

INF8225 TP1 H22

Date limite : 8h30 le 28 février 2022

Remettez votre fichier collab en 2 formats: **.pdf** ET **.ipynb** (jupyter notebook format). Lien vers le Colab: https://colab.research.google.com/drive/19UqMDsk3lR0mt9C8gZ6_Y624G3oufHTI#scrollTo=x3wjnIDGHZj

1 Partie 1 (10 points)

L'objectif de la partie 1 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.

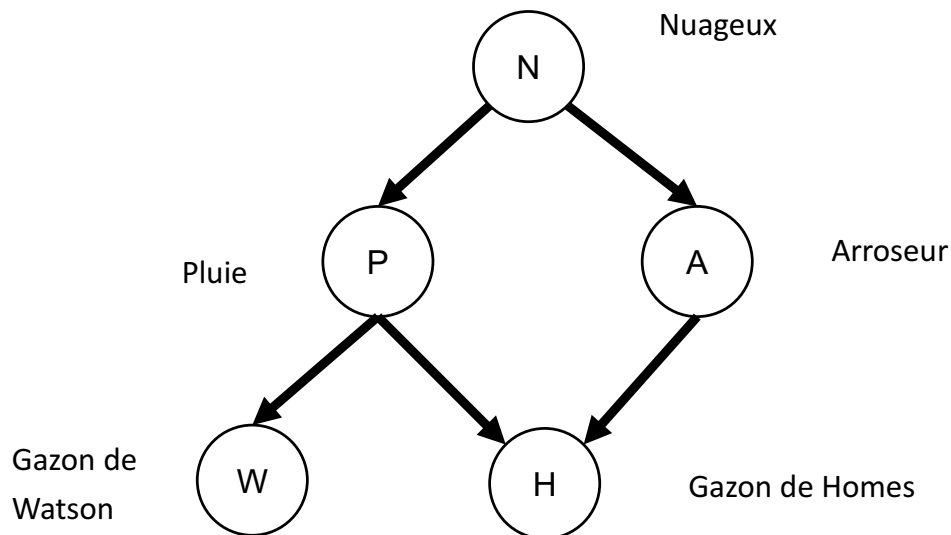


Figure 1: « The Wetgrass Network (v2) »

Voici les tables de probabilités conditionnelles fournies:

- La probabilité qu'il est Nuageux: $Pr(N = 1) = 0.2$
- La probabilité que l'arroseur a été utilisé sachant qu'il est nuageux ou non:
 $Pr(A = 1|N = 1) = 0.1$, $Pr(A = 1|N = 0) = 0.3$
- La probabilité qu'il ait plu, étant donné que le temps est nuageux:
 $Pr(P = 1|N = 1) = 0.4$, $Pr(P = 1|N = 0) = 0.1$
- 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

1.1.1 À l'aide de ces tables de probabilité conditionnelles, calculez les requêtes ci-dessous et commentez vos résultats

- a) $Pr(H = 1)$
- b) $Pr(H = 1|A = 1)$
- c) $Pr(H = 1|do(A = 1))$
- d) $Pr(H = 1|W = 1)$
- e) $Pr(H = 1|do(W = 1))$
- f) $Pr(W = 1|P = 1)$
- g) $Pr(W = 1|do(P = 1))$
- h) $Pr(H = 1|P = 1)$
- i) $Pr(H = 1|do(P = 1))$
- j) $Pr(P = 1|W = 1, H = 1, N = 1)$

1.1.2 Répondez aux questions suivantes

a) Vrai ou Faux:

- i) $H \perp\!\!\!\perp N | P$?
- ii) $H \perp\!\!\!\perp N | A$?
- iii) $W \perp\!\!\!\perp H | P$?
- iv) $P \perp\!\!\!\perp A | N$?
- v) $P \perp\!\!\!\perp A | N, H$?
- vi) $H \perp\!\!\!\perp N | A$?

b) Expliquez:

- i) Pourquoi est-ce que $Pr(W|P) = Pr(W|do(P))$?
- ii) Pourquoi est-ce que $Pr(H|A) \neq Pr(H|do(A))$?

