Notes Projet Jupyter Notebook

Ce projet Jupyter Notebook contient le code pour nos expériences et analyses. Voici quelques informations importantes :

1. **Installation des dépendances** : Assurez-vous d'installer toutes les dépendances nécessaires en exécutant la commande suivante :

   ```
   pip install -r requirements.txt
   ```

   Cela garantira que toutes les bibliothèques requises sont correctement installées avant d'exécuter le notebook.

2. **Analyses et explications des expériences** : Les analyses détaillées et les explications de nos expériences sont disponibles dans le fichier `experiments.pdf` (celu-ci). Veuillez vous y référer pour une compréhension approfondie de nos résultats.

3. **Code des expériences** : Le notebook principal (`experiments.ipynb`) contient le code de nos expériences. Nous avons ajouté des commentaires détaillés tout au long du code pour faciliter la compréhension. N'hésitez pas à parcourir le notebook et à lire les commentaires pour comprendre le fonctionnement de chaque étape de l'expérience.

N'hésitez pas à nous contacter si vous avez des questions ou des problèmes avec le projet. Bonne exploration !

Cordialement, Juquel Antonin Hiruthayaraj Raj Porus

Introduction au Deep Learning et aux Réseaux de Neurones

Dans ce projet, nous allons construire des modèles de deep learning pour résoudre une tâche de reconnaissance d'images de chiffres manuscrits. Nous commencerons par un perceptron simple, puis nous ajouterons des couches pour créer un MLP, et enfin, nous expérimenterons avec un CNN. Nous évaluerons et comparerons les performances de ces différents modèles pour comprendre leurs forces et leurs faiblesses.

Dans ce projet nous utiliserons :

- Le jeu de données **MNIST**, qui est largement utilisé pour ce type de tâches.
- **TensorFlow**, qui comprend **Keras** pour la construction de modèles de deep learning.
- **Matplotlib** pour tracer les courbes d'apprentissage pour montrer l'évolution de la précision et de la perte sur les ensembles d'entraînement et de validation au fil des époques.
- **Seaborn** pour afficher la matrice de confusion pour nous montrer le nombre de fois où chaque classe a été prédite correctement ou incorrectement.

Pour commencer nous avons écrit une fonction pour créer des modèles de deep learning avec différentes architectures. Cette fonction prend en entrée une liste de couches et le nombre d'epochs et retourne un modèle Keras compilé avec les couches spécifiées. Nous utiliserons cette fonction pour créer des modèles avec différentes architectures et comparer leurs performances.

0. Lecture du livre et préparation

Le livre *"Neural Networks and Deep Learning"* de Michael Nielsen nous guide à travers l'essentiel de la construction et de l'entraînement des réseaux de neurones. Nous avons suivi ses conseils précieux pour structurer notre approche de modélisation et de formation, que nous avons finalement encapsulée dans la fonction `create_model` .

1. **Perceptrons**: Ce fut notre point de départ, avec le concept de base des neurones artificiels. Les perceptrons constituent le fondement des réseaux de neurones. La structure `Sequential` de notre modèle reflète cette idée de perceptrons interconnectés.

2. **Comment le réseau de neurones apprend avec la rétropropagation**: Nielsen présente ici le réseau de neurones et l'algorithme de rétropropagation, une technique essentielle pour l'entraînement de ces réseaux. En se basant sur ces principes, nous avons compilé notre modèle avec l'optimiseur `Adam`, qui utilise la rétropropagation pour ajuster les poids du réseau en fonction de l'erreur de sortie.

3. **Améliorer la manière dont les réseaux de neurones apprennent**: En nous inspirant des techniques d'amélioration décrites par Nielsen, nous avons inclus dans notre fonction la possibilité de personnaliser les couches, la fonction d'activation, le taux d'apprentissage et d'autres paramètres. Nous avons également intégré des mécanismes pour la surveillance de l'entraînement, comme le fractionnement de validation.

4. **Réseaux de neurones convolutifs profonds**: Enfin, Nielsen nous initie aux réseaux de neurones convolutifs (CNN), une architecture de réseau spécialement conçue pour le traitement des images. Dans notre fonction, nous avons la possibilité d'inclure des couches convolutives dans la structure de notre modèle, en utilisant la classe `Conv2D` de Keras, pour explorer ces techniques d'apprentissage de caractéristiques plus avancées.

