INF8225 TP1 H19

Date limite : 12h le 4 février 2019

1 Partie I (10 points)

L'objectif de la partie I du travail pratique est de permettre à l'étudiant de se familiariser avec les réseaux Bayésiens et la librairie Numpy. Considérons le réseau Bayésien ci-dessous. Il s'agit d'un exemple bien connu appelé « wet grass », souvent utilisé pour illustrer l'effet de « explaining away ».

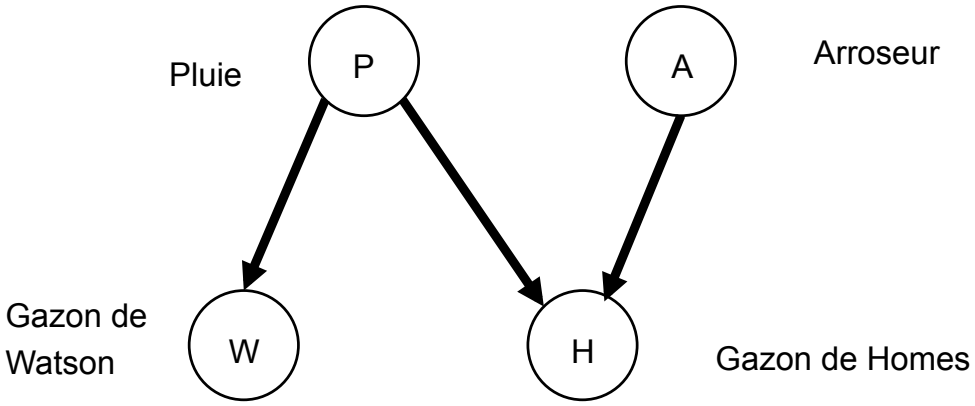

Figure 1: « The Wetgrass Network »

Voici les tables de probabilités conditionnelles fournies:

- La probabilité qu'il ait plu: $Pr(P = 1) = 0.2$
- La probabilité que l'arroseur ait fonctionné: $Pr(A = 1) = 0.1$
- La probabilité que le gazon de Watson soit mouillé...
  - ... sachant qu'il a plu est $Pr(W = 1|P = 1) = 1$
  - ... sachant qu'il n'a **pas** plu: $Pr(W = 1|P = 0) = 0.2$
- La probabilité que Holmes remarque que son gazon est mouillé...
  - ... sachant que l'arroseur a fonctionné et qu'il n'a **pas** plu: $Pr(H = 1|P = 0, A = 1) = 0.9$
  - ... sachant que l'arroseur n'a **pas** fonctionné et qu'il n'a **pas** plu: $Pr(H = 1|P = 0, A = 0) = 0$
  - ... sachant qu'il a plu, et que l'arroseur ait ou pas fonctionné: $Pr(H = 1|P = 1, A = 0, 1) = 1$

1.1 Description des tâches

Calculez les probabilités suivantes en utilisant le squelette du code python fourni ci-après:

a) $Pr(W = 1)$

b) $Pr(W = 1|H = 1)$

c) $Pr(W = 1|H = 1, A = 0)$

d) $Pr(W = 1|A = 0)$

e) $Pr(W = 1|P = 1)$

```python
import numpy as np
import matplotlib.pyplot as plt

# les arrays sont batis avec les dimensions suivantes:
# pluie, arroseur, watson, holmes
# et chaque dimension: faux, vrai

prob_pluie = np.array([0.8, 0.2]).reshape(2, 1, 1, 1)
print ("Pr(Pluie)={}\n".format(np.squeeze(prob_pluie)))
prob_arroseur = np.array([0.9, 0.1]).reshape(1, 2, 1, 1)
print ("Pr(Arroseur)={}\n".format(np.squeeze(prob_arroseur)))
watson = np.array([[0.8, 0.2], [0, 1]]).reshape(2, 1, 2, 1)
print ("Pr(Watson|Pluie)={}\n".format(np.squeeze(watson)))
holmes = None # TODO
print ("Pr(Holmes|Pluie,arroseur)={}\n".format(np.squeeze(holmes)))

watson[0,:,1,:]  # prob watson mouille - pluie
(watson * prob_pluie).sum(0).squeeze()[1]  # prob gazon watson mouille

holmes[0,1,0,1]  # prob gazon holmes mouille si arroseur - pluie
```

