Introduction à l'apprentissage profond grâce à PyTorch

Antoine Perquin

1 Introduction

Le but de cet exercice est de modéliser des fonctions mathématiques à l'aide de réseaux de neurones implenté grâce au framework Pytorch. Par exemple, connaissant x et y, il s'agira d'apprendre une approximation de fonctions linéaires telles que $f : x, y \rightarrow x + y$ et $f : x, y \rightarrow x \times y$, mais aussi d'approximer des fonctions non-linéaires telles que $f : x, y \rightarrow x^2$ et $f : x, y \rightarrow log(x)$.

Le code nécessaire pour charger les données, mais aussi pour entraîner et évaluer le modèle est déjà écrit. L'étudiant doit simplement rédiger le code construisant les réseaux de neurones. Dans cet exercice, on considérera uniquement les réseaux de neurones dits "feed-forward and fully-connected". Dans ce cas, les modèles peuvent être implémentés simplement grâce au conteneur Sequential[1], de la manière suivante :

```
model = Sequential(
    Linear(2, 3),
    Tanh(),
    Linear(3, 3),
    Tanh(),
    Linear(3, 1),
    Sigmoid()
)
```

Ce court exemple de code implémente un réseau de neurones avec 3 couches. Les deux premières ont chacune une dimension de sortie égale à 3, avec une fonction d'activation *tanh*. La dernière couche a pour dimension de sortie 1, avec une fonction d'activation *sigmoid*.

2 Approximation de fonctions linéaires

L'un des intérêts des réseaux de neurones est leur capacité à modéliser des phénomènes non-linéaires. Avant de s'attaquer à ce cas, tentons de modéliser

[1] https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html

des phénomènes linéaires afin de prendre en main le framework Pytorch sur des problèmes plus triviaux.

Dans un premier temps, lancer le fichier `train.py` tel quel. Cela entraînera le réseau de neurones présenté dans l'introduction à prédire $x + y$ en fonction de x et y. Une fois l'entraînement terminé, un graphique traçant la progression de la fonction de coût au cours de l'apprentissage sur le jeu d'entraînement et le jeu de validation est donné dans le fichier `training_plot.png`. De plus le terminal affiche un exemple d'entrée, de prédiction et de valeur réelle (de la taille d'un batch).

Question 1 Le réseau de neurones proposé en introduction ne donne pas de bonnes performances. Comment peut-on le diagnostiquer ? Pourquoi est-ce impossible que ce réseau offre de bonnes performances ?

Indice : Les jeux d'entraînement, de validation et de tests peuvent-être inspectés dans les fichiers `train_data.csv`, `validation_data.csv` et `evaluation_data.csv`

Question 2 Écrire et entraîner un réseau de neurones avec une seule couche afin de prédire $x + y$ en fonction de x et y.

Question 3 Entraîner le même réseau à prédire $x \times y$ en fonction de x et y. Que pensez-vous des performances du réseau ? Comment peut-on modifier le réseau pour améliorer ses performances ?

Remarque : La variable `output_columns` doit-être changée à `"x*y"` afin que les données chargées correspondent à la tâche.

Question 4 Implémenter ces améliorations et entraîner le réseau à prédire $x \times y$ en fonction de x et y.

Une fois satisfait de votre architecture, vous pouvez vérifier que votre réseau généralise bien à des données totalement nouvelles en utilisant le jeu d'entraînement. Pour cela, passer la variable `evaluate_on_test_set` à `True`.

3 Approximation de fonctions non-linéaires

Question 5 Construire et entraîner un réseau de neurones afin de prédire $log(x)$ en fonction de x et y.

Question 6 Sans le modifier, entraîner le réseau de neurones afin de prédire x^2 en fonction de x et y. Évaluer ses performances.

Question 7 Que pourrait-on faire pour améliorer ces performances ?

Indice : L'apprentissage d'un modèle dépend de plusieurs hyper-paramètres : l'architecture du réseau, les données utilisées, la méthode d'optimisation, la fonction de coût, etc.

La fonction $f : x \to x^2$ propose un large éventail de valeurs. En observant les valeurs dans le jeu d'entraînement et les valeurs prédites, on remarque rapidement que le réseau prédit relativement correctement les grandes valeurs, mais échoue envers les petites. En effet, la fonction de coût utilisée est l'erreur moyenne quadratique qui favorise la minimisation d'erreurs sur de grandes valeurs.

Changer la fonction de coût pour l'erreur moyenne log-quadratique permettrait de pénaliser aussi bien les erreurs sur de grandes valeurs que celles sur des petites valeurs. Cette nouvelle fonction de coût est déjà implémentée dans `train.py`, sous le nom de `MSLELoss`.

Question 8 Entraîner le réseau de neurones à prédire x^2 en fonction de x et y, en minimisant l'erreur log-quadratique. Que pensez-vous du résultat ?