AMAL - TP 1
Définition de fonctions en `pyTorch`

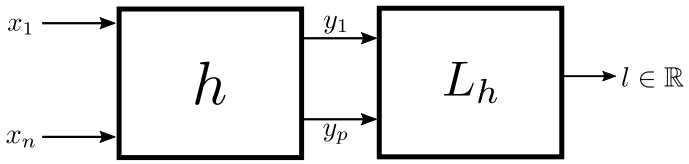
Nicolas Baskiotis - Edouard Oyallon - Benjamin Piwowarski - Laure Soulier

2021-2022

Notations

x	un scalaire
$\mathbf{x}$	un vecteur dans $\mathbb{R}^n$
$\mathbf{X}$	une matrice dans $\mathbb{R}^{n\times p}$
$\frac{\partial f}{\partial x}$	La dérivée partielle de f par rapport à x.
$\frac{\partial f}{\partial \mathbf{x}}$	Vecteur de dérivée partielles de f par rapport à $\mathbf{x}_i$.
$\frac{\partial f}{\partial \mathbf{X}}$	Vecteur de dérivée partielles de f par rapport à $\mathbf{X}_{ij}$.

1 Fonction dans un graphe



On suppose qu'on a une fonction $h : \mathbb{R}^n \mapsto \mathbb{R}^p$ et que sa sortie sert à calculer une fonction objectif L. Les entrés $\mathbf{x}_1, \ldots, \mathbf{x}_n$ peuvent provenir d'autre fonctions – mais ceci n'a pas d'importance. La sortie de la fonction h est un vecteur $h(\mathbf{x}) = \mathbf{y} \in \mathbb{R}^p$.

Question 1

Caluler la dérivée partielle

$$\frac{\partial L \circ h}{\partial \mathbf{x}_i}(\mathbf{x})$$

en fonction de $\frac{\partial h_j}{\partial \mathbf{x}_i}(\mathbf{x})$ et de $(\nabla L)_j = \frac{\partial L}{\partial \mathbf{y}_j}(h(\mathbf{x}))$ où $(\nabla L)_j$ permet de simplifier la notation. Pour une fonction h connue (ce qui est toujours le cas pour nous), on peut calculer explicitement sa dérivée par rapport à $\mathbf{x}_i$.

Le calcul précédent permet d'intuiter un résultat sur lequel nous reviendrons dans les TPs suivants : pour calculer le gradient d'une fonction composée de manière analytique, il suffit de connaître le gradient de chaque fonction par rapport à chacune de ses entrées et appliquer une dérivation chaînée.

Les plateformes modernes d'apprentissage profond exploitent ce résultat pour modulariser et rendre très efficace la programmation de nouvelles architectures. Pour cela, une *fonction* doit être implémentée non seulement le résultat de son application - appelé passe *forward* - mais également la dérivée partielle par rapport à chacune de ses entrées - appelée passe *backward*. En particulier,

Forward Calcul de $y = h(\mathbf{x})$ en fonction de $\mathbf{x}$

Backward Calcul de $\frac{\partial h}{\partial \mathbf{x}}(\mathbf{x})$ en fonction de $\nabla L(\mathbf{y})$ et de données dérivées de la phase forward ($\mathbf{x}$). Certains calculs partiels peuvent être également conservés pour accélérer les calculs.

2 Application : Régression Linéaire

2.1 Calcul du gradient (scalaire)

Le modèle de régression linéaire peut être vu comme la composition de deux éléments :

— une fonction linéaire, responsable du calcul de la prédiction :

$$\hat{y} = f(\mathbf{x}, \mathbf{w}, b) = \mathbf{x} \cdot \mathbf{w} + b$$

— un coût aux moindres carrés (MSE) : $mse(\hat{y}, y) = (\hat{y} - y)^2$

Afin d'effectuer une descente de gradient, il faut calculer le gradient du coût par rapport aux paramètres $\mathbf{w}$ et b. Nous allons calculer ce gradient grâce au chaînage des dérivées partielles. Afin de traiter le cas général, nous supposons que le coût est défini de la manière suivante :

$$C(\mathbf{x}, \mathbf{w}, b, y) = mse(f(\mathbf{x}, \mathbf{w}, b), y)$$

où $\mathbf{x}$ est l'entrée, y la sortie désirée dans $\mathbb{R}$, et $\mathbf{w} \in \mathbb{R}^n$ et $b \in \mathbb{R}$ les paramètres du modèle linéaire.

Afin de calculer le gradient de C par rapport à $\mathbf{w}$ et b, ce qui permet d'optimiser les paramètres via une descente de gradient, nous devons tout d'abord nous intéresser aux gradients des fonctions f et mse.