Listing 1: Réseau Bayésien

Donnez une équation pour chaque calcul et une explication de quelques phrases concernant les calculs en termes de concepts d'indépendance conditionnelle et de dépendance conditionnelle quand ces concepts sont applicables. Vous pouvez calculer les probabilités a), b), c), d), et e) en utilisant les *arrays* suivantes et les propriétés de « broadcasting » de numpy. Vous devez soumettre votre code et inclure dans le rapport vos explications écrites et les équations.

2 Partie II (20 points)

L'objectif de la partie II du travail pratique est de permettre à l'étudiant de se familiariser avec l'apprentissage automatique via la régression logistique. Nous allons donc résoudre un problème de classification d'images en utilisant l'approche de descente du gradient (*gradient descent*) pour optimiser la log-vraisemblance négative (*negative log-likelihood*) comme fonction de perte.

L'algorithme à implémenter est une variation de descente de gradient qui s'appelle l'algorithme de descente de gradient stochastique par mini-ensemble (*mini-batch stochastic gradient descent*). Votre but est d'écrire un programme en Python pour optimiser les paramètres d'un modèle étant donné un ensemble de données d'apprentissage, en utilisant un ensemble de validation pour déterminer quand arrêter l'optimisation, et finalement de montrer la performance sur l'ensemble du test.

2.1 La régression logistique et le calcul du gradient

Il est possible d'encoder l'information concernant l'étiquetage avec des vecteurs multinomiaux (*one-hot vectors*), c.-à-d. un vecteur de zéros avec un seul 1 pour indiquer quand la classe $C = k$ dans la dimension k. Par exemple, le vecteur $\mathbf{y} = [0, 1, 0, \cdots, 0]^T$ pour représenter la deuxième classe. Les caractéristiques (*features*) sont données par des vecteurs $\mathbf{x}_i \in \mathbb{R}^D$. En définissant les paramètres de notre modèle comme : $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_K]^T$ et $\mathbf{b} = [b_1, b_2, \cdots b_K]^T$ et la fonction *softmax* comme fonction de sortie, on peut exprimer notre modèle sous la forme :

$$p(\mathbf{y}|\mathbf{x}) = \frac{\exp(\mathbf{y}^T\mathbf{W}\mathbf{x} + \mathbf{y}^T\mathbf{b})}{\sum_{\mathbf{y}_k \in \mathcal{Y}} \exp(\mathbf{y}_k^T\mathbf{W}\mathbf{x} + \mathbf{y}_k^T\mathbf{b})} \tag{1}$$

$$= \frac{1}{Z(\mathbf{x})} \exp\left(\mathbf{y}^T\mathbf{W}\mathbf{x} + \mathbf{y}^T\mathbf{b}\right) \tag{2}$$

L'ensemble de données consiste de n paires (*label*, *input*) de la forme $(\tilde{\mathbf{y}}_i, \tilde{\mathbf{x}}_i)_{i=1}^n$, où nous utilisons l'astuce de redéfinir $\tilde{\mathbf{x}}_i = [\tilde{\mathbf{x}}_i^T 1]^T$, et en utilisant la définition de la matrice de paramètres $\boldsymbol{\theta} \in \mathbb{R}^{K \times (D+1)}$ (voir des notes de cours pour la relation entre $\boldsymbol{\theta}$ et $\mathbf{W}$), nous avons :

$$\frac{\partial}{\partial \boldsymbol{\theta}} \log \prod_{i=1}^{N} P(\tilde{\mathbf{y}}_i|\tilde{\mathbf{x}}_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \frac{\partial}{\partial \boldsymbol{\theta}} \left\{ \log \left(\frac{1}{Z(\tilde{\mathbf{x}}_i)} \exp\left(\tilde{\mathbf{y}}_i^T \boldsymbol{\theta} \tilde{\mathbf{x}}_i\right) \right) \right\} \tag{3}$$