2 Partie 2 (20 points)

L'objectif de la partie 2 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 objectif 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 Théorie: 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 $\mathbf{y} = [0, 1, 0, \dots, 0]^T$ représente 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 : $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]^T$ et $\mathbf{b} = [b_1, b_2, \dots, b_K]^T$ et la fonction softmax comme fonction de sortie, on peut exprimer notre modèle sous la forme :

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

L'ensemble de données consiste de n paires (label, input) de la forme $\mathcal{D} := (\tilde{\mathbf{y}}_i, \tilde{\mathbf{x}}_i)_{i=1}^n$, où nous utilisons l'astuce de redéfinir $\tilde{\mathbf{x}}_i = [\tilde{\mathbf{x}}_i^T 1]^T$ et nous redéfinissons 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}). Notre fonction de perte, la log-vraisemblance négative des données selon notre modèle est définie comme:

$$\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}) := -\log \prod_{i=1}^N P(\tilde{\mathbf{y}}_i | \tilde{\mathbf{x}}_i; \boldsymbol{\theta}) \quad (2)$$

Pour cette partie du TP, nous avons calculé pour vous le gradient de la fonction de perte par rapport aux paramètres du modèle:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}) = -\sum_{i=1}^N \frac{\partial}{\partial \boldsymbol{\theta}} \left\{ \log \left(\frac{\exp(\tilde{\mathbf{y}}_i^T \boldsymbol{\theta} \tilde{\mathbf{x}}_i)}{\sum_{\mathbf{y}_k \in \mathcal{Y}} \exp(\mathbf{y}_k^T \boldsymbol{\theta} \tilde{\mathbf{x}}_i)} \right) \right\} \quad (3)$$

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

$$= \sum_{i=1}^N \hat{\mathbf{p}}_i \tilde{\mathbf{x}}_i^T - \sum_{i=1}^N \tilde{\mathbf{y}}_i \tilde{\mathbf{x}}_i^T \quad (5)$$

où $\hat{\mathbf{p}}_i$ est un 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.

Finalement, il reste à discuter de l'évaluation du modèle. Pour la tâche d'intérêt, qui est une instance du problème de classification, il existe plusieurs métriques pour mesurer les performances du modèle la précision de classification, l'erreur de classification, le taux de faux/vrai positifs/négatifs, etc. Habituellement dans le contexte de l'apprentissage automatique, la précision est la plus commune.

La précision est définie comme le rapport du nombre d'échantillons bien classés sur le nombre total d'échantillons à classer:

$$\tau_{acc} := \frac{|\mathcal{C}|}{|\mathcal{D}|}$$

où l'ensemble des échantillons bien classés \mathcal{C} est:

$$\mathcal{C} := \{(\mathbf{x}, \mathbf{y}) \in \mathcal{D} \mid \arg \max_k P(\cdot | \tilde{\mathbf{x}}_i; \boldsymbol{\theta})_k = \arg \max_k \tilde{y}_{i,k}\}$$

En mots, il s'agit du sous-ensemble d'échantillons pour lesquels la classe la plus probable selon notre modèle correspond à la vraie classe.

2.2 Description des tâches

2.2.1 Code à compléter

On vous demande de compléter l'extrait de code ci-dessous pour résoudre ce problème. Vous devez utiliser la librairie PyTorch cette partie du TP: <https://pytorch.org/docs/stable/index.html>. Mettez à jour les paramètres de votre modèle avec la descente par *mini-batch*. Exécutez des expériences avec trois différents ensembles: un ensemble d'apprentissages avec 90% des exemples (choisis au hasard), un ensemble de validation avec 10%. Utilisez uniquement l'ensemble de test pour obtenir votre meilleur résultat une fois que vous pensez avoir obtenu votre meilleure stratégie pour entraîner le modèle.

2.2.2 Rapport à rédiger

Présentez vos résultats dans un rapport. Ce rapport devrait inclure:

- **Recherche d'hyperparamètres:** Faites une recherche d'hyperparamètres 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 pour des modèles entraînés avec SGD. Présentez dans un tableau la précision finale du modèle, sur l'*ensemble de validation*, pour ces différentes combinaisons d'hyperparamètres.
- **Analyse du meilleur modèle:** Pour votre meilleur modèle, présentez deux figures montrant la progression de son apprentissage sur l'*ensemble d'entraînement et l'ensemble de validation*. La première figure montrant les courbes de log-vraisemblance négative moyenne après chaque epoch, la deuxième montrant la précision du modèle après chaque epoch. Finalement donnez la précision finale sur l'ensemble de test.
- Lire l'article de recherche - Adam: a method for stochastic optimization. Kingma, D., & Ba, J. (2015). International Conference on Learning Representation (ICLR). <https://arxiv.org/pdf/1412.6980.pdf>. Implémentez Adam, répétez les deux étapes précédentes (recherche d'hyperparamètres et analyse du meilleur modèle) cette fois en utilisant Adam, et comparez les performances finales avec votre meilleur modèle SGD.

IMPORTANT

L'objectif du TP est de vous faire implémenter la rétropropagation à la main. **Il est donc interdit d'utiliser les capacités de construction de modèles ou de différentiation automatique de pytorch – par exemple, aucun appels à `torch.nn`, `torch.autograd` ou à la méthode `.backward()`.** L'objectif est d'implémenter un modèle de classification logistique ainsi que son entraînement en utilisant uniquement des opérations matricielles de base fournies par PyTorch e.g. `torch.sum()`, `torch.matmul()`, etc.

