Rapport de Travaux Pratiques 2

Amélioration de Réseaux de Neurones Profonds

Joseph BOUIYODA
Département Génie Informatique, ENSPY
bouiyodajoseph@gmail.com
Lien vers le dépôt GitHub (Branche TP2)

22 janvier 2026

Table des matières

1 Introduction

Ce second rapport de Travaux Pratiques fait suite à la construction d'un premier modèle de Deep Learning. L'objectif est d'aller au-delà de l'entraînement simple pour explorer des techniques avancées visant à améliorer la performance et la robustesse des modèles. Nous aborderons le diagnostic des problèmes de biais et de variance, l'application de techniques de régularisation, la comparaison d'optimiseurs avancés et l'utilisation de la Batch Normalization pour accélérer la convergence. Toutes les expériences pratiques seront suivies et analysées à l'aide de MLflow.

Première partie
Concepts Théoriques

(Cette section synthétise les concepts clés abordés dans le TP2.)

2 Diagnostic de Performance : Biais vs. Variance

Le diagnostic se fait en comparant l'erreur du modèle sur les données d'entraînement et de validation.

Biais élevé (sous-apprentissage) : Le modèle est trop simple. L'erreur est élevée sur l'ensemble d'entraînement et de validation.

Variance élevée (surapprentissage) : Le modèle est trop complexe. L'erreur est faible sur l'ensemble d'entraînement mais significativement plus élevée sur l'ensemble de validation.

3 Régularisation et Normalisation

— **Régularisation L2 :** Ajoute une pénalité à la fonction de coût proportionnelle au carré des poids du modèle. Cela incite le modèle à conserver des poids faibles, ce qui le rend plus simple et moins susceptible de surapprendre.
— **Dropout :** Désactive aléatoirement une fraction des neurones à chaque étape de l'entraînement, forçant le réseau à apprendre des caractéristiques plus robustes et moins interdépendantes.
— **Batch Normalization :** Normalise les activations des couches cachées pour stabiliser et accélérer l'entraînement. Elle a également un léger effet de régularisation.

4 Algorithmes d'Optimisation Avancés

— **Momentum :** Améliore la SGD en ajoutant une "inertie" qui aide à accélérer la convergence et à surmonter les minima locaux.
— **RMSprop :** Adapte le taux d'apprentissage pour chaque poids individuellement, en le réduisant pour les poids dont les gradients sont élevés et instables.
— **Adam :** Combine les avantages de Momentum et de RMSprop. C'est un optimiseur adaptatif très efficace, souvent utilisé par défaut.

Deuxième partie

Exercices Pratiques

5 Exercice 1 : Analyse Biais/Variance

5.1 Objectif

Diagnostiquer le comportement de notre modèle de base du TP1 en utilisant un ensemble de validation explicite pour identifier les problèmes de biais ou de variance.

5.2 Résultats

Le modèle a été entraîné sur 15 époques. La sortie de la dernière époque était : `... accuracy: 0.9961 - loss: 0.0123 - val_accuracy: 0.9822 - val_loss: 0.0696`

5.3 Analyse et Réponse

Question : D'après les résultats, le modèle souffre-t-il de biais ou de variance ? Justifiez.
Réponse : Le modèle souffre clairement de **variance élevée (surapprentissage)**.
Justification :

1. **Analyse des métriques finales :** Le modèle atteint une précision d'entraînement excellente de **99.61%**, indiquant un **biais faible**. Cependant, la précision sur l'ensemble de validation n'est que de **98.22%**. Cet écart significatif de 1.39% montre que le modèle performe moins bien sur des données qu'il n'a jamais vues.
2. **Analyse des courbes d'apprentissage :** En observant les courbes (générées par le script), on constate que la perte de validation (`val_loss`) atteint son minimum autour de la 5ème époque avant de remonter, tandis que la perte d'entraînement continue de diminuer. C'est la signature classique du surapprentissage.