L'ensemble de notre démarche, de la construction du modèle à l'évaluation, en passant par l'entraînement, est inspiré de Nielsen. En outre, nous avons ajouté des graphiques de performance et une matrice de confusion pour visualiser et interpréter les résultats de notre modèle, soulignant l'importance de la visualisation dans la compréhension et l'amélioration de nos modèles de réseau de neurones.

1. Classification des données MNIST avec un réseau multicouches

Perceptron de base

Pour initier ce travail, nous avons commencé par un perceptron simple sans couche cachée. Nous avons employé la fonction `create_model` afin de concevoir un modèle avec une couche d'entrée de 784 neurones (correspondant à une image de 28x28 pixels) et une couche de sortie composée de 10 neurones (un pour chaque classe de chiffre). Nous avons choisi d'utiliser la fonction d'activation `softmax` pour la couche de sortie, puis nous avons calculé la perte avec la fonction d'erreur `categorical_crossentropy` et déterminé la précision grâce à la métrique `accuracy` .

Au cours de cette première série d'expériences, nous avons varié le nombre d'epochs.

Les résultats obtenus lors de ces trois expériences montrent une amélioration de la précision du test en fonction de l'augmentation du nombre d'epochs. Cependant, cette amélioration n'est pas linéaire.

Au cours de ma première expérience, nous avons utilisé 5 epochs et obtenu une précision de test de 92.58%. Lors de ma deuxième expérience, en utilisant 15 epochs, nous avons pu observer une augmentation de la précision jusqu'à 92.90%. Finalement, lors de ma troisième expérience, avec 30 epochs, la précision des tests a légèrement augmenté pour atteindre 92.97%.

Il est néanmoins crucial de noter que l'augmentation du nombre d'epochs ne garantit pas systématiquement une amélioration de la précision du modèle. En effet, un nombre d'epochs trop élevé peut conduire à un surapprentissage, où le modèle s'adapte trop spécifiquement aux données d'entraînement et n'est pas capable de généraliser efficacement à de nouvelles données. Dans ces trois expériences, il semble que le modèle commence à montrer des signes de surapprentissage après environ 15 epochs, puisque la précision de validation commence à se stabiliser, voire à diminuer légèrement.

Les matrices de confusion confirment que le modèle est généralement performant pour classer correctement les chiffres, la plupart des erreurs se produisant entre les chiffres dont les formes sont similaires.

En conclusion, selon les résultats obtenus, il semble que le nombre optimal d'epochs pour ce modèle précis soit d'environ 15. Au-delà de ce nombre, les améliorations en termes de précision sont marginales et le risque de surapprentissage devient plus élevé.

Perceptron à deux couches

nous avons ensuite procédé à la création d'un perceptron à deux couches. Dans ce cadre, nous avons ajouté une couche cachée à mon modèle, permettant ainsi au réseau de neurones de comprendre des représentations plus complexes des données.

Pour y parvenir, nous avons inséré une couche `Dense` dotée de `n` neurones et d'une activation `ReLU` avant la couche de sortie. La fonction d'activation ReLU (Rectified Linear Unit) est fréquemment utilisée dans les réseaux de neurones profonds puisqu'elle facilite l'apprentissage de représentations non linéaires.

Dans le but d'étudier l'effet du nombre de neurones dans la couche cachée sur les performances du modèle, nous avons réitéré cette opération avec des nombres différents de neurones, puis nous avons comparé les résultats.

Comme nous l'avons observé, l'ajout d'une couche cachée à mon perceptron a conduit à une nette amélioration des performances par rapport à une simple configuration de perceptron à une couche. Cela démontre l'utilité des couches cachées pour apprendre des représentations plus complexes des données.

En confrontant les performances des trois modèles, nous avons noté que le nombre de neurones dans la couche cachée avait un effet significatif. L'augmentation du nombre de neurones dans la couche cachée, passant de 64 à 128, a entraîné une amélioration de la précision du test. Cependant, augmenter encore ce nombre pour atteindre 256 neurones n'a pas considérablement amélioré la précision du test. Cela pourrait indiquer que le modèle, avec 128 neurones, est déjà suffisamment complexe pour représenter les données d'entraînement.

