ADRIEN LAVILLONNIERE – HIEN MINH NGUYEN

RAPPORT DU PROJET DE BIG DATA, PARTIE 2

Polytech Paris-Sud, ET5 Informatique, 2018

TABLE DES MATIÈRES

1 INTRODUCTION

L'objectif de ce projet est de créer un programme capable de classer automatiquement des images de nombres, issues du monde réel.

Nous utiliserons la bibliothèque SVHN, qui contient environ 600 000 numéros de maison extraits de Google Street View. Nous n'en utiliserons que 100 000 environ afin d'entraîner notre modèle et le tester. L'intérêt de SVHN comparé à d'autre datasets repose principalement sur la quantité très élevée de données issues du monde réel. Ces données possèdent toutes un label correspondant à chaque chiffre présent sur une image.

Dans un premier temps, nous utiliserons plusieurs algorithmes de classification tels que le classifieur à distance minimum (DMIN), les classifieurs de Scikit-Learn (SVM). Ces méthodes seront assistées d'un traitement préalable des données (preprocessing), ainsi qu'une réduction de la dimension des vecteurs (PCA). Dans un deuxième temps, nous utiliserons des réseaus de neurones pour effectuer cette classification.

Ce rapport traitera exclusivement de la deuxième partie du projet. Après avoir compris le fonctionnement d'un programme d'apprentissage profond et d'un réseau de neurones, nous avons implémenté plusieurs de ces réseaux et comparé leurs résultats, en fonction des paramètres donnés en entrée. Nous avons pu voir à quel point les réseaux de neurones étaient plus efficace que les classifieurs à distance minimale classiques.

Ce projet nous a permis d'expérimenter avec les bases du machine learning afin de gérer un gros volume de données. En effet, les classificateurs d'images sont une application répandue et très utile au monde des big data de nos jours. La reconnaissance de nombres est une étape basique mais nécessaire dans de nombreux domaines, comme la surveillance, la robotique ou encore la réalité augmentée.

Le code source du projet ainsi que certains résultats sont disponibles ici : `https://github.com/minh-n/NumberReco`.

2 RÉPONSES AUX QUESTIONS

2.1 QUESTION 1

A quoi correspond une epoch ?

Lorsque l'on a passé tout le dataset dans l'algorithme, une epoch s'est écoulée. A la ligne 52, e est donc un entier qui indique le numéro de l'epoch à laquelle on se situe actuellement.

Qu'est-ce qu'un mini-batch et pourquoi l'utilise-t-on ?

Un batch est un échantillon du dataset. Un mini-batch est donc un petit échantillon de ce batch, permettant un gain en efficacité et en terme de mémoire sur une grande quantité de donnes.

A quoi correspondent les méthodes init et forward de la classe CNN ?

La méthode **init** est le constructeur de la classe CNN : elle initialise le réseau et défini chacune des couches.

La méthode **forward** définit la structure du réseau. Elle prend en entrée des données et applique des transformations sur ces données. Elle est appelée à chaque itération de l'algorithme.

À quoi servent nn.Conv2d et nn.Linear ?

nn.Conv2d permet d'appliquer une convolution 2D sur des données en entrée. **nn.Linear** permet d'appliquer une transformation linéaire de la forme y = xAT + b sur les données en entrée.

À quoi servent F.relu et F.max_pool2d ?

F.relu est une couche du réseau faisant office de fonction d'activation, c'est-à-dire qu'elle transforme une valeur en entrée en une autre, selon une fonction. Cela permet de

F.max_pool2d est une couche de "pooling". Elle permet de réduire la dimension des données en entrée et permet d'éviter entre autre le surapprentissage.

À quoi sert l'optimizer optim.SGD ?

L'optimizer optim.SGD (Stochastic Gradient Descent) est un algorithme du gradient permettant de minimiser une fonction objectif. Il sert donc à optimiser le réseau, en mettant à jour les poids du réseau à chaque itération.

A quoi correspondent les prédictions faites par le réseau ?

Une prédiction faite par le réseau pour une image correspond à la classe associée à cette image avec le poids le plus élevé..

Qu'est ce que la loss ?

La loss est la valeur qui indique le degré de perfection des prédictions du modèle. Si le modèle prédit de manière 100% correcte, la loss est de 0. Il faudra donc minimiser la loss afin d'obtenir un modèle compétent. Afin de calculer la loss, il faut faire la différence entre les prédictions et les valeurs réelles des données de test.

Que fait loss.backward() ?

La fonction backward réalise la backpropagation et modifie les poids du réseau en fonction de la loss. Elle permet donc d'améliorer le réseau et optimiser les prédictions de celui-ci.

Question 2) Le code a été modifié pour afficher l'évolution de la précision des données d'entrainement (train_data) et de validation (test_data). La précision converge vers 7% lorsque nous lançons le programme.