Question 2

Appliquer le résultat trouvé dans la question 1 pour calculer

— Les dérivées pour la fonction de coût MSE

$$\frac{\partial L \circ mse}{\partial y}(\hat{y}, y) \text{ et } \frac{\partial L \circ mse}{\partial \hat{y}}(\hat{y}, y)$$

— Les dérivées de la fonction linéaire f par rapport à ses entrées $\mathbf{x}$ ($\mathbf{w}$ et b sont obtenus de manière similaire)

$$\frac{\partial L \circ f}{\partial \mathbf{x}_j}(\mathbf{x}, \mathbf{w}, b)$$

La fonction L introduite ici est générique et est présente uniquement pour bien montrer le fonctionnement du chaînage des dérivées. Par rapport au problème qui nous intéresse, dans le premier cas comme la fonction mse est la dernière de étape de calcul, on peut considérer que L correspond à l'identité ; dans le deuxième cas il s'agit bien sûr de la fonction $mse(\cdot, \hat{y})$.

2.2 Calcul du gradient (matriciel)

On considère maintenant un cas plus général, qui sera celui utilisé en PyTorch, où nous travaillerons avec un lot (*batch*) de q exemples (au lieu d'un) et p sorties (au lieu d'une).

— Nous avons toujours une fonction linéaire, responsable du calcul de la prédiction :

$$\hat{\mathbf{Y}} = f(\mathbf{X}, \mathbf{W}, \mathbf{b}) = \mathbf{X}\mathbf{W} + \begin{pmatrix} \mathbf{b} \\ \vdots \\ \mathbf{b} \end{pmatrix}$$

où $\mathbf{X} \in \mathbb{R}^{q \times n}$ sont les entrées, $\mathbf{W} \in \mathbb{R}^{n \times p}$ est la matrice de poids, et $\mathbf{b} \in \mathbb{R}^{1 \times p}$ est le biais [1]. La sortie $\mathbf{Y}$ est donc une matrice dans $\mathbb{R}^{q \times p}$.

— un coût aux moindres carrés (MSE) que l'on généralise à un lot d'exemples et plusieurs sorties : $mse(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{q}\|\hat{\mathbf{Y}} - \mathbf{Y}\|^2$. Notez que l'on divise par le nombre d'exemple, ce qui permet d'avoir une magnitude du gradient qui ne varie pas en fonction du nombre d'exemples.

Le coût C est alors défini comme

$$C(\mathbf{X}, \mathbf{W}, \mathbf{b}, \mathbf{Y}) = mse(f(\mathbf{X}, \mathbf{W}, \mathbf{b}), \mathbf{Y})$$

1. Avec des tenseurs PyTorch (similaires aux tenseurs numpy), cela s'écrit `X @ W + b` en utilisant la multiplication matricielle et le *broadcast*

Question 3

Étendre les résultats obtenus dans la question précédente pour calculer

— Les dérivées pour la fonction de coût MSE

$$\frac{\partial L \circ mse}{\partial \mathbf{Y}_{ij}}(\hat{\mathbf{Y}}, \mathbf{Y})$$

et

$$\frac{\partial L \circ mse}{\partial \hat{\mathbf{Y}}_{ij}}(\hat{\mathbf{Y}}, \mathbf{Y})$$

— Les dérivées de la fonction linéaire f par rapport à ses entrées

$$\frac{\partial L \circ f}{\partial \mathbf{X}_{ij}}(\mathbf{X}, \mathbf{W}, \mathbf{b}),\ \frac{\partial L \circ f}{\partial \mathbf{W}_{ij}}(\mathbf{X}, \mathbf{W}, \mathbf{b}) \text{ et } \frac{\partial L \circ f}{\partial \mathbf{b}_{i}}(\mathbf{X}, \mathbf{W}, \mathbf{b})$$

Afin d'optimiser les calculs, et d'exploiter au mieux les architectures parallèles (GPU), il faut ensuite essayer d'exprimer au maximum les calcul de manière matricielle.

Question 4

Utiliser une écriture matricielle pour exprimer les résultats précédents, i.e. calculer

— Les dérivées pour la fonction de coût MSE

$$\frac{\partial L \circ mse}{\partial \mathbf{Y}}(\hat{\mathbf{Y}}, \mathbf{Y})$$

et

$$\frac{\partial L \circ mse}{\partial \hat{\mathbf{Y}}}(\hat{\mathbf{Y}}, \mathbf{Y})$$

— Les dérivées de la fonction linéaire f par rapport à ses entrées

$$\frac{\partial L \circ f}{\partial \mathbf{X}}(\mathbf{X}, \mathbf{W}, \mathbf{b}),\ \frac{\partial L \circ f}{\partial \mathbf{W}}(\mathbf{X}, \mathbf{W}, \mathbf{b}) \text{ et } \frac{\partial L \circ f}{\partial \mathbf{b}}(\mathbf{X}, \mathbf{W}, \mathbf{b})$$

2.3 Calcul du gradient de C

Maintenant que nous avons calculé toutes les dérivées partielles, il faut montrer comment calculer la dérivée de C par rapport à chacune de ses entrées.