Il est important de remarquer que, même si l'augmentation du nombre de neurones peut contribuer à capturer des représentations plus complexes, elle peut également entraîner un surapprentissage si le modèle devient trop complexe. Un modèle surappris peut afficher des performances exceptionnelles sur les données d'entraînement, mais il peut ne pas réussir à généraliser correctement à de nouvelles données. Par conséquent, il est crucial d'équilibrer la complexité du modèle avec la quantité et la diversité des données disponibles pour l'entraînement.

De plus, il est également important de mentionner que la matrice de confusion révèle que les modèles ont tendance à commettre certaines erreurs plus fréquemment que d'autres. Par exemple, certains chiffres peuvent être plus souvent confondus avec d'autres. Cela pourrait indiquer des limites dans la capacité du modèle à distinguer certaines classes, ou cela pourrait refléter des caractéristiques spécifiques du jeu de données (par exemple, si certains chiffres sont écrits de manière très similaire).

Perceptron multicouche

nous avons ensuite élaboré un réseau de neurones profond en incorporant davantage de couches cachées à mon modèle. Comme précédemment, nous avons expérimenté avec différents nombres de couches et de neurones par couche, et nous avons observé l'impact sur les performances du modèle.

- Avec 2 couches cachées, le modèle atteint une précision de test de 0.9778 après 15 époques. Il semble qu'un certain surapprentissage se manifeste, la précision sur les données d'entraînement continuant à augmenter, tandis que la perte sur les données de validation commence à augmenter après la 7ème époque. Cela signifie que le modèle commence à mémoriser l'ensemble d'entraînement plutôt qu'à apprendre des représentations généralisables.
- Avec 3 couches cachées, le modèle atteint une précision de test légèrement supérieure de 0.9797 après 15 époques. À l'instar du modèle à 2 couches cachées, ce modèle semble aussi présenter un surapprentissage, la perte de validation commençant à augmenter après la 7ème époque, alors que la précision sur l'ensemble d'entraînement continue à augmenter.
- Avec 4 couches cachées, le modèle atteint une précision de test de 0.9792, qui est légèrement inférieure à celle du modèle à 3 couches cachées, mais supérieure à celle du modèle à 2 couches cachées. Une fois de plus, il semble y avoir un surapprentissage, la perte de validation commençant à augmenter après la 6ème époque, alors que la précision sur l'ensemble d'entraînement continue à augmenter.

Dans l'ensemble, l'ajout de couches cachées a permis d'améliorer légèrement la précision des tests, mais cela a également entraîné un surapprentissage plus marqué. Il serait donc judicieux d'explorer d'autres techniques pour contrôler le surapprentissage, comme la régularisation ou l'abandon (dropout). Les matrices de confusion fournies donnent également un aperçu des types d'erreurs commises par chaque modèle.

2. Classification par réseau multicouches avec convolution

Le réseau de neurones convolutionnel (CNN) que nous avons mis en œuvre est relativement simple, mais efficace pour les tâches de classification d'images. Il se compose des éléments suivants :

1. Une couche de convolution avec 32 filtres de taille 3x3 et une fonction d'activation ReLU. Cette couche extrait des caractéristiques locales de l'image d'entrée.
2. Une couche de max pooling de taille 2x2. Cette couche réduit la dimensionnalité de l'image tout en conservant les caractéristiques les plus importantes.
3. Une couche de Flatten qui transforme la matrice 2D en un vecteur 1D pour être utilisée dans les couches pleinement connectées.
4. Une couche dense (pleinement connectée) avec 64 neurones et une fonction d'activation ReLU. Cette couche apprend des combinaisons non linéaires des caractéristiques extraites par les couches précédentes.
5. Une couche de sortie dense avec 10 neurones (un pour chaque classe de chiffre de 0 à 9) et une fonction d'activation softmax, qui convertit les sorties en probabilités.

Concernant les résultats, il semble que le CNN performe très bien. Le modèle atteint une précision de 98,6 % sur le jeu de test, ce qui est nettement supérieur aux résultats obtenus avec les perceptrons précédents. La matrice de confusion montre que le modèle est capable de classer correctement la majorité des images dans chaque catégorie.

Cela suggère que l'ajout de couches convolutionnelles au début du réseau peut aider à extraire des caractéristiques plus complexes et pertinentes des images, ce qui améliore la performance de la classification.

Le modèle semble également bien généraliser, car la précision sur les données de validation et de test est similaire. Il n'y a pas de signes évidents de surajustement (où le modèle apprend trop bien les données d'entraînement et performe mal sur les données de test), ce qui est un bon signe.

