INF8225 TP1 H25 (v2.0)

Prénom - NOM / Matricule : Mohamed-Ali - LAJNEF /2404991

Date limite :

20h30 le 6 février 2025 (Partie 1 et 2)

20h30 le 20 février 2025 (Partie 3)

Remettez votre fichier Colab sur Moodle en 2 formats: **.pdf** ET **.ipynb**

Comment utiliser:

Il faut copier ce notebook dans vos dossiers pour avoir une version que vous pouvez modifier, voici deux façons de le faire:

- File / Save a copy in Drive ...
- File / Download .ipynb

Pour utiliser un GPU

Runtime / Change Runtime Type / Hardware Accelerator / GPU

Partie 1 (16 points)

Objectif

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.

Problème

Considérons le réseau Bayésien ci-dessous.

Ceci représente un modèle simple pour les notes à un examen (G) et sa relation avec les étudiants qui se préparent aux examens et font correctement le travail pour les devoirs (S), les étudiants qui ont des difficultés dans la vie juste avant l'examen final (D), les étudiants qui réussissent bien à un entretien technique pour un emploi axé sur le sujet du cours (R), et des étudiants qui se retrouvent sur une sorte de palmarès de leur programme (L).

Trucs et astuces

Nous utiliserons des vecteurs multidimensionnels `5d-arrays` dont les `axes` représentent:

```
axe 0 : Se préparer (S)
axe 1 : Difficultés avant l'exam (D)
axe 2 : Réussir l'entretien technique (R)
axe 3 : Note dans le cours (Grade) (G)
axe 4 : Liste d'honneur (L)
```

Chaque `axe` serait de dimension `2` ou `3`:

```
Exemple pour S:
0 : s0
1 : s1

Exemple pour G:
0 : g0
1 : g1
2 : g2
```

Quelques point à garder en tête:

- Utiliser la jointe comme point de départ pour vos calculs (ne pas développer tous les termes à la main).
- Attention à l'effet du do-operator sur le graphe.
- L'argument "keepdims=True" de "np.sum()" vous permet conserver les mêmes indices.
- Pour un rappel sur les probabilités conditionelles, voir: https://www.probabilitycourse.com/chapter1/1_4_0_conditional_probability.php

1. Complétez les tables de probabilités ci-dessous

```
import numpy as np
np.set_printoptions(precision=5)

# Les tableaux sont bâtis avec les dimensions (S, D, R, G, L)
# et chaque dimension avec les probablités associées aux 2 ou 3
valeurs possibles ({0, 1} ou {0, 1, 2})

Pr_S = np.array([0.2, 0.8]).reshape(2, 1, 1, 1, 1) # Donné en exemple
Pr_D = np.array([0.9, 0.1]).reshape(1, 2, 1, 1, 1) # TODO
Pr_R_given_S = np.array([[0.9, 0.1] , [0.2 , 0.8]]).reshape(2, 1, 2,
1, 1) # TODO
Pr_G_given_SD = np.array([[[0.5 , 0.3 , 0.2] , [0.9 , 0.08 , 0.02]] ,
[[0.1 , 0.2 , 0.7] , [0.3 , 0.4 , 0.3]]]).reshape(2, 2, 1, 3, 1) #
TODO
Pr_L_given_G =  np.array([[0.9, 0.1] , [0.6 , 0.4] , [0.01 ,
0.99]] ).reshape(1, 1, 1, 3, 2)  # TODO

print (f"Pr(S)=\n{np.squeeze(Pr_S)}\n")
print (f"Pr(D)=\n{np.squeeze(Pr_D)}\n")
print (f"Pr(R|S)=\n{np.squeeze(Pr_R_given_S)}\n")
```

```
print (f"Pr(G|S,D)=\n{np.squeeze(Pr_G_given_SD)}\n")
print (f"Pr(L|G)=\n{np.squeeze(Pr_L_given_G)}\n")
```

```
Pr(S)=
[0.2 0.8]

Pr(D)=
[0.9 0.1]

Pr(R|S)=
[[0.9 0.1]
 [0.2 0.8]]

Pr(G|S,D)=
[[[0.5  0.3  0.2 ]
  [0.9  0.08 0.02]]

 [[0.1  0.2  0.7 ]
  [0.3  0.4  0.3 ]]]

Pr(L|G)=
[[0.9  0.1 ]
 [0.6  0.4 ]
 [0.01 0.99]]
```

2. À l'aide de ces tables de probabilité conditionnelles, calculez les requêtes ci-dessous. Dans les cas où l'on compare un calcul non interventionnel à un calcul interventionnel, commentez sur l'interprétation physique des deux situations et les résultats obtenus à partir de vos modèles.

a) $Pr(G)=\left[P(G=g^0),P(G=g^1),P(G=g^2)\right)$

```
joint  = Pr_G_given_SD * Pr_S * Pr_D* Pr_R_given_S * Pr_L_given_G
answer_a = np.sum(joint, axis=(0, 1, 2 , 4)) # TODO
print(f"Pr(G)={answer_a}")
```

```
Pr(G)=[0.204  0.2316 0.5644]

# Ce texte est au format code
```

b) $Pr\left(G \vee R=r^1\right)$

```
Pr_GR = np.sum(joint[: , : , 1:2 ,: ,:] , axis=(0,1,4))
Pr_R = np.sum(joint[: , : , 1:2 ,: ,:] , axis=(0,1,3,4))
answer_b = (Pr_GR / Pr_R).squeeze()
print(f"Pr(G|R=r1)={answer_b}")
```

```
Pr(G|R=r1)=[0.13273 0.22176 0.64552]
```

c) $Pr(G \vee R=r^0)$

```
Pr_GR = np.sum(joint[: , : , 0:1 ,: ,:] , axis=(0,1,4))
Pr_R = np.sum(joint[: , : , 0:1 ,: ,:] , axis=(0,1,2,3,4))

answer_c = (Pr_GR / Pr_R).squeeze()
print(f"Pr(G|R=r0)={answer_c}")
```

```
Pr(G|R=r0)=[0.34235 0.25071 0.40694]
```

d) $Pr(G \vee R=r^1, S=s^0)$

```
Pr_GRS = np.sum(joint[0:1,:,1:2,:,:] , axis=(0,1,2,4))
Pr_RS = np.sum(joint[0:1 , : , 1:2 ,: ,:] , axis=(0,1,2,3,4))
answer_d =  (Pr_GRS / Pr_RS).squeeze()
print(f"Pr(G|R=r1, S=s0)={answer_d}")
```

```
Pr(G|R=r1, S=s0)=[0.54  0.278 0.182]
```

e) $Pr(G \vee R=r^0, S=s^0)$

```
Pr_GRS = np.sum(joint[0:1,:,0:1,:,:] , axis=(0,1,2,4))
Pr_RS = np.sum(joint[0:1 , : , 1:2 ,: ,:] , axis=(0,1,2,3,4))

answer_e = (Pr_GRS / Pr_RS).squeeze()  # TODO
print(f"Pr(G|R=r0, S=s0)={answer_e}")
```

```
Pr(G|R=r0, S=s0)=[4.86  2.502 1.638]
```

f) $Pr(R \vee D=d^1)$

```
Pr_RD =  np.sum(joint[:,1:2,:,:,:] , axis=(0,1,3,4))
Pr_D1 =  np.sum(joint[:,1:2,:,: ,:] , axis=(0,1,2,3,4))
answer_f = (Pr_RD / Pr_D1).squeeze()

print(f"Pr(R|D=d1)={answer_f}")
```

```
Pr(R|D=d1)=[0.34 0.66]
```

g) $Pr(R \vee D=d^0)$

```
Pr_RD =  np.sum(joint[:,0:1,:,:,:] , axis=(0,1,3,4))
Pr_D1 =  np.sum(joint[:,0:1,:,: ,:] , axis=(0,1,2,3,4))
answer_g = (Pr_RD / Pr_D1).squeeze()   # TODO
print(f"Pr(R|D=d0)={answer_g}")
```

```
Pr(R|D=d0)=[0.34 0.66]
```

