TP2 – Régression softmax

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

L'objectif de ce TP est d'implémenter un **modèle de régression softmax** pour classer de petites images à l'aide de **NumPy**. Ce TP servira également d'introduction au travail avec les données d'images, les opérations sur les tableaux numpy, et l'optimisation basée sur les gradients.

Vous testerez votre modèle sur le dataset Fashion-MNIST, qui contient des images en niveaux de gris de vêtements, chacune de taille 28×28. Le dataset comprend 10 classes, définies dans la variable `TEXT_LABELS` du fichier de départ. Les données sont fournies sur la page Github sous forme de fichiers `.gz` et peuvent être chargées à l'aide des fonctions utilitaires `load_images` et `load_labels` du fichier `tp2_starter.py`.

Chargement et visualisation des données

1. **Charger le dataset.** Téléchargez les fichiers `.gz` depuis Github, lisez le code dans `tp2_starter.py`, et utilisez les fonctions utilitaires pour charger les datasets d'entraînement et de test. Vous devriez obtenir quatre tableaux NumPy :

   - `train_imgs` ($60000 \times 28 \times 28$),
   - `train_labels` (60000),
   - `test_imgs` ($10000 \times 28 \times 28$),
   - `test_labels` (10000).

   Vérifiez les `shape` de ces tableaux pour vérifier qu'ils ont été chargés correctement.

2. **Visualiser quelques exemples.** Écrivez une fonction `show_image(index)` qui affiche la n-ième image du dataset d'entraînement avec son étiquette. Vous pouvez par exemple utiliser `px.imshow()`, après un `import plotly.express as px`.

3. **Aplatir les images.** Chaque image doit être transformée en un vecteur de taille 784 ($= 28 \times 28$). Créez les tableaux NumPy `train_imgs_flat` de forme 60000×784 et `test_imgs_flat` de forme 10000×784.

Softmax, stabilité numérique et perte d'entropie croisée

La fonction **softmax**, qui transforme un vecteur $\mathbf{o}$ de longueur n (réels) en un vecteur de probabilités $\mathbf{p}$ de longueur n, est définie comme suit pour $1 \leq k \leq n$:

$$p_k := \frac{\exp(o_k)}{P(\mathbf{o})}, \text{ où } P(\mathbf{o}) = \sum_{i=1}^{n} \exp(o_i).$$

4. (a) Implémentez la fonction softmax.

(b) Que se passe-t-il avec les valeurs de la fonction softmax lorsque le vecteur d'entrée contient la valeur 1000 ? Pour remédier à ce problème, définissons $\overline{o} := \max \mathbf{o}$ comme la valeur maximale de $\mathbf{o}$, et définissons $\mathbf{o}'$ par $\mathbf{o}'_j := \mathbf{o}_j - \overline{o}$, pour chaque $1 \leq j \leq n$. Montrez que $\text{softmax}(\mathbf{o}') = \text{softmax}(\mathbf{o})$, d'abord par expérimentation avec Numpy, puis par un calcul sur papier. Modifiez l'implémentation de softmax en conséquence.

La **perte d'entropie croisée** d'un vecteur de probabilités $\mathbf{p}$ pour un exemple dont la classe correcte est $c \in \{0, \ldots, 9\}$ est définie comme le logarithme négatif (base e) de la probabilité p_c assignée à la classe correcte :

$$L = -\log p_c.$$

5. Implémentez la fonction de perte d'entropie croisée pour un seul exemple.

Le *gradient* de la perte L pour un exemple peut être calculé comme suit. Si $\mathbf{p} = \text{softmax}(x^{\mathrm{T}}W + b)$ est le vecteur de probabilités, on définit $\mathbf{p}'$ comme le vecteur de la même taille que $\mathbf{p}$, défini par

$$\mathbf{p}'_j := \begin{cases} p_j & \text{si } j \neq c \\ p_j - 1 & \text{si } j = c \end{cases}$$

où c est la classe pour l'exemple x. Alors :

$$\frac{\partial L}{\partial W} = x \otimes \mathbf{p}' \text{ et } \frac{\partial L}{\partial b} = \mathbf{p}'.$$

Ici, $\otimes$ est le *produit extérieur*, calculé avec `np.outer()`.

