AMAL - TP 1
Définition de fonctions en `pyTorch`

Nicolas Baskiotis - Benjamin Piwowarski

2019-2020

Notations, définitions et rappels

Les notations suivantes seront utilisées dans la plupart des TPs :
— un espace de représentation $\mathbb{R}^d$ à d dimensions
— un espace de sortie $\mathbb{R}^c$ à c dimensions
— un exemple $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$, son étiquette $\mathbf{y} = (y_1, \ldots, y_c) \in \mathbb{R}^c$
— un ensemble d'apprentissage $\mathcal{X} = \{(\mathbf{x}^i, \mathbf{y}^i) \in \mathbb{R}^d \times \mathbb{R}^c\}_{i=1}^N$ de N exemples
— une fonction de coût $L : \mathbb{R}^c \times \mathbb{R}^c \to \mathbb{R}$
— une fonction $f_{\mathbf{w}} : \mathbb{R}^d \to \mathbb{R}^c$ paramétrée par $\mathbf{w}$.

La fonction $f_{\mathbf{w}}$ correspond au prédicteur (ou classifieur) que l'on souhaite apprendre, i.e. trouver le paramètre $\mathbf{w}$ qui minimise le risque d'erreur sur les prédictions. Pour un exemple $\mathbf{x}$ donné, la sortie du prédicteur $\hat{\mathbf{y}} = f_{\mathbf{w}}(\mathbf{x})$ est comparée à la sortie attendu $\mathbf{y}$ grâce à la fonction de coût $L(\hat{\mathbf{y}}, \mathbf{y})$ qui permet de quantifier l'erreur. Pour un paramètre $\mathbf{w}$, le coût associé à l'ensemble d'apprentissage est

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} L(f_{\mathbf{w}}(\mathbf{x}^i), \mathbf{y}^i)$$

.

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

$$\mathbf{w}^* = argmin_{\mathbf{w}} L(\mathbf{w}) = argmin_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} L(f_{\mathbf{w}}(\mathbf{x}^i), \mathbf{y}^i)$$

.

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 :

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

avec ϵ un paramètre appelé le pas de gradient.

Trois variantes sont très utilisées :
— gradient **batch** : le gradient est calculé sur la somme des coûts de tous les exemples d'apprentissage, i.e. $L(\mathbf{w})$;
— gradient **stochastique** : à chaque itération, un exemple est tiré aléatoirement ; le coût et le gradient du coût n'est calculé que pour cet exemple et la mise-à-jour du paramètre est fait selon ce gradient ;
— gradient **mini-batch** : l'ensemble d'apprentissage est partitionné en petits sous-ensembles appelés *mini-batch* ; à chaque itération le coût et le gradient est calculé sur un des sous-ensembles, et le paramètre est mis-à-jour selon ce gradient. La taille des mini-batchs est un paramètre supplémentaire, généralement quelques dizaines d'exemples.

1 Régression linéaire et descente de gradient

Dans cette partie, nous allons nous focaliser sur l'implémentation en `pyTorch` d'un modèle linéaire couplé à un coût aux moindres carrés pour résoudre les problèmes de classification et régression.

Le modèle de régression linéaire peut être vu comme la composition de deux éléments :
— une fonction linéaire, responsable du calcul de la prédiction :

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}, b) = \begin{pmatrix} \sum_i \mathbf{W}_{i,1} x_i \\ \vdots \\ \sum_i \mathbf{W}_{i,c} x_i \end{pmatrix}^\top = \mathbf{x}\mathbf{W}^\top + b$$

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

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

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

où g est une fonction quelconque à valeurs réelles.

Calcul du gradient

Le calcul du gradient dans les réseaux de neurones est basé sur le calcul de dérivées partielles de fonction chaînées, i.e. la rétro-propagation du gradient. Nous allons donc décomposer le calcul de la façon suivante : on suppose que la fonction dont on veut calculer

la dérivée est $h : \mathbb{R}^n \to \mathbb{R}^p, \mathbf{x} \to \mathbf{y}$ et que le coût est calculé depuis la valeur $h(x)$ par une fonction $L_h(\mathbf{y})$, i.e.

$$L(\mathbf{x}) = L_h(h(\mathbf{x}))$$

Question 1. Calculer la dérivée partielle

$$\frac{\partial L_h \circ h}{\partial \mathbf{x}_k}(\mathbf{x})$$

en fonction de $\frac{\partial h}{\partial \mathbf{x}_i}(\mathbf{x})$ et de $\frac{\partial L_h}{\partial \mathbf{y}_j}(h(\mathbf{x}))$