6 Exercice 2 : Application de la Régularisation

6.1 Objectif

Corriger le surapprentissage identifié précédemment en ajoutant des techniques de régularisation (L2 et Dropout) au modèle.

6.2 Modification du Modèle

La couche `Dense` a été modifiée pour inclure une régularisation L2 et le taux de Dropout a été augmenté à 0.5 pour un effet plus marqué.

```
from tensorflow.keras import regularizers

model = keras.Sequential([
    keras.layers.Input(shape=(784,)),
    keras.layers.Dense(512, activation='relu',
                       kernel_regularizer=regularizers.l2(0.001)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])
```

Listing 1 – Définition du modèle régularisé

6.3 Résultats

À la fin de l'entraînement, les métriques étaient : `... accuracy: 0.9640 - loss: 0.2175 - val_accuracy: 0.9802 - val_loss: 0.1753`

6.4 Analyse et Réponse

Question : Les techniques de régularisation ont-elles amélioré les performances sur l'ensemble de validation ? Expliquez pourquoi.

Réponse : Oui, les techniques de régularisation ont été très efficaces pour lutter contre le surapprentissage.

Explication : Le résultat le plus frappant est que la précision de validation (**98.02%**) est devenue **supérieure** à la précision d'entraînement (**96.40%**). Ce phénomène est dû au Dropout, qui est actif pendant l'entraînement (bridant le modèle) mais inactif pendant la validation (où le modèle utilise 100% de ses capacités). Cela prouve que le surapprentissage a été complètement éliminé. La régularisation a forcé le modèle à apprendre des caractéristiques plus générales, améliorant ainsi sa robustesse.

7 Exercice 3 : Comparaison des Optimiseurs

7.1 Objectif

Comparer expérimentalement la performance et la vitesse de convergence de trois optimiseurs (SGD avec momentum, RMSprop, Adam) à l'aide de MLflow.

7.2 Résultats

Les performances finales sur l'ensemble de test après 10 époques sont résumées dans le tableau ci-dessous.

Optimiseur	Précision de Test Finale
Adam	98.28%
RMSprop	98.23%
SGD with Momentum	97.38%

Table 1 – Performances finales des différents optimiseurs.

7.3 Analyse et Réponse

L'analyse des courbes de convergence dans MLflow (voir Figure **??**) montre que les optimiseurs adaptatifs **Adam** et **RMSprop** convergent beaucoup plus rapidement que SGD. Ils atteignent une haute précision en seulement 3-4 époques. Adam s'est avéré être le plus performant, justifiant son statut de choix par défaut.

8 Exercice 4 : Batch Normalization

8.1 Objectif

Évaluer l'impact de l'ajout d'une couche de Batch Normalization sur la vitesse de convergence du modèle.

8.2 Résultats

Le modèle avec Batch Normalization a atteint une précision finale de test de **97.86%**.

8.3 Analyse et Réponse

Le principal bénéfice de la Batch Normalization a été une **accélération significative de la convergence**. Le modèle a atteint une précision de validation de **97.62% dès la deuxième époque**, alors que le modèle Adam sans Batch Normalization nécessitait quatre époques pour atteindre un niveau similaire. En stabilisant la distribution des activations, la Batch Normalization permet un entraînement plus rapide et plus fiable.

9 Conclusion Générale

Ce second TP a permis de mettre en pratique un ensemble de techniques fondamentales pour l'amélioration des réseaux de neurones. Nous sommes partis d'un modèle simple présentant du surapprentissage, que nous avons corrigé avec succès grâce à la régularisation L2 et au Dropout. L'expérimentation avec les optimiseurs, suivie via MLflow, a démontré la supériorité des algorithmes adaptatifs comme Adam. Finalement, l'ajout de la Batch Normalization a prouvé son efficacité pour accélérer la convergence. Ces expériences fournissent une base solide pour construire des modèles de Deep Learning plus performants et robustes.