En résumé, ce réseau de neurones convolutionnel offre de bien meilleurs résultats que les perceptrons précédents pour cette tâche de classification d'images.

Dans le deuxième modèle que nous avons partagé, nous avons ajouté deux éléments importants à notre modèle CNN initial :

Initialisation de He : L'initialisation de He est une méthode d'initialisation des poids qui porte le nom de He et al., 2015, qui ont présenté cette méthode d'initialisation dans leur document. Cette méthode est généralement utilisée pour les fonctions d'activation ReLU ou ses variantes.

Régularisation L2 : C'est une forme de régularisation RIDGE qui pénalise les poids au carré dans la fonction de perte. Cela aide à prévenir le surapprentissage en limitant la complexité du modèle.

Dans le troisième modèle, en plus de l'initialisation de He et de la régularisation L2, nous avons ajouté une autre couche de Conv2D et MaxPooling2D, augmenté le nombre de neurones dans la couche Dense à 128, et doublé la taille de la deuxième couche Conv2D à 64. L'ajout d'une autre couche de convolution et de mise en commun aide à capturer des caractéristiques plus complexes de l'image d'entrée.

Comparons les trois modèles :

Le premier modèle atteint une précision de test de 98,60% avec une perte de test de 0,0465. Le deuxième modèle atteint une précision de test légèrement inférieure de 98,08% avec une perte de test légèrement plus élevée de 0,1301. Le troisième modèle atteint une précision de test de 98,83%, ce qui est légèrement supérieur aux deux modèles précédents, avec une perte de test de 0,0907.

En comparant les matrices de confusion, tous les modèles ont des performances similaires avec quelques erreurs dans les mêmes classes. Cependant, le troisième modèle semble être légèrement meilleur que les deux autres en termes de performances globales.

En comparaison avec les perceptrons précédents, ces modèles CNN ont des performances nettement supérieures, car les réseaux neuronaux convolutifs sont spécifiquement conçus pour traiter des tâches liées à l'image et sont capables de capturer les caractéristiques spatiales de l'image, ce que les perceptrons ne peuvent pas faire. C'est pourquoi les CNN sont généralement préférés pour les tâches de classification d'images.

Dans l'ensemble, ces trois modèles ont de très bonnes performances sur l'ensemble de test, mais le troisième modèle a une performance légèrement meilleure.

Conclusion

Lors de la conception de réseaux de neurones, la modification de l'architecture, notamment en augmentant le nombre de neurones dans la couche cachée ou en ajoutant plus de couches cachées, peut potentiellement améliorer les performances du modèle. Ces ajustements permettent au modèle d'apprendre des représentations plus complexes des données. Cependant, ils peuvent également rendre l'entraînement plus complexe et augmenter le risque de surapprentissage. Il est donc crucial de surveiller attentivement l'évolution de la perte et de la précision sur l'ensemble de validation pour détecter d'éventuels signes de surapprentissage.

Cela dit, lors de l'expérimentation avec différentes architectures, on peut constater que les performances du modèle ne s'améliorent pas toujours en ajoutant plus de couches ou de neurones. C'est parce que la capacité du modèle doit être ajustée en fonction de la complexité des données. Un modèle trop simple peut mener à un sous-apprentissage, tandis qu'un modèle trop complexe peut mener à un surapprentissage.

Dans le contexte de la classification d'images, nous avons constaté qu'intégrer des couches convolutionnelles dans nos réseaux de neurones a permis d'améliorer significativement les performances de classification. Ces couches convolutionnelles extraient des caractéristiques plus complexes et pertinentes des images, ce qui rend le modèle plus performant.

Par ailleurs, la régularisation, l'initialisation des poids (comme l'initialisation de He) et la modification des paramètres, tels que le taux d'apprentissage, ont également un impact sur les performances du réseau. Ces techniques peuvent aider à optimiser le modèle et à prévenir le surapprentissage.

Dans l'ensemble, bien que les perceptrons multicouches soient des outils puissants pour des tâches de classification, les réseaux de neurones convolutionnels sont souvent supérieurs pour des tâches de classification d'images en raison de leur capacité à extraire des caractéristiques spatiales pertinentes. Il est donc important de choisir l'architecture et les paramètres du modèle en fonction de la tâche spécifique à réaliser.