h) $Pr(R \vee D=d^1, G=g^2)$

```
Pr_RDG =  np.sum(joint[:,1:2,:,2:3,:] , axis=(0,1,3,4))
Pr_DG =  np.sum(joint[:,1:2,:,2:3 ,:] , axis=(0,1,2,3,4))
answer_h =  (Pr_RDG / Pr_DG).squeeze()   # TODO
print(f"Pr(R|D=d1, G=g2)={answer_h}")
```

```
Pr(R|D=d1, G=g2)=[0.21148 0.78852]
```

i) $Pr(R \vee D=d^0, G=g^2)$

```
Pr_RDG =  np.sum(joint[:,0:1,:,2:3,:] , axis=(0,1,3,4))
Pr_DG =  np.sum(joint[:,0:1,:,2:3 ,:] , axis=(0,1,2,3,4))
answer_i =  (Pr_RDG / Pr_DG).squeeze()   # TODO
print(f"Pr(R|D=d0, G=g2)={answer_i}")
```

```
Pr(R|D=d0, G=g2)=[0.24667 0.75333]
```

j) $Pr(R \vee D=d^1, L=l^1)$

```
Pr_RDL =  np.sum(joint[:,1:2,:,:, 1:2] , axis=(0,1,3,4))
Pr_DL =  np.sum(joint[:,1:2,:,: ,1:2] , axis=(0,1,2,3,4))
answer_j =  (Pr_RDL / Pr_DL).squeeze() # TODO
print(f"Pr(R|D=d1, L=l1)={answer_j}")
```

```
Pr(R|D=d1, L=l1)=[0.2475 0.7525]
```

k) $Pr(R \vee D=d^0, L=l^1)$

```
Pr_RDL =  np.sum(joint[:,0:1,:,:, 1:2] , axis=(0,1,3,4))
Pr_DL =  np.sum(joint[:,0:1,:,: ,1:2] , axis=(0,1,2,3,4))
answer_k =  (Pr_RDL / Pr_DL).squeeze() # TODO
print(f"Pr(R|D=d1, L=l1)={answer_k}")
```

```
Pr(R|D=d1, L=l1)=[0.2736 0.7264]
```

l) $Pr(R \vee do(G=g^2))$

```
joint_2 = Pr_L_given_G[:,:,:,2:3 ,:] * Pr_R_given_S * Pr_S * Pr_D
answer_l =  np.sum(joint_2[:,:,:,:,:] , axis=(0,1,3,4)) # TODO
print(f"Pr(R|do(G=g2))={answer_l}")

Pr(R|do(G=g2))=[0.34 0.66]
```

m) $Pr(R \vee G = g^2)$

```
Pr_RG =  np.sum(joint[:,:,:,2:3,:] , axis=(0,1,3,4))
Pr_G2 =  np.sum(joint[:,:,:,2:3 ,:] , axis=(0,1,2,3,4))
answer_m = (Pr_RG / Pr_G2).squeeze() # TODO
print(f"Pr(R|G=g2)={answer_m}")

Pr(R|G=g2)=[0.24515 0.75485]
```

n) $Pr(R)$

```
answer_n = np.sum(Pr_R_given_S * Pr_S, axis=(0,1,3 ,4)) # TODO
print(f"Pr(R)={answer_n}")

Pr(R)=[0.34 0.66]
```

o) $Pr(G \vee do(L = l^1))$

```
joint_3 = Pr_G_given_SD * Pr_S * Pr_D * Pr_R_given_S
answer_o =  np.sum( joint_3, axis=(0,1,2,4))# TODO
print(f"Pr(G|do(L=l1))={answer_o}")

Pr(G|do(L=l1))=[0.204  0.2316 0.5644]
```

p) $Pr(G = 1 \vee L = l^1)$

```
Pr_GL =  np.sum(joint[:,:,:,:,1:2] , axis=(0,1,2,4))
Pr_L1 =  np.sum(joint[:,:,:,:,1:2] , axis=(0,1,2,3,4))

answer_p = (Pr_GL / Pr_L1).squeeze()[1]  # TODO
print(f"Pr(G=1|L=l1)={answer_p}")

Pr(G=1|L=l1)=0.13789900505510602
```

Réponse:

Comparaison entre $Pr(R \vee do(G = g^2))$ et $Pr(R \vee G = g^2)$:

Nous observons que $Pr(R | do(G=g^2))= [0.34 , 0.66] = Pr(R) $, ce qui est cohérent étant donné que l'intervention coupe le lien entre (S) et (G). En forçant l'obtention de bonnes notes, cela n'a pas d'effet direct sur la réussite de l'entretien technique, car (S) n'est plus pris en compte dans la relation causale. En revanche, lorsque nous **observons** que l'on a obtenu de bonnes notes

$G=g^2$ cela nous apporte de l'information sur la variable (S), qui influence également (R). Ainsi, la connaissance d'une bonne note augmente la probabilité de réussir l'entretien technique, ce qui explique pourquoi $Pr\left(R \vee G=g^2\right)=[0.24515, 0.75485)$ est différent de $Pr\left(R \vee do\left(G=g^2\right)\right)$.

Comparaison entre $Pr\left(G \vee do\left(L=l^1\right)\right)$ et $Pr\left(G=1 \vee L=l^1\right)$:

Nous observons que $ Pr(G | do(L=l^1)) = [0.204, 0.2316, 0.5644] = Pr(G) $, ce qui est cohérent étant donné que l'intervention coupe le lien entre (G) et (L). En revanche, la probabilité conditionnelle $ Pr(G=1 | L=l^1) = 0.1379$ est plus faible, car lorsqu'on observe $\left(L=l^1\right)$ sans intervenir, cela fournit de l'information sur (G). Ainsi, observer que quelqu'un est sur la liste d'honneur nous informe sur ses résultats scolaires et les facteurs sous-jacents, tandis qu'une intervention directe sur (L) ne modiifie pas les probabilités de base d'obtenir de bonnes notes.

Partie 2 (20 points)

Objectif

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.

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 $y=[0, 1, 0, \cdots, 0)^T$ représente la deuxième classe. Les caractéristiques (features) sont données par des vecteurs $x_i \in R^D$. En définissant les paramètres de notre modèle comme : $W=\left[w_1, \cdots, w_K\right)^T$ et $b=\left[b_1, b_2, \cdots b_K\right)^T$ et la fonction softmax comme fonction de sortie, on peut exprimer notre modèle sous la forme : \ begin{eqnarray} p(\mathbf{y}|\mathbf{x}) &=& \frac{\exp(\mathbf{y}^T \mathbf{W} \mathbf{x} + \mathbf{y}^T \mathbf{b})}{\sum_{\mathbf{y}_k \in \mathscr{Y}} \exp(\mathbf{y}_k^T \ mathbf{W} \mathbf{x} + \mathbf{y}*k^T |mathbf{b})} |end{eqnarray} L'ensemble de données consiste de n paires (label, input) de la forme $D:=\left(\tilde{y}_i, \tilde{x}_i\right)_{i=1}^n$, où nous utilisons l'astuce de redéfinir $\tilde{x}_i=\left[\tilde{x}_i^T 1\right)^T$ et nous redéfinissions la matrice de paramètres $\theta \in R^{K \times(D+1)}$ (voir des notes de cours pour la relation entre θ et W). Notre fonction de perte, la log-vraisemblance négative

des données selon notre modèle est définie comme: |begin{equation} |mathscr{L}|big(|boldsymbol{|theta}, |mathscr{D} |big) := -|log |prod{i=1}^N P(\mathbf{\tilde{y}}_i|\mathbf{\tilde{x}}*i; |boldsymbol{|theta}) |end{equation} Pour cette partie du TP, nous avons calculé pour vous le gradient de la fonction de perte par rapport par rapport aux paramètres du modèle: |begin{eqnarray} |frac{|partial}{|partial |boldsymbol{|theta}} |mathscr{L}|big(|boldsymbol{|theta}, |mathscr{D} |big) &=& -|sum*{i=1}^N \frac{\partial}{\partial \boldsymbol{\theta}} \Bigg{\log \Bigg(\frac{\exp(\mathbf{\tilde{y}}_i^T \boldsymbol{\theta} \mathbf{\tilde{x}}*i)}{|sum*{\mathbf{y}_k \in \mathscr{Y}} \exp(\mathbf{y}_k^T \boldsymbol{\theta} \mathbf{\tilde{x}}*i)} |Bigg) |Bigg} | &=& -|sum*{i=1}^N \left(\mathbf{\tilde{y}}*i |mathbf{|tilde{x}}^T_i- |sum*{\mathbf{y}_k \in \mathscr{Y}} P(\mathbf{y}_k|\mathbf{\tilde{x}}_i,\boldsymbol{\theta}) \mathbf{y}*k |mathbf{|tilde{x}}^T_i |right) | &=& |sum*{i=1}^N \mathbf{\hat{p}}*i |mathbf{|tilde{x}}^T_i - |sum*{i=1}^N \mathbf{\tilde{y}}_i \mathbf{\tilde{x}}^T_i \end{eqnarray} où $\hat{p}_i$ est un vecteur de probabilités produit par le modèle pour l'exemple $\tilde{x}_i$ et $\tilde{y}_i$ 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:

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

où l'ensemble des échantillons bien classés C est:

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

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.

