TP3 – Calculs vectorisés

Introduction à l'apprentissage profond
M2 informatique, Université Paris Cité, 2024-2025

La semaine dernière, vous avez implémenté un modèle de régression softmax avec NumPy. Dans ce TP, vous verrez comment les calculs vectorisés peuvent rendre ce code plus rapide.

Comme la semaine dernière, nous utiliserons le jeu de données Fashion-MNIST, et nous supposons que vous savez le charger avec NumPy — consultez le fichier `load_fmnist.py` fourni avec ce TP. On vous fournit également une solution modèle pour le TP 2 dans le fichier `softmax_reg.py`.

Point de départ

1. Parcourez le code dans `softmax_reg.py` et exécutez-le. (Changez le nombre d'époques de 10 à 3 si cela est trop lent sur votre machine.) Vous devriez obtenir une *validation accuracy* de 82,42 %.
2. Au lieu de valider uniquement à la fin de l'entraînement, il est plus courant de réaliser la validation à la fin de chaque époque. Implémentez cela et créez un graphique montrant les courbes des pertes de validation et d'entraînement.

Le code n'est pas très efficace, car chaque image dans un batch est traitée séparément. Le module Python `cProfile` peut aider à identifier où un programme passe le plus de temps. Par exemple,

```
python -m cProfile -s tottime myscript.py > profile.log
```

exécutera un profilage pendant l'exécution de `myscript.py` et enregistrera le résultat dans `profile.log`.

3. Analysez le programme `softmax_reg.py` avec le profiler. (Le profilage ralentira l'exécution du programme.) Quelles opérations et fonctions prennent le plus de temps ?

Vectorisation et broadcasting

Nous pouvons réduire le nombre total d'appels aux fonctions `loss` et `gradient` en utilisant des calculs **vectorisés** et le **broadcasting**.

4. Lisez le fichier `tp3_vec_exos.py` et faites les petits exercices dans ce fichier.

Vectorisation de softmax, loss et gradient

Nous voulons maintenant intégrer ces idées dans notre code d'entraînement.

5. Copiez le fichier `softmax_reg.py` dans un nouveau fichier `softmax_reg_vec.py`. Dans ce fichier, renommez la fonction `train_sgd` en `train_sgd_vec`, et remplacez les deux boucles par la boucle unique suivante :

```
for batch_imgs, batch_labels in create_batches(train_imgs_flat, train_labels, batch_size):
    probs = softmax_batch(np.dot(batch_imgs, W) + b)
    loss += cross_entropy_loss_batch(probs, batch_labels)
    grad_W, grad_b = grad_batch(batch_imgs, probs, batch_labels)
    W -= lr * grad_W
    b -= lr * grad_b
```

6. Implémentez les fonctions `softmax_batch`, `cross_entropy_loss_batch` et `grad_batch`. Chaque implémentation peut avoir au maximum quatre lignes et ne doit contenir aucune boucle `for`.

7. Exécutez l'entraînement en utilisant `train_sgd_vec` à la place de `train_sgd`. Vous devriez constater une amélioration significative de la vitesse (10 fois plus rapide lors de nos tests).

8. Comparez la perte et la précision par époque entre les fichiers `softmax_reg_vec.py` et `softmax_reg.py`. Assurez-vous d'avoir le même réglage de `np.random.seed` dans les deux fichiers. Il peut y avoir de très petites variations dans la perte, mais `np.isclose(lossv1, lossv2)` devrait retourner `True` pour toutes les époques.

9. Écrivez également une version vectorisée de la fonction `evaluate`. Utilisez la fonction `np.argmax` pour cela, ainsi que le fait que l'expression `A == B`, sur des tableaux numpy de même longueur, retourne un tableau numpy de valeurs booléennes.