TP 4
Modèles probabilistes discriminants pour le Machine Learning

Elie Azeraf

Ce TP nécessite l'installation de python 3 avec la librairie torch (accessible ici), torchvision, et matplolib. Aucune autre librairie n'est autorisé.
Il est à rendre, au choix, soit dans un fichier Jupyter notebook, soit un PDF avec les codes joints en fichiers .py.
Pour être réalisé, il nécessite le dossier *TP4_functions*.
Il est à rendre pour le 10/11/2021 avant 12h00, les seules commentaires écrits requis sont ceux demandés. Tant que le code est clair, vous n'avez pas besoin d'expliquer ce que vous faites.

1 Exercice 1 (3 point) La fonction softmax

Coder la fonction softmax, optimisée tel que vue en cours.
Uniquement les fonctions torch.exp, torch.max et torch.sum sont autorisées.

Vous testerez votre fonction avec les valeurs suivantes:

```
x = torch.tensor([5., 1., 2., -1.])
softmax(x)
```

Cela devrait vous retourner le vecteur $[0.9341, 0.0171, 0.0465, 0.0023]$.

Testez également:

```
x = torch.tensor([100000., 120000., 200000., 200000.])
softmax(x)
```

Cela devrait vous retourner les valeurs $[0.8705, 0.1178, 0.0059, 0.0059]$.

2 Préambule: comment utiliser une fonction d'optimisation de gradient avec PyTorch ?

Durant le cours, vous avez vu comment appliquer la descente de gradient stochastique (SGD). Cette technique, SGD, est la plus simple de descente de gradient, et il en existe d'autres méthodes d'optimisation: Adam [1], RMSProp [2], etc...
Ils utilisent un ensemble de techniques permettant d'être "meilleur" que SGD, comme l'utilisation

du momentum par exemple. Une review est disponible dans [3]. En général, le meilleur d'entre eux est Adam, que nous allons donc utiliser.

En reprenant l'exemple du cours, voici comment appliquer Adam avec PyTorch [4]:

```
import torch
import matplotlib.pyplot as plt
from torch import optim

def g(x, a, b):
    return a*x + b

train_set = torch.tensor([(1, 3), (2, 8)])

def L(g, train_set, a, b):
    mse = 0
    for line in train_set:
        x = line[0]
        y = line[1]
        mse += (y - g(x, a, b))**2

    mse /= train_set.shape[0]
    return mse

ak = torch.tensor([3.], requires_grad=True)
bk = torch.tensor([1.], requires_grad=True)

alpha = 0.01

K = 500
list_L = torch.zeros(K)
optimizer = optim.Adam([ak, bk], lr=alpha)

for k in range(K):
    optimizer.zero_grad()
    loss = L(g, train_set, ak, bk)
    loss.backward()
    optimizer.step()

    list_L[k] = loss.item()

    if k % 50 == 0:
        print(k)
        print("Loss:", list_L[k])
        print("theta", ak, bk, "\n")


plt.plot(list_L)
plt.show()

print("theta", ak, bk, "\n")
```

Tout au long de ce TP, vous devrez appliquer exactement ce schéma pour entraîner les paramètres de vos différents modèles.

3 Exercice 2 (10 points)
Classification de chiffre avec la régression logistique

Durant cette exercice, vous allez utiliser le très populaire dataset MNIST. Il s'agit du dataset le plus connu en Computer Vision: il est constitué de 60 000 chiffres écrit à la main:

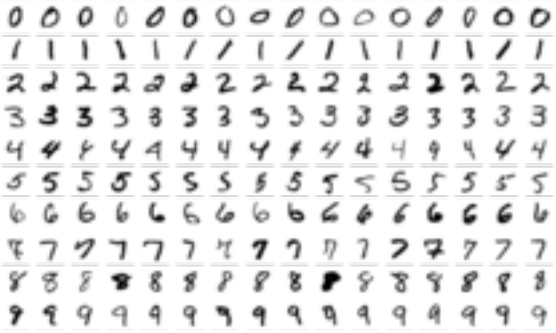

Figure 1: Quelques exemples du dataset MNIST

Nous nous prenons comme dataset d'entrainement les 50 000 premiers exemples du dataset, et le reste comme dataset de test.

Votre objectif durant cet exercice est de coder un modèle de régression logistique sur ce dataset.

3.1 Prise en main du dataset

Vous chargerez le dataset grâce aux fonctions suivantes, permettant de télécharger et formater les données:

```
dataset = torchvision.datasets.MNIST("./", download = True)
train_set_y, train_set_x = dataset.data[:50000], dataset.targets[:50000]
test_set_y, test_set_x = dataset.data[50000:], dataset.targets[50000:]

train_set_y = (train_set_y.view(50000, -1) + 0.0)
mean_train, std_train = torch.mean(train_set_y), torch.std(train_set_y)
train_set_y = (train_set_y - mean_train)/std_train

train_set = []
for i in range(50000):
    train_set.append([train_set_x[i], train_set_y[i]])

test_set_y = (test_set_y.view(10000, -1) + 0.0)
mean_test, std_test = torch.mean(test_set_y), torch.std(test_set_y)
test_set_y = (test_set_y - mean_test)/std_test

test_set = []
for i in range(10000):
    test_set.append([test_set_x[i], test_set_y[i]])
```

Ainsi, les outputs train_set et test_set sont vos variables d'intérêt. Chaque élément est une liste composé de la valeur du chiffre dessiné, et d'un vecteur de taille 784 représentant les pixels du chiffre en niveau de gris (à remettre au format (28, 28) pour être représenter).