Description des tâches

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. 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 entrainés 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'*ensembe d'entrainement 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 utilisat 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 entainement en utilisant uniquement des opérations matricielles de base fournies par PyTorch e.g. torch.sum(), torch.matmul(), etc.

Fonctions fournies

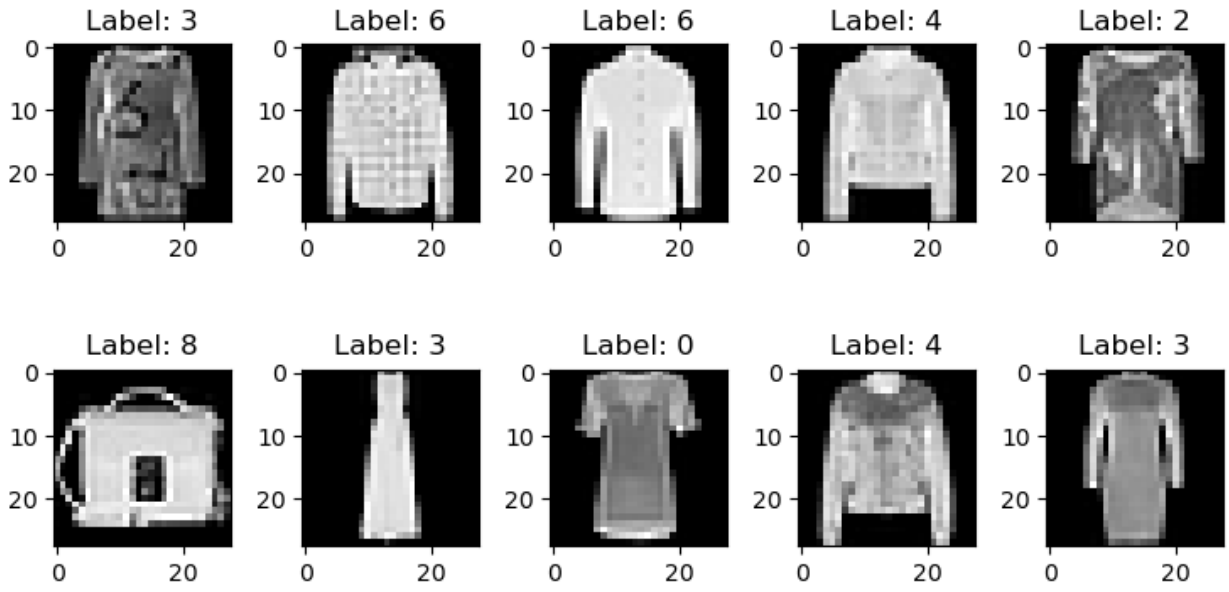
```
# fonctions pour charger les ensembles de donnees
from torchvision.datasets import FashionMNIST
from torchvision import transforms
import torch
from torch.utils.data import DataLoader, random_split
from tqdm import tqdm
import matplotlib.pyplot as plt

def get_fashion_mnist_dataloaders(val_percentage=0.1, batch_size=1):
  dataset = FashionMNIST("./dataset", train=True,  download=True,
transform=transforms.Compose([transforms.ToTensor()]))
  dataset_test = FashionMNIST("./dataset", train=False,
download=True, transform=transforms.Compose([transforms.ToTensor()]))
  len_train = int(len(dataset) * (1.-val_percentage))
  len_val = len(dataset) - len_train
  dataset_train, dataset_val = random_split(dataset, [len_train,
len_val])
  data_loader_train = DataLoader(dataset_train,
```

```
batch_size=batch_size,shuffle=True,num_workers=4)
  data_loader_val   = DataLoader(dataset_val,
batch_size=batch_size,shuffle=True,num_workers=4)
  data_loader_test  = DataLoader(dataset_test,
batch_size=batch_size,shuffle=True,num_workers=4)
  return data_loader_train, data_loader_val, data_loader_test

def reshape_input(x, y):
    x = x.view(-1, 784)
    y = torch.FloatTensor(len(y), 10).zero_().scatter_(1,y.view(-
1,1),1)
    return x, y


# call this once first to download the datasets
_ = get_fashion_mnist_dataloaders()

# simple logger to track progress during training
class Logger:
    def __init__(self):
        self.losses_train = []
        self.losses_valid = []
        self.accuracies_train = []
        self.accuracies_valid = []

    def log(self, accuracy_train=0, loss_train=0, accuracy_valid=0,
loss_valid=0):
        self.losses_train.append(loss_train)
        self.accuracies_train.append(accuracy_train)
        self.losses_valid.append(loss_valid)
        self.accuracies_valid.append(accuracy_valid)

    def plot_loss_and_accuracy(self, train=True, valid=True):

        assert train and valid, "Cannot plot accuracy because neither
train nor valid."

        figure, (ax1, ax2) = plt.subplots(nrows=1, ncols=2,
                                          figsize=(12, 6))

        if train:
            ax1.plot(self.losses_train, label="Training")
            ax2.plot(self.accuracies_train, label="Training")
        if valid:
            ax1.plot(self.losses_valid, label="Validation")
            ax1.set_title("CrossEntropy Loss")
            ax2.plot(self.accuracies_valid, label="Validation")
            ax2.set_title("Accuracy")

        for ax in figure.axes:
```

```
            ax.set_xlabel("Epoch")
            ax.legend(loc='best')
            ax.set_axisbelow(True)
            ax.minorticks_on()
            ax.grid(True, which="major", linestyle='-')
            ax.grid(True, which="minor", linestyle='--',
color='lightgrey', alpha=.4)

    def print_last(self):
        print(f"Epoch {len(self.losses_train):2d}, \
                Train:loss={self.losses_train[-1]:.3f},
accuracy={self.accuracies_train[-1]*100:.1f}%, \
                Valid: loss={self.losses_valid[-1]:.3f},
accuracy={self.losses_valid[-1]*100:.1f}%", flush=True)
```

Aperçu de l'ensemble de données FashionMnist

```
def plot_samples():
  a, _, _ = get_fashion_mnist_dataloaders()
  num_row = 2
  num_col = 5# plot images
  num_images = num_row * num_col
  fig, axes = plt.subplots(num_row, num_col,
figsize=(1.5*num_col,2*num_row))
  for i, (x,y) in enumerate(a):
      if i >= num_images:
        break
      ax = axes[i//num_col, i%num_col]
      x = (x.numpy().squeeze() * 255).astype(int)
      y = y.numpy()[0]
      ax.imshow(x, cmap='gray')
      ax.set_title(f"Label: {y}")

  plt.tight_layout()
  plt.show()
plot_samples()
```



