de 128 neurones (nombre que nous avons vu beaucoup de personnes utiliser dans leur solution en ligne) et une couche finale de 19 neurones pour classifier les images. Nous avons donc utilisé des couches de type *Dense* puisqu'elles représentent des neurones simples(4). Pour les fonctions d'activation, nous avons utilisé la $ReLU$ pour les couches cachées et la *softmax* pour la dernière couche.

Pour notre seconde itération du réseau de neurones classique, nous avons eu l'idée d'ajouter une couche et d'augmenter le nombre de neurones dans les couches cachées. Nous nous sommes donc retrouvé avec 3 couches de 256, 256 et 512 neurones respectivement.

Par la suite, nous avons beaucoup cherché en ligne pour trouver une solution qui allait mieux fonctionner et nous sommes tombé sur les couches convolutives type *Conv2D* (5). En effet, nous avons trouvé que ces couches sont particulièrement bonnes pour de l'identification d'objets. Ces couches sont utilisées avant d'appliquer les couches classiques *Dense* et sont utiles pour identifier des *features* récurrentes. Nous avons aussi utilisé les méthodes suivantes:

- *flatten* utile pour transformer les données de 2 dimensions, qui sont données en output par les couches *Conv2D*, en 1 dimension pour être utilisées par les couches *Dense* (6);
- *MaxPooling2D* qui *downsample* un input 2D en prenant la valeur maximale d'une fenêtre de taille donnée en paramètre avec *pool_size* (7);
- *Dropout* qui séléctionne aléatoirement des neurones avec la probabilité donnée en paramètre et les désactive pour ne plus les compter dans le modèle. Ceci aide a prévenir le sur-apprentissage (8).

Après plusieurs itérations, nous avons donc construit un modèle qui contient 4 couches de type

Conv2D (de 64, 64, 128 et 128 neurones respectivement), 2 couches de type *Dense* (de 256 neurones chacune) et une couche de sortie type *Dense* de 19 neurones pour la classification. Toutes les couches sauf la dernière utilisent comme méthode d'activation *ReLU*. Nous avons aussi placé:

- *MaxPooling2D* après chaque 2 couches *Conv2D*;
- *Dropout* entre les couches cachées *Conv2D* et *Dense*;
- *Dropout* entre les couches cachées *Denses* et la couche de output;
- *Flatten* entre les couches cachées *Conv2D* et *Dense* pour passer de 2D a 1D.

Pour ce qui est de l'optimiseur, nous avons utilisé pour tous les modèles l'optimiseur *Adam* (une version améliorée de la descente de gradient)(9). Nous avons essayé des valeurs dans l'intervalle [0.0001 0.001] pour le taux d'apprentissage avant de sélectionner 0.0007 pour notre modèle final. Tous les autres paramètres sont ceux par défaut.

Pour ce qui est de l'entrainement, nous avons utilisé une fonction de coût d'entropie croisée (*categorical_crossentropy*), une valeur de *batch_size* de 50 et effectuons 25 *epochs*. Ces valeurs ont été choisies par essai-erreur.

1.6 Résultats

1.6.1 Régression Logistique

Notre meilleur résultat pour la régression logistique sur Kaggle est de 9%.
Nous avons vite remarqué qu'après quelques pas de gradient, le modèle a tendance à tomber sur un optimum local et le taux d'erreur du modèle stagne. La plupart du temps, le modèle est très biaisé vers une des classes les plus présentes dans le jeu de données (typiquement 8,9 ou 10), on observe alors que tous les biais sont négatifs sauf celui de la classe prédite systématiquement.
Nous pensons que le modèle n'a pas assez de capacité pour pouvoir différencier autant de classes de manière éfficace.