Question 3) Le problème était dû à la valeur de l'hyperparamètre learning_rate. Celui-ci était égal à 10 donc supérieur à 1, alors que la fonction du taux d'apprentissage devrait diminuer. Fixer un learning_rate inférieur à 1 (0.001 par exemple) suffit pour résoudre le problème.

Question 4) La précision du modèle est assez mauvaise et plafonne autour de 20-30% : on peut alors conclure qu'il n'y a pas assez de données d'apprentissage. Le nombre d'images utlisées pour l'entraînement est égal à 100, bien inférieur aux 1000 images de test. Il se produit donc un phénomène d'over-fitting (sur-apprentissage) : le réseau ne connait que les caractéristiques des 100 images d'entrainement et s'entraine trop de fois dessus, il est donc limité pour reconnaître autre chose que ces 100 images.

Question 5) Mise en place d'un réseau de type LeNet [1]. Un réseau de type LeNet semble être plus rapide qu'un de type CNN de base, en terme de temps de traitement par epoch. Nous pouvons donc exploiter ce gain de temps en augmentant le nombre d'epoch ou de données d'entraînement.

Question 6) Mise en place d'un réseau de type MLP.

Le réseau de type MLP n'utilise que des couches fully connected, il n'y a donc pas de conv2D. Comme le CNN, on choisir d'avoir 4 couches car plus de couche rendrait les temps de calcul plus longs.

Des articles de recherche[2] montrent également les faibles performances d'un réseau MLP dans notre cas de reconnaissance d'images.

1 https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
2 https://hal-upec-upem.archives-ouvertes.fr/hal-01525504/document

Question 7)

Nous avons testé les 3 réseaux pour les paramètres suivants :

Epoch : 10 Train size : 1000

Test size : 100

Batch size : 10

Learning rate : 0.001

CNN :

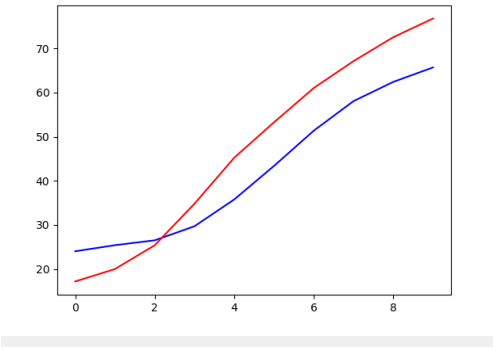

Figure 1: Graphe de l'évolution du succès avec un réseau CNN.

Succès en pourcentage pour les données d'entraînement : 75%

Succès en pourcentage pour les données de test : 72%

MLP :

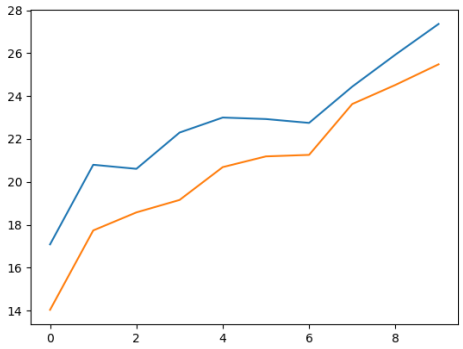

Figure 2: Graphe de l'évolution du succès avec un réseau MLP.

Succès en pourcentage pour les données d'entraînement : 26%

Succès en pourcentage pour les données de test : 28%

LeNet :

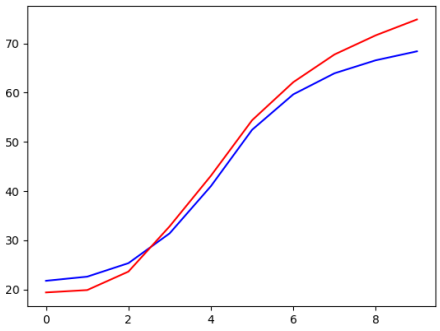

Figure 3: Graphe de l'évolution du succès avec un réseau LeNET.

Succès en pourcentage pour les données d'entraînement : 76%

Succès en pourcentage pour les données de test : 68%

Pour conclure, les CNN sont spécialisés dans le domaine de la reconnaissance d'image : ils ont donc un avantage certain face aux MLP et également par rapport aux classifieurs de la première partie.

Les CNN sont en effet composés de neurones convolutionnels, qui ne sont pas totalement connectés les uns aux autres : la convolution est appliquée sur l'ensemble d'une image pour le CNN, alors qu'un réseau composé de couches fully connected mélange l'image et la rend illisible.

Afin d'améliorer les résultats, nous pourrions appliquer un pré-processing sur l'ensemble des images, comme la première partie.

La valeur du learning rate pourrait également varier dans le temps et diminuer, afin de rendre l'apprentissage plus efficace.