Fonctions à compléter

```
def accuracy(y, y_pred) :
    # todo : nombre d'éléments à classifier.
    card_D = len(y)

    # todo : calcul du nombre d'éléments bien classifiés.
    card_C = torch.sum(torch.argmax(y_pred, dim=1) == torch.argmax(y,
dim=1)).item()

    # todo : calcul de la précision de classification.
    acc = card_C/card_D

    return acc, (card_C, card_D)

def accuracy_and_loss_whole_dataset(data_loader, model):
    cardinal = 0
    loss     = 0.
    n_accurate_preds  = 0.

    for x, y in data_loader:
        x, y = reshape_input(x, y)
        y_pred                = model.forward(x)
        xentrp                = cross_entropy(y, y_pred)
        _, (n_acc, n_samples) = accuracy(y, y_pred)

        cardinal = cardinal + n_samples
        loss = loss + xentrp
        n_accurate_preds  = n_accurate_preds + n_acc

    loss = loss / float(cardinal)
    acc  = n_accurate_preds / float(cardinal)
```

```
    return acc, loss

def cross_entropy(y, y_pred):
    # todo : calcul de la valeur d'entropie croisée.
    y_pred = torch.clamp(y_pred, min=1e-9, max=1.0)
    loss = -torch.sum(y * torch.log(y_pred))
    return loss.item()

def softmax(x, axis=-1):
    # assurez vous que la fonction est numeriquement stable
    # e.g. softmax(torch.tensor([[1000, 10000, 100000],]))

    # todo : calcul des valeurs de softmax(x)
    x_max = torch.max(x, dim=axis, keepdim=True)
    x = x - x_max.values
    exp_x = torch.exp(x)
    values = exp_x / torch.sum(exp_x, dim=axis, keepdim=True)

    return values

def inputs_tilde(x, axis=-1):
    # augments the inputs `x` with ones along `axis`
    # todo : implémenter code ici.
    ones_shape = list(x.shape)

    ones_shape[axis] = 1
    ones = torch.ones(ones_shape, dtype=x.dtype, device=x.device)

    # Concatenate the ones along the specified axis
    x_tilde = torch.cat([x, ones], dim=axis)

    return x_tilde

class LinearModel:
    def __init__(self, num_features, num_classes):
      self.params = torch.normal(0, 0.01, (num_features + 1,
num_classes))

      self.t = 0
      self.m_t = 0 # pour Adam: moyennes mobiles du gradient
      self.v_t = 0 # pour Adam: moyennes mobiles du carré du gradient

    def forward(self, x):
      # todo : implémenter calcul des outputs en fonction des inputs
`x`.
      inputs = inputs_tilde(x)
      outputs = softmax(torch.matmul(inputs, self.params))
      return outputs

    def get_grads(self, y, y_pred, X):
```

```
      # todo : implémenter calcul des gradients.
      grads = inputs_tilde(X).T @ (y_pred - y)
      return grads

    def sgd_update(self, lr, grads):
      # TODO : implémenter mise à jour des paramètres ici.
      self.params -= lr * grads

    def adam_update(self, lr, grads):
      # TODO : implémenter mise à jour des paramètres ici.
      B1 = 0.9
      B2 = 0.999
      eps = 1e-8
      self.t += 1
      self.m_t = 0.9 * self.m_t + 0.1 * grads
      self.v_t = 0.999 * self.v_t + 0.001 * torch.square(grads)
      m_hat = self.m_t / (1 - B1**self.t)
      v_hat = self.v_t / (1 - B2**self.t)
      self.params -= lr * m_hat / (torch.sqrt(v_hat) + eps)

def train(model, lr=0.1, nb_epochs=10, sgd=True,
data_loader_train=None, data_loader_val=None):
    best_model = None
    best_val_accuracy = 0
    logger = Logger()

    for epoch in range(nb_epochs+1):
        # at epoch 0 evaluate random initial model
        #   then for subsequent epochs, do optimize before evaluation.
        #print(f"epoch {epoch}")
        #print(data_loader_train)

        if epoch > 0:
          for x, y in data_loader_train:
              x, y = reshape_input(x, y)
              y_pred = model.forward(x)
              loss = cross_entropy(y, y_pred)
              grads = model.get_grads(y, y_pred, x)
              if sgd:
                model.sgd_update(lr, grads)
              else:
                model.adam_update(lr, grads)

        accuracy_train, loss_train =
accuracy_and_loss_whole_dataset(data_loader_train, model)
        accuracy_val, loss_val =
accuracy_and_loss_whole_dataset(data_loader_val, model)

        if accuracy_val > best_val_accuracy:
          best_val_accuracy = accuracy_val
```

```
            best_model = model
            #print("best model")   # TODO : record the best model
parameters and best validation accuracy

        logger.log(accuracy_train, loss_train, accuracy_val, loss_val)
        print(f"Epoch {epoch:2d}, \
                Train: loss={loss_train:.3f},
accuracy={accuracy_train*100:.1f}%, \
                Valid: loss={loss_val:.3f},
accuracy={accuracy_val*100:.1f}%", flush=True)

    return best_model, best_val_accuracy, logger

batch_size = 5
data_loader_train, data_loader_val, data_loader_test =
get_fashion_mnist_dataloaders(val_percentage=0.1,
batch_size=batch_size)
logger = Logger()
model = LinearModel(num_features=784, num_classes=10)
for x, y in data_loader_train:
    #print(x.shape)
    #print(y.shape)
    x, y = reshape_input(x, y)

    y_pred = model.forward(x)
    loss = cross_entropy(y, y_pred)
    grads = model.get_grads(y, y_pred, x)
    model.sgd_update(0.1, grads)

accuracy_train, loss_train =
accuracy_and_loss_whole_dataset(data_loader_train, model)
accuracy_val, loss_val =
accuracy_and_loss_whole_dataset(data_loader_val, model)
```