Il est possible d'afficher l'un des exemple de la manière suivante:

```
exemple = 1
```

```
plt.imshow(train_set[exemple][1].view(28, 28))
plt.show()
print(train_set[exemple][0])
```

3.2 Votre objectif !

Pour cet exercice, vous devrez coder une régression logistique (n'hésitez pas à utiliser la fonction softmax de début de TP, ou alors la fonction torch.softmax) ayant pour input le vecteur d'observations de taille 784, et devant avoir comme output des probabilités que l'image appartienne à chacune des classes.

Cette régression logistique aura comme paramètres initiaux W^{LR} et b^{LR} chargés de la manière suivante:

```
W_lr = torch.load("TP4_functions/W_lr.pt").requires_grad_(True)
b_lr = torch.load("TP4_functions/b_lr.pt").requires_grad_(True)
```

Voici la configuration requise:

- Optimizer: Adam
- Learning rate: 5×10^{-5}
- Fonction de coût: Cross-Entropy $L_{CE} = -\frac{1}{N} \sum_{x,y} log(f(y)[x])$, avec N le nombre d'exemples, $f(y)$ renvoie les probabilités des différents outputs, et x le vrai label
- Nombre d'itérations: 1000
- Mini-batch size: 256

Il vous sera demandé d'afficher la valeur de la fonction de coût pour les 100 premiers exemples du train set (valeur attendue: 2.4652). De plus, vous afficherez la loss sur le batch ainsi que le score (pourcentage d'exemples bien labélisées) sur le test set toutes les 50 itérations.

Enfin, vous afficher le graphique des loss, ainsi que le score final en fin d'entraînement (approximativement égal à 87%).

4 Exercice 3 (7 points) Maximum Entropy Markov Model pour le POS tagging

4.1 Nos données

Le dernier exercice consiste à appliquer le MEMM avec des fonctions modélisé par des régressions logistiques pour le POS tagging sur le dataset CoNLL 2000, étudié au TP3.

En premier lieu, une étape importante du processus consiste à convertir un mot en vecteur de nombres. En effet, notre MEMM ne peut pas considérer des chaînes de caractère en input. Nous utilisons donc un algorithme de Word Embedding (convertissant un mot en vecteur) appelé GloVe [5], qui permet de convertir n'importe quel mot en vecteur de dimension 100.

Ainsi, le chargement des datasets se fait de la manière suivante:

```
from TP4_functions.load_conll2000 import load_conll2000_glove

```

```
path = "TP4_functions/"
Omega_X, train_set, test_set = load_conll2000_glove(path)
```

Omega_X contient la liste des différents POS tags, train_set et test_set sont nos données d'intérêt. Ici, chaque élément d'un jeu de données est formé de deux éléments: [une liste des POS tags de taille T, un vecteur de taille $T \times 100$]. N'hésitez pas à utiliser la commande:

```
train_set[0]
```

par exemple, pour vous faire une idée.

4.2 Votre objectif !

Votre mission pour ce nouvel exercice est de coder le MEMM avec les fonctions L^1 et L^2 modélisés par des régressions logistiques. Les valeurs initiales des paramètres devront être chargés ainsi:

```
W_lr1 = torch.load("TP4_functions/W_lr1.pt").requires_grad_(True)
b_lr1 = torch.load("TP4_functions/b_lr1.pt").requires_grad_(True)
W_lr2 = torch.load("TP4_functions/W_lr2.pt").requires_grad_(True)
b_lr2 = torch.load("TP4_functions/b_lr2.pt").requires_grad_(True)
```

Voici les différentes configurations:

- Optimizer: Adam
- Learning rate: 5×10^{-4}
- Fonction de coût: Cross-Entropy
- Nombre d'itérations: 1000
- Mini-batch size: 32

Au cours de cet exercice, il vous est demandé:

- D'afficher la valeur de la fonction de coût avec les paramètres initiaux pour les dix premiers éléments du train set (valeur attendue aux alentours de 2.7) avant l'entraînement.
- Toutes les 50 itérations de votre algorithme, affichez la loss sur le mini-batch et le score sur le test set.
- A la fin de votre learning, affichez la loss sur les 1000 itérations, et le score final sur le test set.

N'hésitez pas à vectoriser un maximum vos calculs tout au long de ce TP.

Bon courage ! N'hésitez pas en cas de questions, ou par mail à azeraf.elie@telecom-sudparis.eu.

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[2] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[3] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[5] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.