$$= \sum_{i=1}^{N} \left(\tilde{\mathbf{y}}_i \tilde{\mathbf{x}}_i^T - \sum_{\mathbf{y}_k \in \mathcal{Y}} P(\mathbf{y}_k|\tilde{\mathbf{x}}_i, \boldsymbol{\theta}) \mathbf{y}_k \tilde{\mathbf{x}}_i^T \right) \tag{4}$$

$$= \sum_{i=1}^{N} \tilde{\mathbf{y}}_i \tilde{\mathbf{x}}_i^T - \sum_{i=1}^{N} \hat{\mathbf{y}}_i \tilde{\mathbf{x}}_i^T \tag{5}$$

où $\hat{\mathbf{y}}_i$ est le vecteur de probabilités produit par le modèle pour l'exemple $\tilde{\mathbf{x}}_i$ et $\tilde{\mathbf{y}}_i$ est le vrai *label* pour ce même exemple.

2.2 Description des tâches

On vous demande de compléter l'extrait de code ci-dessous pour résoudre ce problème. Mettez à jour les paramètres de votre modèle avec la descente par *mini-batch*. Voir les notes de cours supplémentaires disponibles sur moodle pour plus de détails concernant cette méthode. Utilisez 20 *mini-batches*. Explorez quelques taux d'apprentissage et quelques autres grandeurs pour les *mini-batches* et discutez de leur impact.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()

X = digits.data

y = digits.target
```

```
y_one_hot = np.zeros((y.shape[0], len(np.unique(y))))
y_one_hot[np.arange(y.shape[0]), y] = 1  # one hot target or shape NxK


X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.3, random_state=42)

X_test, X_validation, y_test, y_validation = train_test_split(X_test, y_test, test_size=0.5,
    random_state=42)


W = np.random.normal(0, 0.01, (len(np.unique(y)), X.shape[1]))  # weights of shape KxL

best_W = None
best_accuracy = 0
lr = 0.001
nb_epochs = 50
minibatch_size = len(y) // 20

losses = []
accuracies = []

def softmax(x):
    # assurez vous que la fonction est numeriquement stable
    # e.g. softmax(np.array([1000, 10000, 100000], ndim=2))
    pass

def get_accuracy(X, y, W):
    pass

def get_grads(y, y_pred, X):
    pass

def get_loss(y, y_pred):
    pass

for epoch in range(nb_epochs):
    loss = 0
    accuracy = 0
    for i in range(0, X_train.shape[0], minibatch_size):
        pass # TODO
    losses.append(loss)  # compute the loss on the train set
    accuracy = None  # TODO
    accuracies.append(accuracy) # compute the accuracy on the validation set
    if accuracy > best_accuracy:
        pass  # select the best parameters based on the validation accuracy

accuracy_on_unseen_data = get_accuracy(X_test, y_test, best_W)
print(accuracy_on_unseen_data)  # 0.897506925208

plt.plot(losses)