6. **Calcul des gradients.** Écrivez une fonction `grad` qui prend en entrée `x`, le vecteur de probabilités $\mathbf{p}$, et l'indice de la classe correcte `c`, et calcule les gradients par rapport à `W` et `b`.

Sur-apprentissage d'un batch unique

Utilisez un batch de données d'entraînement comprenant les 64 premiers exemples. Votre objectif est de faire correspondre parfaitement ce modèle à ces 64 images (on parle de "sur-apprentissage d'un batch unique").

7. **Initialiser les poids et biais** :
   - Créez une matrice de poids `W` de taille 784×10 avec des valeurs tirées d'une distribution normale ($\mu = 0$, $\sigma = 0,01$).
   - Créez un vecteur de biais `b` de taille 10, initialisé à zéro.
8. **Entraîner sur un batch unique.** Implémentez une boucle d'entraînement qui effectue les étapes suivantes :
   - Calculez les logits ($x \cdot W + b$) et les probabilités (softmax).
   - Calculez la perte moyenne d'entropie croisée sur le batch.
   - Calculez les gradients de `W` et `b`.
   - Mettez à jour `W` et `b` avec la descente de gradient.

Vérifiez que le modèle s'adapte au batch. Avec un taux d'apprentissage de `0,1` et `500` répétitions, vous devriez obtenir une perte moyenne d'entropie croisée inférieure à `0,05` pour le batch.

9. Visualisez la courbe de perte.
10. **Évaluation.** Écrivez une fonction pour calculer la précision du modèle (= le pourcentage d'exemples où la prédiction du modèle est correcte). Comme prédiction, choisissez la classe avec la probabilité la plus élevée. Quelle est la précision du modèle : (a) sur le batch sur lequel vous vous êtes entraîné ? (b) sur un batch du jeu de validation ? Pour (a), vous devriez obtenir 100 %. Pour (b), la précision devrait être d'au moins 60 %.

Entraînement sur le dataset complet

La *descente de gradient par mini-batchs* effectue les mêmes étapes que précédemment, sauf qu'un "epoch" considère maintenant tous les batchs du dataset d'entraînement. Avec un taille de batch de `64` et `50000` exemples d'entraînement, cela signifie qu'un epoch fait $50000/64 \approx 781$ répétitions de la boucle "batch unique" que vous avez écrite ci-dessus.

11. **Entraînement par mini-batchs.** Implémentez la descente de gradient par mini-batchs. Utilisez une taille de batch de 64 et itérez sur tout le dataset d'entraînement pendant 10 epochs. Au début de chaque epoch, vous devez randomiser l'ordre des données d'entraînement à l'aide de `np.shuffle`. Suivez et tracez la perte d'entraînement et de validation à chaque epoch, et affichez le temps d'exécution d'une epoch. Une epoch ne devrait pas dépasser quelques secondes. Pour améliorer la performance, vous pouvez essayer de *vectoriser* votre code.
12. **Évaluation.** Évaluez la précision de votre modèle sur le dataset de test. Vous devriez obtenir une précision supérieure à 80%.

Questions supplémentaires

13. **Réglage du taux d'apprentissage** :
    - Expérimentez différents taux d'apprentissage (`0,01`, `0,1`, `1,0`) et observez leur effet sur la convergence.
14. **Autres données** : Le dataset MNIST contient des images de chiffres manuscrits. Téléchargez ce dataset (par exemple depuis https://ossci-datasets.s3.amazonaws.com/mnist/) et entraînez un nouveau modèle. Quelle est la performance, comparée au dataset Fashion-MNIST ?