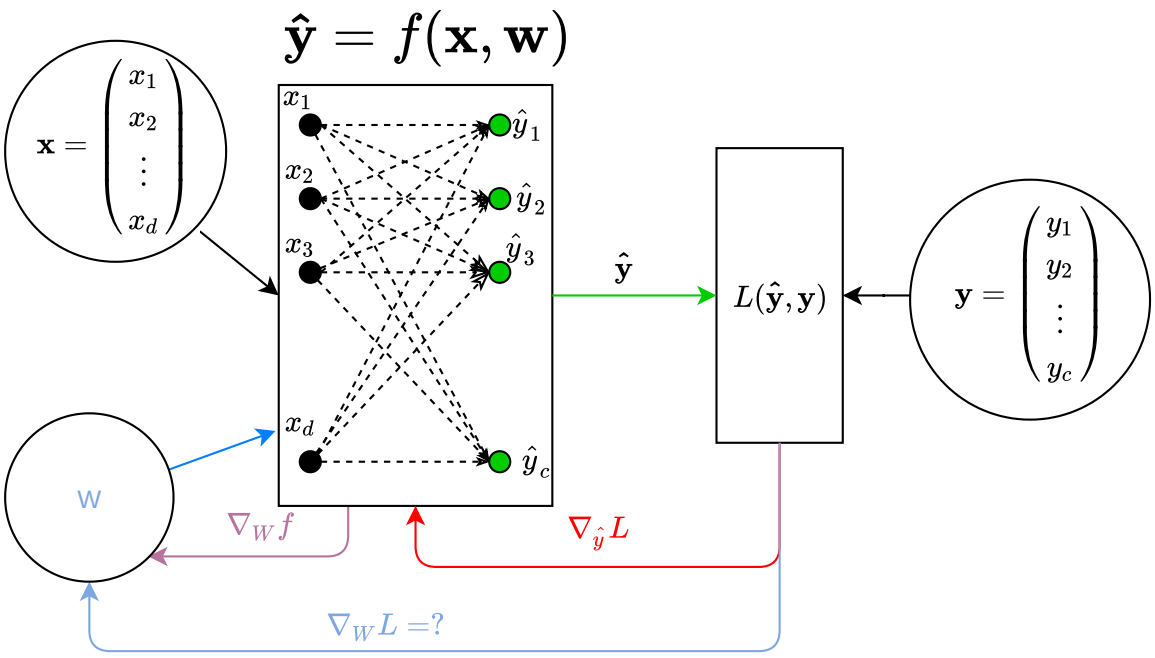
Question 2. Appliquer ce résultat en identifiant h et L_h dans les deux cas pour calculer (on prend le cas $c = 1$)

$$\frac{\partial g \circ mse}{\partial \mathbf{y}}(\mathbf{y}),\ \frac{\partial g \circ mse \circ f}{\partial \mathbf{x}_i}(\mathbf{x}, \mathbf{w}, b),\ \frac{\partial g \circ mse \circ f}{\partial \mathbf{w}_i}(\mathbf{x}, \mathbf{w}, b) \text{ et } \frac{\partial g \circ mse \circ f}{\partial b}(\mathbf{x}, \mathbf{w}, b)$$

puis passer à la notation vectorielle.

Question 3 (Cas général - optionnel). Étendre les résultats obtenus pour obtenir une forme matricielle (taille du batch $b > 1$, $c > 1$), i.e. pour calculer

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



Implémentation

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 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 $\frac{\partial L_h}{\partial \mathbf{y}}(y)$ et de données dérivées de la phase forward ($\mathbf{x}$ dans le cas le plus simple, mais certains calculs partiels peuvent être également conservés)

`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.

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.
3. Implémenter l'algorithme de descente de gradient stochastique.
4. Tester votre implémentation avec les données de Boston Housing. Tracer la courbe du coût en apprentissage et celle en test. Utiliser pour cela de préférence `tensorboard`.

Utilisez pour l'instant seulement la fonction `add_scalar` après avoir créé un fichier de log grâce à la commande `SummaryWriter(`*`path`*`)`.

5. Implémenter une descente de gradient batch et une mini-batch. Comparer la vitesse de convergence et les résultats obtenus.