Évaluation

SGD: Recherche d'hyperparamètres

```
# SGD
# 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.
batch_size_list = [1,20,200,1000]   # Define ranges in a list
lr_list = [0.1,0.01,0.001]           # Define ranges in a list

with torch.no_grad():
  for lr in lr_list:
    for batch_size in batch_size_list:
```

```
print("----------------------------------------------------------------
---")
      print("Training model with a learning rate of {0} and a batch
size of {1}".format(lr, batch_size))
      data_loader_train, data_loader_val, data_loader_test =
get_fashion_mnist_dataloaders(val_percentage=0.1,
batch_size=batch_size)

      model = LinearModel(num_features=784, num_classes=10)
      _, val_accuracy, _ = train(model,lr=lr, nb_epochs=5, sgd=True,
data_loader_train=data_loader_train, data_loader_val=data_loader_val)
      print(f"validation accuracy = {val_accuracy*100:.3f}")
```

```
----------------------------------------------------------------
Training model with a learning rate of 0.1 and a batch size of 1
Epoch  0,                 Train: loss=2.317, accuracy=11.6%,
Valid: loss=2.316, accuracy=12.1%
Epoch  1,                 Train: loss=3.275, accuracy=73.8%,
Valid: loss=3.283, accuracy=72.9%
Epoch  2,                 Train: loss=2.547, accuracy=80.5%,
Valid: loss=2.656, accuracy=79.8%
Epoch  3,                 Train: loss=2.339, accuracy=80.2%,
Valid: loss=2.463, accuracy=79.3%
Epoch  4,                 Train: loss=1.732, accuracy=83.1%,
Valid: loss=1.935, accuracy=81.8%
Epoch  5,                 Train: loss=1.549, accuracy=85.3%,
Valid: loss=1.798, accuracy=83.4%
validation accuracy = 83.350
----------------------------------------------------------------
Training model with a learning rate of 0.1 and a batch size of 20
Epoch  0,                 Train: loss=2.295, accuracy=11.4%,
Valid: loss=2.296, accuracy=11.5%
Epoch  1,                 Train: loss=4.008, accuracy=73.1%,
Valid: loss=3.963, accuracy=73.4%
Epoch  2,                 Train: loss=2.164, accuracy=83.4%,
Valid: loss=2.177, accuracy=83.5%
Epoch  3,                 Train: loss=2.491, accuracy=82.7%,
Valid: loss=2.536, accuracy=82.5%
Epoch  4,                 Train: loss=2.540, accuracy=81.2%,
Valid: loss=2.662, accuracy=80.8%
Epoch  5,                 Train: loss=3.717, accuracy=77.6%,
Valid: loss=3.702, accuracy=77.7%
validation accuracy = 83.533
----------------------------------------------------------------
Training model with a learning rate of 0.1 and a batch size of 200
Epoch  0,                 Train: loss=2.295, accuracy=11.5%,
Valid: loss=2.292, accuracy=11.8%
Epoch  1,                 Train: loss=4.281, accuracy=77.8%,
Valid: loss=4.125, accuracy=78.5%
Epoch  2,                 Train: loss=3.085, accuracy=83.3%,
```

```
Valid: loss=3.022, accuracy=83.3%
Epoch  3,                 Train: loss=5.594, accuracy=71.1%,
Valid: loss=5.451, accuracy=72.0%
Epoch  4,                 Train: loss=4.424, accuracy=76.7%,
Valid: loss=4.534, accuracy=76.3%
Epoch  5,                 Train: loss=4.623, accuracy=75.6%,
Valid: loss=4.653, accuracy=75.7%
validation accuracy = 83.333
------------------------------------------------------------------
Training model with a learning rate of 0.1 and a batch size of 1000
Epoch  0,                 Train: loss=2.334, accuracy=7.6%,
Valid: loss=2.334, accuracy=7.4%
Epoch  1,                 Train: loss=6.433, accuracy=68.3%,
Valid: loss=6.380, accuracy=68.6%
Epoch  2,                 Train: loss=6.932, accuracy=66.0%,
Valid: loss=7.077, accuracy=65.4%
Epoch  3,                 Train: loss=5.814, accuracy=71.4%,
Valid: loss=5.800, accuracy=71.6%
Epoch  4,                 Train: loss=4.878, accuracy=76.0%,
Valid: loss=5.009, accuracy=75.3%
Epoch  5,                 Train: loss=4.817, accuracy=76.3%,
Valid: loss=4.730, accuracy=76.7%
validation accuracy = 76.733
------------------------------------------------------------------
Training model with a learning rate of 0.01 and a batch size of 1
Epoch  0,                 Train: loss=2.310, accuracy=16.5%,
Valid: loss=2.309, accuracy=16.9%
Epoch  1,                 Train: loss=0.641, accuracy=79.4%,
Valid: loss=0.669, accuracy=79.0%
Epoch  2,                 Train: loss=0.431, accuracy=85.6%,
Valid: loss=0.462, accuracy=84.6%
Epoch  3,                 Train: loss=0.474, accuracy=83.9%,
Valid: loss=0.505, accuracy=82.8%
Epoch  4,                 Train: loss=0.446, accuracy=85.1%,
Valid: loss=0.491, accuracy=83.7%
Epoch  5,                 Train: loss=0.449, accuracy=85.5%,
Valid: loss=0.494, accuracy=83.9%
validation accuracy = 84.617
------------------------------------------------------------------
Training model with a learning rate of 0.01 and a batch size of 20
Epoch  0,                 Train: loss=2.326, accuracy=7.4%,
Valid: loss=2.325, accuracy=7.4%
Epoch  1,                 Train: loss=0.920, accuracy=77.3%,
Valid: loss=0.955, accuracy=77.2%
Epoch  2,                 Train: loss=0.591, accuracy=81.9%,
Valid: loss=0.633, accuracy=81.0%
Epoch  3,                 Train: loss=0.532, accuracy=84.6%,
Valid: loss=0.568, accuracy=84.4%
Epoch  4,                 Train: loss=0.564, accuracy=83.8%,
```

```
Valid: loss=0.604, accuracy=83.9%
Epoch  5,                 Train: loss=0.738, accuracy=81.2%,
Valid: loss=0.776, accuracy=81.2%
validation accuracy = 84.383
----------------------------------------------------------------
Training model with a learning rate of 0.01 and a batch size of 200
Epoch  0,                 Train: loss=2.304, accuracy=10.4%,
Valid: loss=2.303, accuracy=10.3%
Epoch  1,                 Train: loss=1.606, accuracy=80.3%,
Valid: loss=1.611, accuracy=80.2%
Epoch  2,                 Train: loss=1.828, accuracy=79.6%,
Valid: loss=1.896, accuracy=78.9%
Epoch  3,                 Train: loss=3.421, accuracy=71.0%,
Valid: loss=3.528, accuracy=70.0%
Epoch  4,                 Train: loss=2.036, accuracy=80.6%,
Valid: loss=2.147, accuracy=79.5%
Epoch  5,                 Train: loss=3.716, accuracy=70.0%,
Valid: loss=3.875, accuracy=68.8%
validation accuracy = 80.250
----------------------------------------------------------------
Training model with a learning rate of 0.01 and a batch size of 1000
Epoch  0,                 Train: loss=2.297, accuracy=6.5%,
Valid: loss=2.297, accuracy=6.2%
Epoch  1,                 Train: loss=6.145, accuracy=66.2%,
Valid: loss=5.996, accuracy=66.8%
Epoch  2,                 Train: loss=4.831, accuracy=72.9%,
Valid: loss=4.868, accuracy=72.6%
Epoch  3,                 Train: loss=3.515, accuracy=79.3%,
Valid: loss=3.526, accuracy=79.4%
Epoch  4,                 Train: loss=4.530, accuracy=74.6%,
Valid: loss=4.380, accuracy=75.3%
Epoch  5,                 Train: loss=4.670, accuracy=72.9%,
Valid: loss=4.668, accuracy=73.2%
validation accuracy = 79.417
----------------------------------------------------------------
Training model with a learning rate of 0.001 and a batch size of 1
Epoch  0,                 Train: loss=2.273, accuracy=14.0%,
Valid: loss=2.273, accuracy=14.2%
Epoch  1,                 Train: loss=0.528, accuracy=82.6%,
Valid: loss=0.527, accuracy=82.5%
Epoch  2,                 Train: loss=0.480, accuracy=83.9%,
Valid: loss=0.481, accuracy=83.7%
Epoch  3,                 Train: loss=0.467, accuracy=84.3%,
Valid: loss=0.467, accuracy=84.1%
Epoch  4,                 Train: loss=0.446, accuracy=85.0%,
Valid: loss=0.446, accuracy=85.0%
Epoch  5,                 Train: loss=0.438, accuracy=85.1%,
Valid: loss=0.442, accuracy=84.8%
validation accuracy = 85.017
```

```
------------------------------------------------------------------
Training model with a learning rate of 0.001 and a batch size of 20
Epoch  0,                 Train: loss=2.307, accuracy=11.0%,
Valid: loss=2.304, accuracy=11.7%
Epoch  1,                 Train: loss=0.526, accuracy=82.8%,
Valid: loss=0.530, accuracy=82.9%
Epoch  2,                 Train: loss=0.481, accuracy=84.0%,
Valid: loss=0.487, accuracy=84.1%
Epoch  3,                 Train: loss=0.468, accuracy=83.9%,
Valid: loss=0.475, accuracy=84.1%
Epoch  4,                 Train: loss=0.454, accuracy=84.3%,
Valid: loss=0.465, accuracy=84.5%
Epoch  5,                 Train: loss=0.437, accuracy=85.1%,
Valid: loss=0.448, accuracy=85.3%
validation accuracy = 85.283
------------------------------------------------------------------
Training model with a learning rate of 0.001 and a batch size of 200
Epoch  0,                 Train: loss=2.292, accuracy=13.9%,
Valid: loss=2.294, accuracy=13.6%
Epoch  1,                 Train: loss=0.678, accuracy=77.1%,
Valid: loss=0.673, accuracy=77.8%
Epoch  2,                 Train: loss=0.506, accuracy=83.0%,
Valid: loss=0.511, accuracy=82.5%
Epoch  3,                 Train: loss=0.543, accuracy=82.5%,
Valid: loss=0.544, accuracy=82.1%
Epoch  4,                 Train: loss=0.478, accuracy=84.0%,
Valid: loss=0.487, accuracy=83.4%
Epoch  5,                 Train: loss=0.453, accuracy=84.5%,
Valid: loss=0.464, accuracy=84.3%
validation accuracy = 84.317
------------------------------------------------------------------
Training model with a learning rate of 0.001 and a batch size of 1000
Epoch  0,                 Train: loss=2.316, accuracy=6.8%,
Valid: loss=2.316, accuracy=7.1%
Epoch  1,                 Train: loss=2.946, accuracy=68.8%,
Valid: loss=2.966, accuracy=68.5%
Epoch  2,                 Train: loss=1.266, accuracy=78.7%,
Valid: loss=1.309, accuracy=78.0%
Epoch  3,                 Train: loss=1.983, accuracy=75.6%,
Valid: loss=2.040, accuracy=74.9%
Epoch  4,                 Train: loss=1.370, accuracy=77.6%,
Valid: loss=1.395, accuracy=77.4%
Epoch  5,                 Train: loss=1.154, accuracy=81.1%,
Valid: loss=1.202, accuracy=80.8%
validation accuracy = 80.800
```