Question 5

Donner les formules qui permettent de calculer

$$\frac{\partial C}{\partial \mathbf{W}} \text{ et } \frac{\partial C}{\partial \mathbf{b}}$$

Vous pouvez utiliser la fonction $mse_{\mathbf{Y}}(\hat{\mathbf{Y}}) = mse(\hat{\mathbf{Y}}, \mathbf{Y})$.

3 Implémentation des fonctions

`PyTorch` utilise une classe abstraite `Function` dont sont héritées toutes les fonctions et qui nécessite l'implémentation de ces deux méthodes :

- méthode `forward(ctx, *inputs)` : calcule le résultat de l'application de la fonction
- méthode `backward(ctx, *grad_outputs)` : calcule le gradient partiel par rapport à chaque entrée de la méthode `forward` ; le nombre de `grad_outputs` doit être égale aux nombre de sorties de `forward` (pourquoi ?) et le nombre de sorties doit être égale aux nombres de `inputs` de `forward`.

Pour des raisons d'implémentation, les deux méthodes doivent être statiques. Le premier paramètre `ctx` permet de sauvegarder un contexte lors de la passe `forward` (par exemple les tenseurs d'entrées) et il est passé lors de la passe `backward` en paramètre afin de récupérer les valeurs. **Attention :** le contexte doit être unique pour chaque appel de `forward`.

Important

Sur votre *propre ordinateur*, vous pouvez reproduire l'environnement de travail en tapant la commande suivante (attention, mettre à jour avant chaque TP)

```
pip install -r http://webia.lip6.fr/~baskiotisn/requirements-amal.txt
```

Sur les machines des salles de TPs, cet environment est déjà présent.

```
source /users/nfs/Enseignants/baskiotisn/venv/3.7/amal/bin/activate
```

Si vous utilisez un éditeur (VS Code), n'oubliez pas spécifier l'environment pour pouvoir développer plus facilement.

Question 6

Télécharger le code fourni sur le site de l'UE. Il contient un squelette pour l'implémentation des fonctions, une classe `Context` pour faire fonctionner votre code et un exemple d'utilisation.

1. Implémenter en `PyTorch` les classes nécessaires pour la régression linéaire : la fonction linéaire et la fonction de coût MSE. Utiliser bien les outils propres à `pyTorch`, en particulier des `Tensor` et pas des matrices `numpy`. Assurez vous que vos fonctions prennent en entrée des batchs d'exemples (matrice 2D) et non un seul exemple (vecteur). N'hésiter pas à prendre un exemple et de déterminer les dimensions des différentes matrices en jeu.
2. `PyTorch` vous permet de vérifier le calcul de vos dérivées grâce à la fonction `gradcheck` qui effectue une approximation numérique par différence finie. Un exemple d'utilisation est donné dans le code source. Tester vos fonctions avec cet outil, en utilisant `tp1_gradcheck.py` que vous devez compléter.

4 Descente de gradient

La fonction f correspond au prédicteur (ou classifieur) que l'on souhaite apprendre, i.e. trouver les paramètres $\mathbf{W}$ et $\mathbf{b}$ qui minimisent le risque d'erreur sur les prédictions. Pour chaque ensemble d'exemples $\mathbf{X}$ donné, la sortie du prédicteur $\hat{\mathbf{Y}} =$ est comparée à la sortie attendue $\mathbf{Y}$ grâce à la fonction de coût $L(\hat{\mathbf{y}}, \mathbf{y})$ qui permet de quantifier l'erreur. Pour les paramètres $\mathbf{W}$ et $\mathbf{b}$, le coût associé à l'ensemble d'apprentissage est

$$C(\mathbf{W}, \mathbf{b}) = mse(f(\mathbf{X}, \mathbf{W}, \mathbf{b}), \mathbf{Y})$$

.

Dans le contexte de la minimisation du risque empirique, la formalisation du problème d'apprentissage est la suivante : trouver le paramètre optimal $\mathbf{w}^*$ qui minimise le coût de prédiction sur l'ensemble des données d'apprentissage

$$\mathbf{W}^*, \mathbf{b}^* = argmin_{\mathbf{w},\mathbf{b}} L(\mathbf{W}, \mathbf{b}) = argmin_{\mathbf{W},\mathbf{b}}\ mse(f(\mathbf{X}, \mathbf{W}, \mathbf{b}), \mathbf{Y})$$

.

Un algorithme d'apprentissage classique pour optimiser le paramètre est l'algorithme de descente de gradient. Il consiste à mettre à jour itérativement le paramètre $\mathbf{w}$ selon la formule :

$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}, \mathbf{b})$$
$$\mathbf{b} \leftarrow \mathbf{b} - \epsilon \nabla_{\mathbf{b}} L(\mathbf{W}, \mathbf{b})$$

avec ϵ un paramètre appelé le pas de gradient.

Question 7

Implémenter la descente de gradient en partant du fichier `tp1_descente.py`.