1.6.2 Réseau de neurones

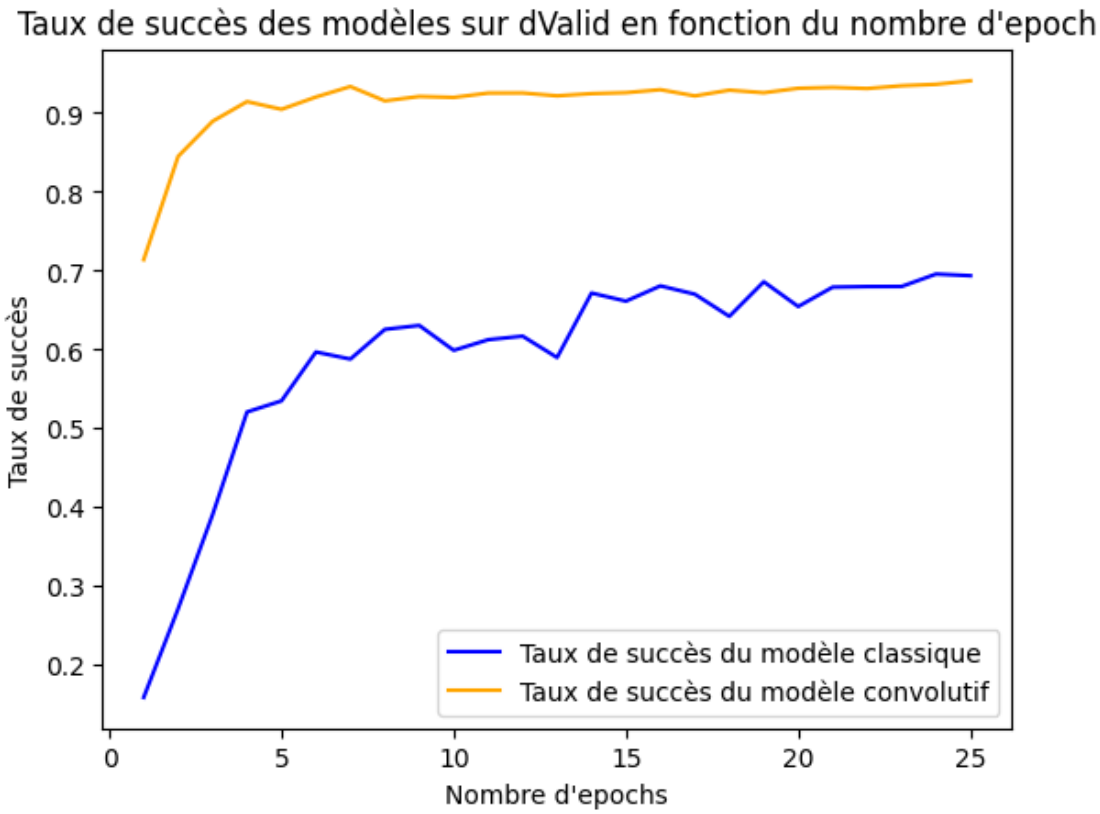
Pour la première version du modèle classique de réseau de neurones, on savait le fait que notre modèle essaie de résoudre un problème encore plus complexe que la classification MNIST, nous nous attendions donc à un mauvais resultat (sous-apprentissage). Nous avions raison puisque notre première itération nous a donné un score d'environ 60%.

Pour la seconde version du modèle classique, le résultat s'était légèrement amelioré et nous avons atteint un score final d'environ 70%.

Pour notre version finale du réseau de neurones avec des couches convolutives, nous avons obtenu un score dans les 95%.

Voici un graphe illustrant les taux d'erreurs du meilleur modèle classique et du meilleur modèle convolution en fonction des *epoch*.

[ ]:



1.7 Discussion

Nous avons adopté une méthodologie itérative pour essayer nos modèles et trouver des paramètres optimaux. Notre processus consistait à essayer de nouvelles valeurs ou modèles, observer le résultat puis juger des modifications les plus pertinantes à effecuer jusqu'à ce qu'on arrive à un résultat qu'on juge satisfaisant. Notre intervention dans le processus a peut-être introduit un biais dans notre recherche de paramètres et modèles optimaux. Une autre méthode aurait pu être de construire directement plusieurs modèles que nous souhaitons essayer avec divers valeurs de paramètres et de sélectionner automatiquement le modèle et les paramètres qui produisent le meilleur résultat. Le majeur défaut de cette méthode est le temps de calcul pour essayer tous ces paramètres. Le fait d'aborder un processus itératif nous a permis d'avancer rapidement dans notre choix de paramètres et de pivoter vers de nouveaux modèles quand on remarquait que la performance de l'architecture actuelle du modèle stagnait.

1.8 Division des contributions

Conception et implémentation du modèle de Régression Logistique : Stéphane Combo
Idée de la génération de données supplémentaires : Matei Hulubas Barnia
Implémentation du script de génération de données supplémentaires : Stéphane Combo
Chargement et pre-traitement des données pour la RL et le réseau de neurones : Stéphane Combo et Matei Hulubas Barnia
Conception et implémentation du réseau de neurones : Matei Hulubas Barnia

"Nous declarons par la presente que tous les travaux presentés dans ce rapport sont ceux des auteurs"

1.9 Références

(1) Sophia Yang, Multiclass logistic regression from scratch
https://towardsdatascience.com/multiclass-logistic-regression-from-scratch-9cc0007da372
https://www.youtube.com/watch?v=wY3PJGZEyY4

(2) The Sequential model, Keras
https://keras.io/guides/sequential_model

(3) Why not use more than 3 hidden layers for MNIST classification?
https://datascience.stackexchange.com/questions/22173/why-not-use-more-than-3-hidden-layers-for-mnist-classification

(4) Dense Layer, Keras
https://keras.io/api/layers/core_layers/dense

(5) When to use Dense, Conv1/2D, Dropout, Flatten, and all the other layers?
https://datascience.stackexchange.com/questions/44124/when-to-use-dense-conv1-2d-dropout-flatten-and-all-the-other-layers

(6) Flatten Layer, Keras
https://keras.io/api/layers/reshaping_layers/flatten

(7) MaxPooling2D Layer, Keras
https://keras.io/api/layers/pooling_layers/max_pooling2d

(8) Dropout Layer, Keras
https://keras.io/api/layers/regularization_layers/dropout

(9) Adam, Keras
https://keras.io/api/optimizers/adam