Tableau pour la précision sur l'ensemble de validation

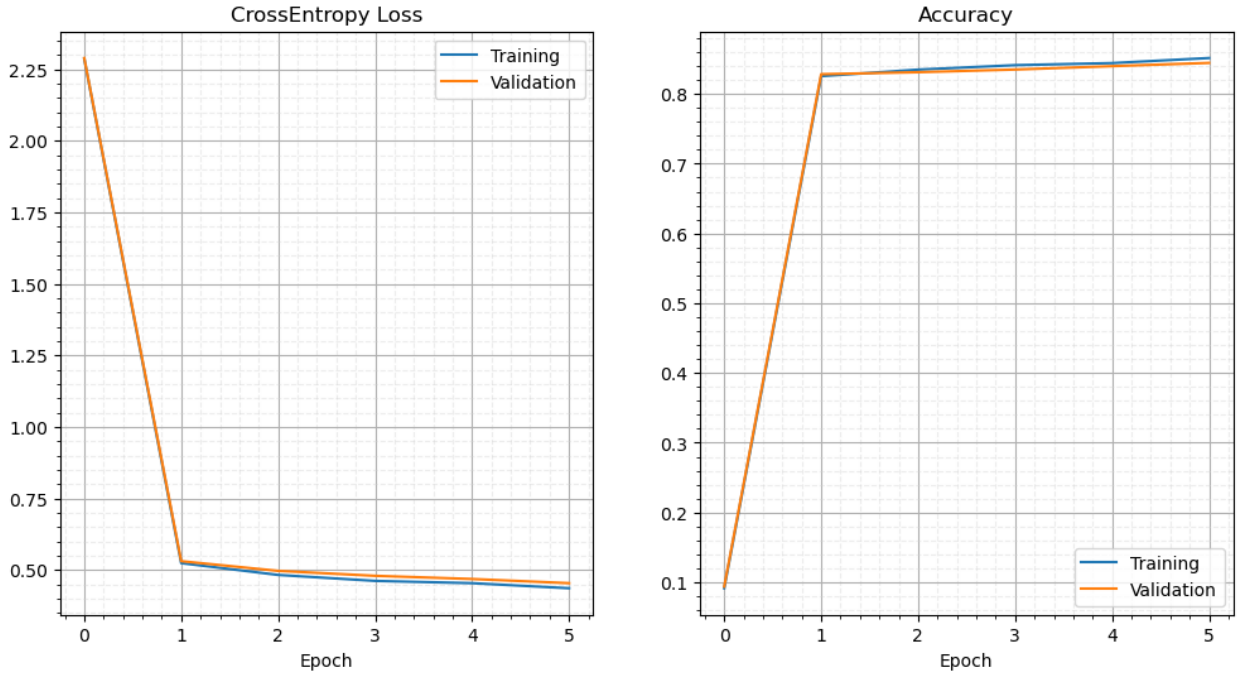
N.B. que les lignes correspondent aux valeurs du taux d'apprentisage et les colonnes correspondent au valeur du batch size. Les valeurs ci-dessous sont donné comme exemples; remplacez-les par les valeurs que vous avez utilisées pour votre recherche d'hyperparamètres.

learning rate\ batch_size	1	20	200	1000
0.1	83.350	83.533	83.333	76.733
0.01	84.617	84.383	80.250	79.417
0.001	85.017	85.283	84.317	80.800

SGD: Analyse du meilleur modèle

```
# SGD
# Montrez les résultats pour la meilleure configuration trouvez ci-
dessus.
batch_size = 1 # TODO: Vous devez modifier cette valeur avec la
meilleur que vous avez eu.
lr = 0.001         # TODO: Vous devez modifier cette valeur avec la
meilleur que vous avez eu.

with torch.no_grad():
  data_loader_train, data_loader_val, data_loader_test =
get_fashion_mnist_dataloaders(val_percentage=0.1,
batch_size=batch_size)

  model = LinearModel(num_features=784, num_classes=10)
  best_model, best_val_accuracy, logger = train(model,lr=lr,
nb_epochs=5, sgd=True,

data_loader_train=data_loader_train, data_loader_val=data_loader_val)
  logger.plot_loss_and_accuracy()
  print(f"Best validation accuracy = {best_val_accuracy*100:.3f}")

  accuracy_test, loss_test =
accuracy_and_loss_whole_dataset(data_loader_test, best_model)
print("Evaluation of the best training model over test set")
print("------")
print(f"Loss : {loss_test:.3f}")
print(f"Accuracy : {accuracy_test*100.:.3f}")

Epoch  0,                 Train: loss=2.287, accuracy=9.2%,
Valid: loss=2.289, accuracy=9.4%
Epoch  1,                 Train: loss=0.524, accuracy=82.5%,
Valid: loss=0.531, accuracy=82.8%
Epoch  2,                 Train: loss=0.483, accuracy=83.5%,
Valid: loss=0.497, accuracy=83.1%
Epoch  3,                 Train: loss=0.462, accuracy=84.1%,
Valid: loss=0.480, accuracy=83.5%
```

```
Epoch  4,                  Train: loss=0.454, accuracy=84.4%,
Valid: loss=0.469, accuracy=84.0%
Epoch  5,                  Train: loss=0.436, accuracy=85.2%,
Valid: loss=0.454, accuracy=84.5%
Best validation accuracy = 84.450
Evaluation of the best training model over test set
------
Loss : 0.474
Accuracy : 83.420
```



Adam: Recherche d'hyperparamètres

Implémentez Adam, répétez les deux étapes précédentes (recherche d'hyperparamètres et analyse du meilleur modèle) cette fois en utilisat Adam, et comparez les performances finales avec votre meilleur modèle SGD.

```
# ADAM
# 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.
batch_size_list = [1 , 20 , 200 , 1000]   # Define ranges in a list
lr_list = [0.1 , 0.01 , 0.001]           # Define ranges in a list

with torch.no_grad():
  for lr in lr_list:
    for batch_size in batch_size_list:
```

```
print("----------------------------------------------------------------
---")
      print("Training model with a learning rate of {0} and a batch
size of {1}".format(lr, batch_size))
      data_loader_train, data_loader_val, data_loader_test =
get_fashion_mnist_dataloaders(val_percentage=0.1,
batch_size=batch_size)

      model = LinearModel(num_features=784, num_classes=10)
      _, val_accuracy, _ = train(model,lr=lr, nb_epochs=5, sgd=False,
data_loader_train=data_loader_train, data_loader_val=data_loader_val)
      print(f"validation accuracy = {val_accuracy*100:.3f}")
```

```
----------------------------------------------------------------
Training model with a learning rate of 0.1 and a batch size of 1
Epoch  0,                  Train: loss=2.339, accuracy=4.2%,
Valid: loss=2.338, accuracy=4.1%
Epoch  1,                  Train: loss=3.311, accuracy=82.7%,
Valid: loss=3.432, accuracy=82.2%
Epoch  2,                  Train: loss=4.286, accuracy=77.6%,
Valid: loss=4.430, accuracy=76.9%
Epoch  3,                  Train: loss=4.533, accuracy=76.7%,
Valid: loss=4.716, accuracy=76.0%
Epoch  4,                  Train: loss=3.532, accuracy=81.6%,
Valid: loss=3.891, accuracy=80.1%
Epoch  5,                  Train: loss=3.797, accuracy=80.2%,
Valid: loss=3.975, accuracy=79.5%
validation accuracy = 82.183
----------------------------------------------------------------
Training model with a learning rate of 0.1 and a batch size of 20
Epoch  0,                  Train: loss=2.315, accuracy=4.0%,
Valid: loss=2.314, accuracy=4.4%
Epoch  1,                  Train: loss=3.466, accuracy=76.7%,
Valid: loss=3.825, accuracy=75.0%
Epoch  2,                  Train: loss=2.652, accuracy=80.9%,
Valid: loss=2.827, accuracy=80.1%
Epoch  3,                  Train: loss=2.557, accuracy=82.8%,
Valid: loss=2.586, accuracy=82.7%
Epoch  4,                  Train: loss=3.281, accuracy=79.1%,
Valid: loss=3.552, accuracy=78.2%
Epoch  5,                  Train: loss=2.134, accuracy=84.2%,
Valid: loss=2.370, accuracy=83.2%
validation accuracy = 83.233
----------------------------------------------------------------
Training model with a learning rate of 0.1 and a batch size of 200
Epoch  0,                  Train: loss=2.311, accuracy=10.1%,
Valid: loss=2.309, accuracy=9.8%
Epoch  1,                  Train: loss=1.397, accuracy=73.8%,
Valid: loss=1.442, accuracy=74.1%
```

```
Epoch  2,                 Train: loss=0.957, accuracy=80.8%,
Valid: loss=0.997, accuracy=80.0%
Epoch  3,                 Train: loss=1.069, accuracy=79.0%,
Valid: loss=1.173, accuracy=79.6%
Epoch  4,                 Train: loss=0.949, accuracy=80.5%,
Valid: loss=1.048, accuracy=79.7%
Epoch  5,                 Train: loss=0.983, accuracy=81.6%,
Valid: loss=1.108, accuracy=80.8%
validation accuracy = 80.783
-----------------------------------------------------------------
Training model with a learning rate of 0.1 and a batch size of 1000
Epoch  0,                 Train: loss=2.302, accuracy=14.3%,
Valid: loss=2.307, accuracy=13.8%
Epoch  1,                 Train: loss=0.884, accuracy=81.5%,
Valid: loss=0.947, accuracy=80.7%
Epoch  2,                 Train: loss=0.588, accuracy=83.8%,
Valid: loss=0.652, accuracy=83.1%
Epoch  3,                 Train: loss=0.450, accuracy=85.7%,
Valid: loss=0.517, accuracy=84.6%
Epoch  4,                 Train: loss=0.496, accuracy=85.5%,
Valid: loss=0.568, accuracy=84.8%
Epoch  5,                 Train: loss=0.491, accuracy=83.4%,
Valid: loss=0.545, accuracy=82.2%
validation accuracy = 84.833
-----------------------------------------------------------------
Training model with a learning rate of 0.01 and a batch size of 1
Epoch  0,                 Train: loss=2.357, accuracy=5.8%,
Valid: loss=2.357, accuracy=5.9%
Epoch  1,                 Train: loss=3.649, accuracy=66.1%,
Valid: loss=3.789, accuracy=65.6%
Epoch  2,                 Train: loss=2.567, accuracy=76.3%,
Valid: loss=2.698, accuracy=75.7%
Epoch  3,                 Train: loss=1.737, accuracy=83.1%,
Valid: loss=1.909, accuracy=82.3%
Epoch  4,                 Train: loss=1.490, accuracy=84.0%,
Valid: loss=1.706, accuracy=82.9%
Epoch  5,                 Train: loss=1.801, accuracy=82.2%,
Valid: loss=2.045, accuracy=80.5%
validation accuracy = 82.900
-----------------------------------------------------------------
Training model with a learning rate of 0.01 and a batch size of 20
Epoch  0,                 Train: loss=2.328, accuracy=4.8%,
Valid: loss=2.327, accuracy=4.3%
Epoch  1,                 Train: loss=0.539, accuracy=83.5%,
Valid: loss=0.560, accuracy=83.4%
Epoch  2,                 Train: loss=0.479, accuracy=85.5%,
Valid: loss=0.539, accuracy=84.4%
Epoch  3,                 Train: loss=0.560, accuracy=84.3%,
Valid: loss=0.636, accuracy=83.2%
```

```
Epoch  4,                  Train: loss=0.528, accuracy=84.8%,
Valid: loss=0.613, accuracy=83.5%
Epoch  5,                  Train: loss=0.852, accuracy=77.9%,
Valid: loss=0.951, accuracy=76.3%
validation accuracy = 84.383
-----------------------------------------------------------------
Training model with a learning rate of 0.01 and a batch size of 200
Epoch  0,                  Train: loss=2.319, accuracy=8.2%,
Valid: loss=2.319, accuracy=7.9%
Epoch  1,                  Train: loss=0.471, accuracy=83.9%,
Valid: loss=0.476, accuracy=83.3%
Epoch  2,                  Train: loss=0.424, accuracy=85.5%,
Valid: loss=0.439, accuracy=84.8%
Epoch  3,                  Train: loss=0.499, accuracy=82.3%,
Valid: loss=0.521, accuracy=81.8%
Epoch  4,                  Train: loss=0.393, accuracy=86.3%,
Valid: loss=0.420, accuracy=85.7%
Epoch  5,                  Train: loss=0.423, accuracy=84.8%,
Valid: loss=0.454, accuracy=84.1%
validation accuracy = 85.717
-----------------------------------------------------------------
Training model with a learning rate of 0.01 and a batch size of 1000
Epoch  0,                  Train: loss=2.303, accuracy=4.9%,
Valid: loss=2.306, accuracy=4.8%
Epoch  1,                  Train: loss=0.541, accuracy=81.6%,
Valid: loss=0.549, accuracy=81.3%
Epoch  2,                  Train: loss=0.477, accuracy=83.9%,
Valid: loss=0.484, accuracy=83.9%
Epoch  3,                  Train: loss=0.444, accuracy=85.0%,
Valid: loss=0.457, accuracy=84.5%
Epoch  4,                  Train: loss=0.430, accuracy=85.2%,
Valid: loss=0.443, accuracy=85.2%
Epoch  5,                  Train: loss=0.421, accuracy=85.6%,
Valid: loss=0.441, accuracy=85.2%
validation accuracy = 85.233
-----------------------------------------------------------------
Training model with a learning rate of 0.001 and a batch size of 1
Epoch  0,                  Train: loss=2.300, accuracy=12.6%,
Valid: loss=2.297, accuracy=13.5%
Epoch  1,                  Train: loss=0.457, accuracy=84.8%,
Valid: loss=0.472, accuracy=84.4%
Epoch  2,                  Train: loss=0.500, accuracy=83.3%,
Valid: loss=0.534, accuracy=82.3%
Epoch  3,                  Train: loss=0.440, accuracy=85.7%,
Valid: loss=0.466, accuracy=85.2%
Epoch  4,                  Train: loss=0.455, accuracy=84.7%,
Valid: loss=0.496, accuracy=84.0%
Epoch  5,                  Train: loss=0.456, accuracy=85.0%,
Valid: loss=0.512, accuracy=83.7%
```

```
validation accuracy = 85.200
----------------------------------------------------------------
Training model with a learning rate of 0.001 and a batch size of 20
Epoch  0,                 Train: loss=2.298, accuracy=14.6%,
Valid: loss=2.296, accuracy=14.8%
Epoch  1,                 Train: loss=0.469, accuracy=84.3%,
Valid: loss=0.459, accuracy=85.1%
Epoch  2,                 Train: loss=0.426, accuracy=85.3%,
Valid: loss=0.420, accuracy=85.8%
Epoch  3,                 Train: loss=0.424, accuracy=85.2%,
Valid: loss=0.420, accuracy=85.9%
Epoch  4,                 Train: loss=0.410, accuracy=85.8%,
Valid: loss=0.414, accuracy=86.1%
Epoch  5,                 Train: loss=0.393, accuracy=86.4%,
Valid: loss=0.397, accuracy=86.7%
validation accuracy = 86.700
----------------------------------------------------------------
Training model with a learning rate of 0.001 and a batch size of 200
Epoch  0,                 Train: loss=2.284, accuracy=12.6%,
Valid: loss=2.285, accuracy=12.2%
Epoch  1,                 Train: loss=0.594, accuracy=80.8%,
Valid: loss=0.605, accuracy=80.6%
Epoch  2,                 Train: loss=0.517, accuracy=83.0%,
Valid: loss=0.532, accuracy=82.4%
Epoch  3,                 Train: loss=0.482, accuracy=83.9%,
Valid: loss=0.498, accuracy=83.2%
Epoch  4,                 Train: loss=0.464, accuracy=84.4%,
Valid: loss=0.481, accuracy=83.8%
Epoch  5,                 Train: loss=0.444, accuracy=85.0%,
Valid: loss=0.464, accuracy=84.2%
validation accuracy = 84.200
----------------------------------------------------------------
Training model with a learning rate of 0.001 and a batch size of 1000
Epoch  0,                 Train: loss=2.298, accuracy=15.7%,
Valid: loss=2.301, accuracy=15.4%
Epoch  1,                 Train: loss=0.842, accuracy=72.9%,
Valid: loss=0.849, accuracy=72.3%
Epoch  2,                 Train: loss=0.693, accuracy=77.8%,
Valid: loss=0.700, accuracy=77.0%
Epoch  3,                 Train: loss=0.625, accuracy=80.0%,
Valid: loss=0.632, accuracy=79.6%
Epoch  4,                 Train: loss=0.586, accuracy=80.9%,
Valid: loss=0.593, accuracy=80.3%
Epoch  5,                 Train: loss=0.555, accuracy=81.9%,
Valid: loss=0.562, accuracy=81.5%
validation accuracy = 81.483
```