plt.imshow(best_W[4, :].reshape(8,8))
```

Listing 2: Régression Logistique

2.3 Préparez un rapport sur votre code et vos expériences

Exécutez des expériences avec trois différents ensembles: un ensemble d'apprentissages avec 70% des exemples (choisis au hasard), un ensemble de validation avec 15% et un ensemble de test avec 15%. Créer un rapport avec vos expériences incluant les figures avec les courbes de log de vraisemblance negative moyenne pour l'ensemble de test et de validation après chaque epoch.

a) Montrez les résultats pour différents taux d'apprentissage, e.g. 0.1, 0.01, 0.001, et différentes tailles de mini-batch, e.g. 1, 20, 200, 1000.

b) Ajouter 8 dimensions supplémentaires à la fin de chaque exemple dans l'ensemble d'entraînement, l'ensemble de validation et l'ensemble de test. Remplissez ces dimensions avec des valeurs aléatoires en utilisant une distribution

aléatoire uniforme entre la plus petite valeur et la plus grande valeur parmi toutes les dimensions de l'ensemble de données d'origine. Ajoutez maintenant la régularisation du types *Elastic Net* à votre modèle. Ajustez les coefficients de ces deux nouveaux termes dans votre fonction de perte en utilisant l'ensemble de validation. Dans votre rapport, donnez l'équations pour le nouveau calcul de gradient. Quelle est la moyenne et la variance des poids de modèle associés à ces dimensions contenant des valeurs aléatoires par rapport à la moyenne et la variance des poids associés aux dimensions contenant des vraies données? Comparez ces moyennes et variances à un modèle appris sans régularisation. Pouvez-vous trouver une combinaison de valeurs pour les deux termes de régularisation qui ramènent à zéro les poids associés aux dimensions aléatoires ajoutées? Discutez de ces expériences dans vos propres mots et proposez quelques visualisations.

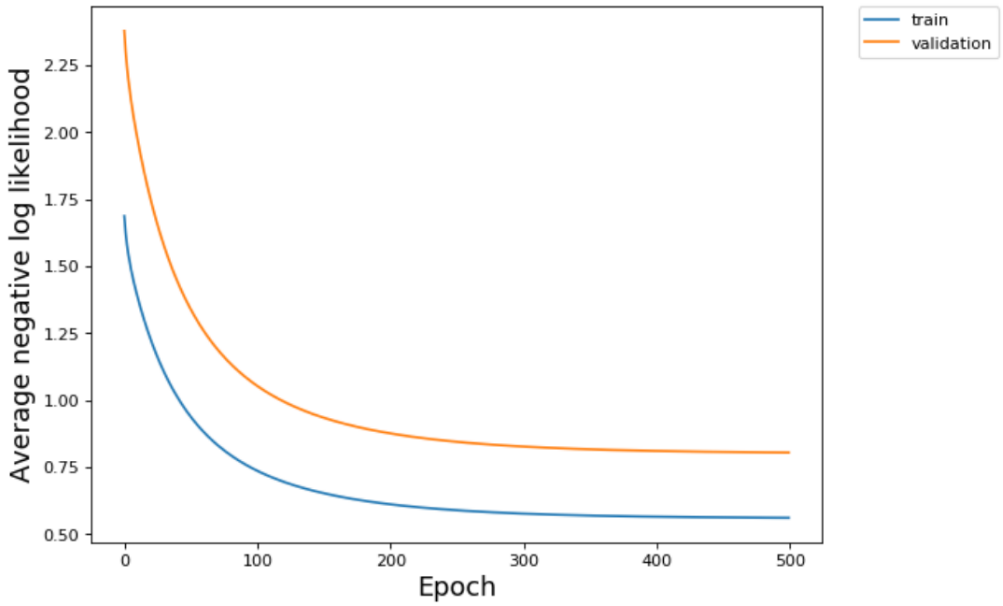

Figure 2: Courbes d'apprentissage.

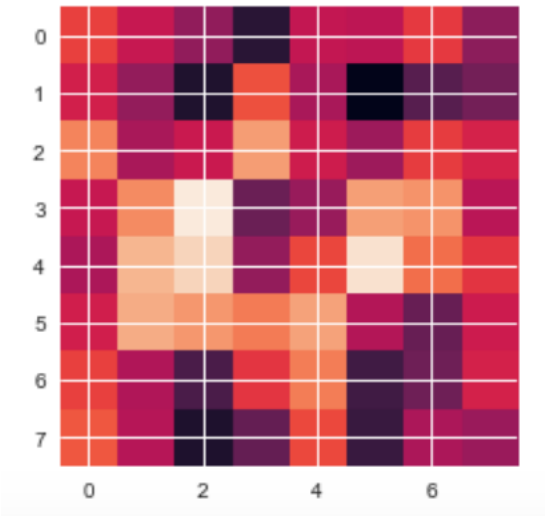

Figure 3: Exemple des poids appris pour le chiffre 4.