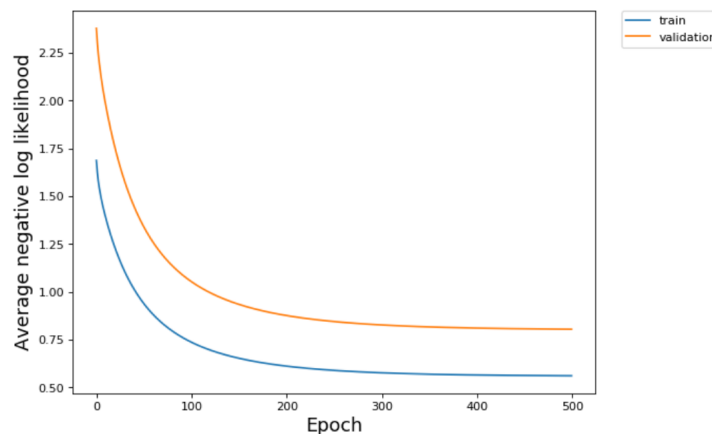


Figure 2: Exemple de courbes d'apprentissage.

3 Partie III (20 points)

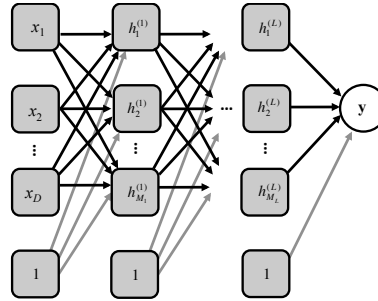


Figure 3: Un réseau de neurones.

Pour cette partie, vous pouvez travailler en groupes de 2, mais il faut écrire sa propre dérivation et soumettre son propre rapport. Si vous travaillez avec un partenaire, il faut indiquer leur nom dans votre rapport.

Considérons maintenant un réseau de neurones avec une couche d'entrée avec $D = 784$ unités, L couches cachées, chacune avec 300 unités et un vecteur de sortie \mathbf{y} de dimension K . Vous avez $i = 1, \dots, N$ exemples dans un ensemble d'apprentissage, où chaque $\mathbf{x}_i \in \mathbb{R}^{784}$ est un vecteur de caractéristiques (features). \mathbf{y} est un vecteur du type *one-hot* – un vecteur de zéros avec un seul 1 pour indiquer que la classe $C = k$ dans la dimension k . Par exemple, le vecteur $\mathbf{y} = [0, 1, 0, \dots, 0]^T$ représente la deuxième classe. La fonction de perte est donnée par

$$\mathcal{L} = - \sum_{i=1}^N \sum_{k=1}^K y_{k,i} \log(f_k(\mathbf{x}_i)) \quad (6)$$

La fonction d'activation de la couche finale a la forme $\mathbf{f} = [f_1, \dots, f_K]$ donné par la fonction d'activation softmax:

$$f_k(\mathbf{a}^{(L+1)}(\mathbf{x}_i)) = \frac{\exp(a_k^{(L+1)})}{\sum_{c=1}^K \exp(a_c^{(L+1)})},$$

et les couches cachées utilisent une fonction d'activation de type ReLU:

$$\mathbf{h}^{(l)}(\mathbf{a}^{(l)}(\mathbf{x}_i)) = \text{ReLU}(\mathbf{a}^{(l)}(\mathbf{x}_i)) = \max(0, \mathbf{a}^{(l)}(\mathbf{x}_i)) \quad (7)$$

où $\mathbf{a}^{(l)}$ est le vecteur résultant du calcul de la préactivation habituelle $\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$, qui pourrait être simplifiée à $\boldsymbol{\theta}^{(l)}\tilde{\mathbf{h}}^{(l-1)}$ en utilisant l'astuce de définir $\tilde{\mathbf{h}}$ comme \mathbf{h} avec un 1 concaténé à la fin du vecteur.

a) (10 points) Donnez le pseudocode incluant des *calculs matriciels—vectoriels*¹ détaillés pour l'algorithme de rétropropagation pour calculer le gradient pour les paramètres de chaque couche **étant donné un exemple d'entraînement**.

b) (10 points) Implémentez l'optimisation basée sur le gradient de ce réseau en Pytorch. Utilisez le code squelette ci-dessous comme point de départ et implémentez les mathématiques de l'algorithme de rétropropagation que vous avez décrit à la question précédente. Utilisez encore l'ensemble de données de Fashion MNIST (voir Partie 2). ****Comparez différents modèles ayant différentes largeurs (nombre d'unités) et profondeurs (nombre de couches)****. Ici encore, n'utilisez l'ensemble de test que pour votre expérience finale lorsque vous pensez avoir obtenu votre meilleur modèle.

¹Voir The Matrix Cookbook: <https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf>