PyTorch : BASE

1 – Comparaison PyTorch VS Tensorflow

1. Utilise des objets natifs de Python
2. Exécute les séquences (pas de compilation)
3. Framework dynamique
4. Moins rapide que Tensorflow
5. Populaire chez les chercheurs
6. Pas d'API de haut niveau

Vocabulaire :

- TENSOR : Les tenseurs

En mathématiques, plus précisément en algèbre multilinéaire et en géométrie différentielle, un tenseur est un objet très général, dont la valeur s'exprime dans un espace vectoriel. On peut l'utiliser entre autres pour représenter des applications multilinéaires ou des multi vecteurs.

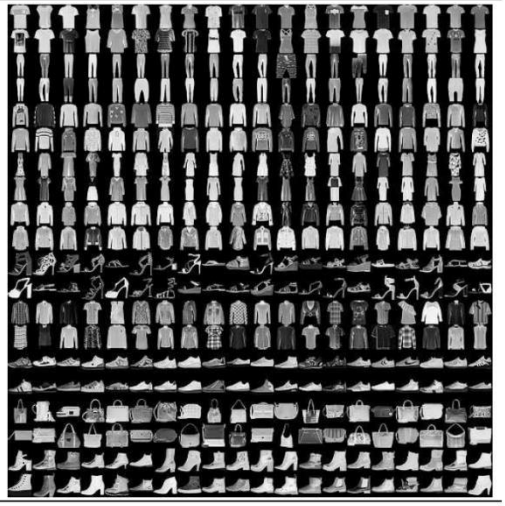
- GRADIENT :

En mathématiques et en physique, le gradient d'une fonction de plusieurs variables est un champ de vecteurs qui combine en chaque point les différentes dérivées partielles et donne ainsi à la fois la direction de la variation la plus forte localement et l'intensité de cette variation.

- VECTEUR :

En mathématiques, un vecteur est un objet généralisant plusieurs notions provenant de la géométrie, de l'algèbre, ou de la physique. Rigoureusement axiomatisée, la notion de vecteur est le fondement de la branche des mathématiques appelée algèbre linéaire.

Fashion MNIST :
Fashion MNIST est un jeu de données de qui contient 70 000 images en niveaux de gris répartie sur 1 des 10 catégories. Les images montrent des vêtements, d'articles de Zalando, en basse résolution.

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

L'Objectif : "Le trieur automatique"

Imagine que tu travailles dans un immense entrepôt de vêtements (Zalando). Tu as des milliers d'articles (chaussures, pulls, pantalons) mélangés. Ton patron te demande de créer un robot capable de regarder une photo de l'article et de dire instantanément : "C'est un sac" ou "C'est une basket".

L'objectif de ce code est de fabriquer le "cerveau" de ce robot.

Le Concept : "Apprendre par l'erreur"

Le code suit un cycle d'apprentissage en 4 étapes majeures :

1. La préparation (Les livres de classe)

Le code télécharge 60 000 images de vêtements.

C'est la base de données.

On sépare les données en deux :

- Le groupe d'entraînement : Les images que le robot a le droit de regarder pour apprendre.
- Le groupe de test : Les images "examen" qu'il n'a jamais vues, pour vérifier s'il est devenu intelligent ou s'il a juste appris par cœur.

2. Le Modèle (Le cerveau vide)

La partie NeuronalNetwork crée une structure de neurones artificiels.

Au début, ce cerveau est "bête" : il répond au hasard.

Si tu lui montres une chaussure, il dira peut-être "C'est une chemise".

3. L'Entraînement (La correction)

C'est la boucle train. Voici ce qu'il se passe des milliers de fois :

1. Prédiction : Le robot regarde une image et donne une réponse.
2. Calcul de la perte (Loss) : On compare sa réponse à la vérité.
3. S'il s'est trompé de beaucoup, la "perte" est élevée.
4. Optimisation : C'est l'étape magique. Le robot ajuste légèrement ses connexions internes pour que, la prochaine fois qu'il verra cette image, il se rapproche de la bonne réponse.

4. L'Époque (La répétition)

Une "Époque" (Epoch), c'est quand le robot a fini de regarder les 60 000 images une fois.

On recommence 5 fois (5 époques) pour que les leçons "rentrent bien dans sa tête".

En résumé, ce que fait chaque bloc :

- **DataLoader :** Il distribue les images par petits paquets (comme des cartes de jeu) pour ne pas fatiguer l'ordinateur.
- **ReLU :** C'est un filtre qui aide le cerveau à comprendre les formes complexes (les courbes, les angles).
- **Backpropagation :** C'est le fait de revenir en arrière pour corriger les erreurs.

Le résultat final

À la fin des 5 époques, tu verras le pourcentage de précision augmenter.

Le robot passera peut-être de 10% de réussite (hasard total) à 80% ou plus.

Il est maintenant capable de reconnaître un vêtement qu'il n'a jamais vu auparavant.

LE ROBOT NE REGARDE PAS DES IMAGES MAIS UNE GRILLE DE NOMBRE :

Voici comment le robot "voit" réellement :

1. La transformation en grille (Matrice)

L'image d'un vêtement dans FashionMNIST fait 28 pixels de large sur 28 pixels de haut. Pour l'ordinateur, chaque pixel est une case contenant un chiffre :

- **0** si le pixel est totalement noir.
- **255** (ou 1.0 en PyTorch) si le pixel est totalement blanc.
- Toute la gamme entre les deux pour les nuances de gris.

2. Le "Flatten" (L'aplatissement)

Dans ton code, il y a une ligne `self.flatten = nn.Flatten()`. C'est une étape cruciale : Comme le cerveau artificiel que tu as construit (les couches `Linear`) ne sait pas lire une grille (tableau 2D), on "déroule" l'image.

- On prend la première ligne de 28 pixels, on colle la deuxième ligne à la suite, puis la troisième...
- À la fin, l'image n'est plus un carré, mais une **longue liste de 784 chiffres** (28×28=784).

3. Les poids : l'importance des cases

L'objectif du robot est de découvrir quels chiffres sont importants. Par exemple :

- Si les cases au milieu de la liste sont "allumées" (chiffres élevés), c'est peut-être un **T-shirt**.
- Si ce sont les cases tout en bas, c'est peut-être une **paire de chaussures**.

Le robot ne "comprend" pas ce qu'est un tissu, il comprend que **"si les cases (14,12) et (14,13) sont blanches, alors il y a 80% de chances que ce soit un sac"**.

4. Les "Logits" : la décision finale

À la toute fin, le robot sort 10 chiffres (un pour chaque catégorie : Pull, Pantalon, etc.).

- Le chiffre le plus élevé dans cette liste de 10 devient la réponse du robot.
- Si le 9ème chiffre est le plus grand, il dit : "C'est une bottine !".

Pour résumer : Ton code transforme la lumière (l'image) en électricité (les chiffres), puis utilise des calculs mathématiques pour trouver des modèles récurrents dans ces chiffres.

Rendre le modèle utilisable :

Voici les explications détaillées de tes nouveaux ajouts :

1. Le "Cerveau de Rechange" (Chargement du modèle)

 Le Concept : Au lieu de repartir d'un cerveau vide (aléatoire) à chaque fois que tu lances le script, tu demandes à PyTorch de lire le fichier `model.pth`.

 L'Objectif : C'est la **mémoire**. Si tu as déjà entraîné ton modèle hier pendant 2 heures, tu récupères toute son expérience instantanément. C'est indispensable pour améliorer un modèle sur le long terme sans tout recommencer à zéro.

2. Le "Coffre-Fort" (Sauvegarde du modèle)

 Le Concept : À la fin de ton entraînement, tu "figes" les réglages (les fameux poids et connexions numériques) dans un fichier sur ton disque dur.

 L'Objectif : Ne pas perdre les progrès. On sauvegarde souvent à la fin des époques pour pouvoir envoyer le fichier à un ami ou l'utiliser plus tard dans une application réelle.

3. Le "Dictionnaire" (La liste `classes`)

 Le Concept : Comme on l'a dit, le robot ne comprend que les chiffres. En sortie, il te dit "9". Pour un humain, "9" ne veut rien dire.

 L'Objectif : Faire la **traduction**. Cette liste permet d'associer l'indice 9 au mot "Chaussure femme". C'est l'interface entre le langage mathématique et le langage humain.

4. La Mise en Pratique (Prédiction finale)

Ici, tu ne fais plus d'entraînement, tu passes à l'**utilisation concrète** :

- model.eval() : Tu dis au robot : "C'est l'examen, concentre-toi, n'apprends plus rien".
- test_data[0] : Tu prends la toute première image du dossier de test (que le robot n'a pas vue pendant l'entraînement).
- pred[0].argmax(0) : Le modèle sort 10 probabilités (ex: 0.1, 0.0, 0.8...). argmax va chercher la plus grande (ici le 0.8 à l'indice 2).
- **L'Objectif :** Vérifier visuellement si le robot a raison en comparant ce qu'il a "deviné" (predicted) avec la vraie étiquette (actuel).

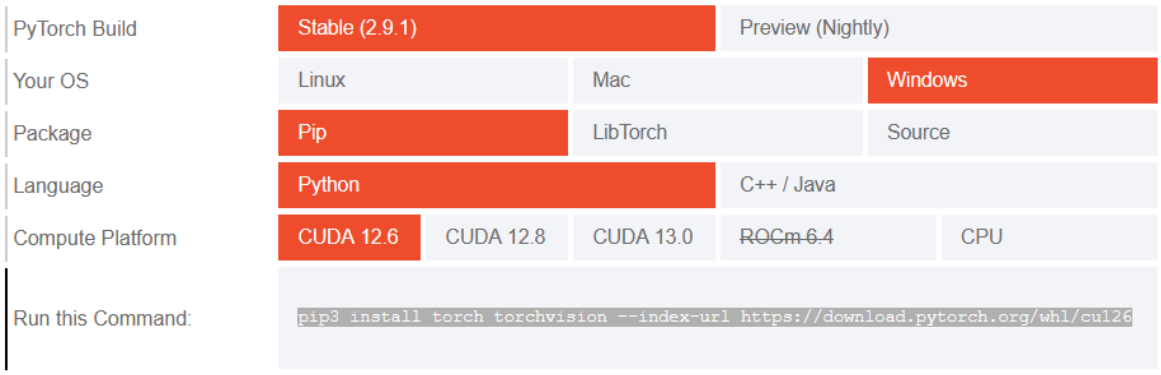
Une petite remarque pour ton code :

La première fois que tu vas lancer ce code, il risque de faire une erreur car le fichier model.pth n'existe pas encore !

***Conseil :** Pour le tout premier lancement, mets un # devant la ligne model.load_state_dict(...) pour la désactiver. Une fois que le script aura tourné une fois et créé le fichier, tu pourras retirer le # pour les fois suivantes.*

LE CODE : main.py

Installer les outils : https://pytorch.org/get-started/locally/



Import des outils dans main.py

```python
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
```

1. Préparation des données :

```python
# ========================================
# 1. PRÉPARATION DES DONNÉES
# ========================================

# Téléchargement du jeu de données d'entraînement
training_data = datasets.FashionMNIST(
    root="./data",
    train=True,
    download=True,
    transform=ToTensor()
)

# Téléchargement du jeu de données de test
test_data = datasets.FashionMNIST(
    root="./data",
    train=False,
    download=True,
    transform=ToTensor()
)

# Création des DataLoaders : ils gèrent le passage des données par "lots" (batches)
# Cela évite de saturer la mémoire vive du PC.
batch_size = 64
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)
```

2. CONFIGURATION DU MATERIEL (CPU vs GPU)

```python
# ========================================
# 2. CONFIGURATION DU MATÉRIEL (CPU vs GPU)
# ========================================

# On vérifie si une carte graphique (CUDA ou MPS) est disponible, sinon on utilise le processeur (CPU)
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Utilisation du matériel : {device}")
```

3. LA CLASSE DE DEFINITION DU MODELE

```python
# ========================================
# 3. DÉFINITION DE L'ARCHITECTURE DU MODÈLE
# ========================================

class NeuronalNetwork(nn.Module):  1 usage
    def __init__(self):
        super().__init__()
        # On aplatit l'image 28x28 en un vecteur de 784 pixels
        self.flatten = nn.Flatten()
        # Définition des couches du réseau (Linaire -> Activation ReLU -> ...)
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28 * 28,  out_features: 512),
            nn.ReLU(),
            nn.Linear( in_features: 512,  out_features: 512),
            nn.ReLU(),
            nn.Linear( in_features: 512,  out_features: 10),  # 10 sorties car il y a 10 types de vêtements
        )

    # La fonction forward définit comment les données passent dans le réseau
    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits


# On crée l'instance du modèle et on l'envoie sur le processeur choisi (CPU ou GPU)
model = NeuronalNetwork().to(device)
# On charge dernière sauvegarde du model pour améliorer de + en + sa précision
model.load_state_dict(torch.load( f: "model.pth", weights_only=True))
```

4. FONCTION D'ENTRAINEMENT ET DE TEST

```python
def train(dataloader, model, loss_fn, optimizer):  2 usages (1 dynamic)
    size = len(dataloader.dataset)
    model.train()  # On passe le modèle en mode entraînement
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # 1. Calcul de l'erreur (prediction vs réalité)
        pred = model(X)
        loss = loss_fn(pred, y)

        # 2. Backpropagation (le coeur de l'apprentissage)
        optimizer.zero_grad()  # On remet les gradients à zéro
        loss.backward()  # On calcule les erreurs pour chaque neurone
        optimizer.step()  # On ajuste les poids du réseau

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"Perte : {loss:>7f} [{current:>5d}/{size:>5d}]")
```

```python
def test(dataloader, model, loss_fn):  1 usage
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()  # On passe le modèle en mode évaluation (désactive le Dropout, etc.)
    test_loss, correct = 0, 0

    # On désactive le calcul des gradients pour aller plus vite (inutile en test)
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            # On compte le nombre de bonnes réponses
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    print(f"Résultats du test : \n Précision: {(100 * correct):>0.1f}%, Perte moyenne: {test_loss:>8f} \n")
```

5. LANCEMENT DE L'APPRENTISSAGE

```python
# ==========================================
# 5. LANCEMENT DE L'APPRENTISSAGE
# ==========================================

# On définit la fonction de perte (CrossEntropy est idéal pour la classification)
# et l'optimiseur (SGD = Descente de gradient stochastique)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

# La boucle finale qui fait tourner les époques
epoques = 5
for t in range(epoques):
    print(f"Époque {t + 1}\n-------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)

print("Entraînement terminé !")
```

6. SAUVEGARDE DU MODELE ET UTILISATION

```python
# ========================================
# 6. SAUVEGRADER LE MODELE
# ========================================
torch.save(model.state_dict(), f: "model.pth")
print("Le modele PyTorch a été sauvegarder dans le fichier model.pth")

# ========================================
# 7. LE MODELE ENTRAINER TOUVE LE VETEMENT
# ========================================
#Tableau de vetement disponible = 10 (catégories)
classes = [
    "T-shirt/top",
    "Pentalon",
    "Pullover",
    "Robe",
    "Blouson",
    "Sandale",
    "Chemise",
    "Basquette",
    "Sac à main",
    "Chaussure femme"
]

model.eval()
x,y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    x = x.to(device)
    pred = model(x)
    predicted, actuel = classes[pred[0].argmax(0)], classes[y]
    print(f"Dernière prédiction : {predicted}, actuel {actuel}")
```