Tableau pour la précision sur l'ensemble de validation

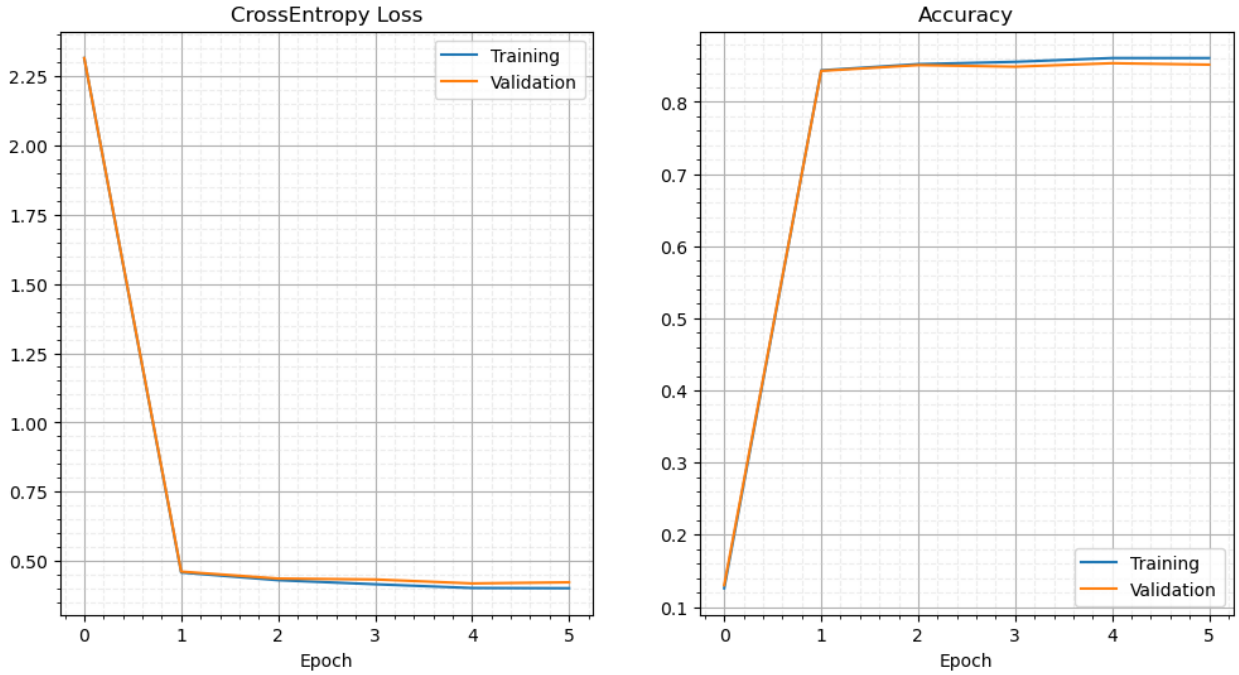
N.B. que les lignes correspondent aux valeurs du taux d'apprentisage et les colonnes correspondent au valeur du batch size. Les valeurs ci-dessous sont donné comme exemples; remplacez-les par les valeurs que vous avez utilisées pour votre recherche d'hyperparamètres.

learning rate\ batch_size	1	20	200	1000
0.1	82.183	83.233	80.783	84.833
0.01	82.900	84.383	85.717	85.233
0.001	85.200	86.700	84.200	81.483

Adam: Analyse du meilleur modèle

```
# ADAM
# Montrez les résultats pour la meilleure configuration trouvez ci-
dessus.
batch_size = 20 # TODO: Vous devez modifier cette valeur avec la
meilleur que vous avez eu.
lr = 0.001         # TODO: Vous devez modifier cette valeur avec la
meilleur que vous avez eu.

with torch.no_grad():
  data_loader_train, data_loader_val, data_loader_test =
get_fashion_mnist_dataloaders(val_percentage=0.1,
batch_size=batch_size)

  model = LinearModel(num_features=784, num_classes=10)
  best_model, best_val_accuracy, logger = train(model,lr=lr,
nb_epochs=5, sgd=False,

data_loader_train=data_loader_train, data_loader_val=data_loader_val)
  logger.plot_loss_and_accuracy()
  print(f"Best validation accuracy = {best_val_accuracy*100:.3f}")

  accuracy_test, loss_test =
accuracy_and_loss_whole_dataset(data_loader_test, best_model)
print("Evaluation of the best training model over test set")
print("------")
print(f"Loss : {loss_test:.3f}")
print(f"Accuracy : {accuracy_test*100.:.3f}")

Epoch  0,                 Train: loss=2.314, accuracy=12.6%,
Valid: loss=2.314, accuracy=13.0%
Epoch  1,                 Train: loss=0.459, accuracy=84.4%,
Valid: loss=0.462, accuracy=84.3%
Epoch  2,                 Train: loss=0.431, accuracy=85.2%,
Valid: loss=0.437, accuracy=85.1%
Epoch  3,                 Train: loss=0.417, accuracy=85.5%,
Valid: loss=0.434, accuracy=84.9%
```

```
Epoch  4,                  Train: loss=0.403, accuracy=86.1%,
Valid: loss=0.420, accuracy=85.4%
Epoch  5,                  Train: loss=0.402, accuracy=86.1%,
Valid: loss=0.424, accuracy=85.2%
Best validation accuracy = 85.350
Evaluation of the best training model over test set
------
Loss : 0.457
Accuracy : 84.080
```



Analyse des Résultats

Analyse des résultats numériques

Les résultats montrent que l'optimiseur Adam offre généralement de meilleures performances que SGD, surtout pour des batch sizes élevés (1000) et des learning rates élevés (0.1 et 0.01). Avec SGD, l'accuracy tend à diminuer lorsque le batch size augmente, notamment pour 1000, ce qui suggère une convergence plus difficile avec des mises à jour moins fréquentes. En revanche, Adam semble plus robuste face à l'augmentation du batch size, affichant une meilleure accuracy pour 200 et 1000 avec 0.01. Cela confirme qu'Adam, en adaptant dynamiquement le learning rate pour chaque paramètre, converge plus efficacement sur une large gamme d'hyperparamètres, alors que SGD est plus sensible à leur choix, nécessitant un tuning plus précis pour de bonnes performances.

Analyse des résultats graphiques

Les courbes montrent que Adam converge plus rapidement et plus efficacement que SGD. Avec SGD, la perte diminue progressivement mais reste légèrement plus élevée que celle d'Adam, et l'accuracy atteint un plateau autour de 0.85 après 5 époques. En revanche, avec Adam, la perte

diminue plus rapidement et l'accuracy atteint une valeur légèrement plus élevée en un temps plus court. Ces résultats confirment qu'Adam est plus efficace pour l'optimisation, car il ajuste dynamiquement le learning rate et compense les variations du gradient, alors que SGD est plus sensible aux hyperparamètres et converge plus lentement.