Compte rendu du TP 1 – 2024/2025
Étudiant : FALL Madou, classe M2 TSI

Rapport de TP : Conditional GAN sur le jeu de données MNIST

Introduction
Les Generative Adversarial Networks (**GANs**) sont des modèles génératifs capables de produire des données réalistes à partir d'une distribution de données d'entraînement. Les Conditional GANs (cGANs) étendent cette capacité en permettant de générer des données conditionnées par des étiquettes, ce qui est particulièrement utile pour des tâches comme la génération d'images spécifiques à une classe. L'objectif de ce TP est de mettre en œuvre un Conditional GAN pour générer des images de chiffres manuscrits à partir du jeu de données MNIST.

Installation des Dépendances
Pour commencer, nous avons installé le package tensorflow/docs, qui fournit des outils utiles pour **TensorFlow**, notamment pour l'affichage de code et de modèles dans des notebooks Jupyter. Cette installation a été réalisée avec la commande suivante :

```
!pip install -q git+https://github.com/tensorflow/docs
```

Ce package est utilisé pour améliorer la présentation du code et des résultats dans un environnement interactif.

Importation des Bibliothèques
Plusieurs bibliothèques ont été importées pour construire, entraîner et visualiser le modèle cGAN. Voici les principales :

- **TensorFlow et Keras** : Pour la construction et l'entraînement des modèles.
- **NumPy** : Pour la manipulation de tableaux.
- **Matplotlib** : Pour la visualisation des images et des résultats.
- **PIL** (Python Imaging Library) : Pour la création d'animations GIF.

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

Prétraitement des Données
Pour entraîner notre modèle cGAN, nous avons utilisé l'intégralité du jeu de données MNIST, comprenant 70 000 images (60 000 pour l'entraînement et 10 000 pour le test). Voici les étapes de prétraitement appliquées :

- Normalisation : Les valeurs des pixels ont été normalisées dans la plage [0, 1] pour faciliter la convergence du modèle.
- Redimensionnement : Les images ont été redimensionnées pour inclure une dimension de canal (28x28x1), nécessaire pour les couches convolutives.
- Encodage des Étiquettes : Les étiquettes ont été converties en format "one-hot" pour permettre une génération conditionnelle par classe.

Un dataset TensorFlow a été créé à partir des données prétraitées. Les données ont été mélangées pour éviter tout biais dans l'ordre des échantillons, puis divisées en lots de taille 64 pour l'entraînement.

- Images : (70000, 28, 28, 1)
- Étiquettes : (70000, 10)

Architecture du Modèle

Le modèle cGAN est composé de deux réseaux principaux : le **générateur** et le **discriminateur**.

a) Discriminateur

Le discriminateur est un réseau de neurones convolutifs qui prend en entrée une image et une étiquette de classe, et décide si l'image est réelle ou générée. Son architecture est la suivante :

- Couches Conv2D : Pour extraire les caractéristiques spatiales.
- LeakyReLU : Fonction d'activation pour introduire de la non-linéarité.
- GlobalMaxPooling2D : Pour réduire les dimensions de l'image.
- Couche Dense : Pour la classification binaire (réel vs généré).

```
discriminator = keras.Sequential(
    [
        keras.layers.InputLayer((28, 28, discriminator_in_channels)),
        layers.Conv2D(64, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(negative_slope=0.2),
        layers.Conv2D(128, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(negative_slope=0.2),
        layers.GlobalMaxPooling2D(),
        layers.Dense(1),
    ],
    name="discriminator",
)
```

b) Générateur

Le générateur prend en entrée un vecteur latent et une étiquette de classe, et génère une image. Son architecture est la suivante :

- Couche Dense : Pour transformer le vecteur latent en une forme utilisable.
- LeakyReLU : Fonction d'activation.
- Couches Conv2DTranspose : Pour augmenter la taille de l'image.
- Sigmoid : Activation finale pour produire des valeurs de pixel entre 0 et 1.

```
generator = keras.Sequential(
    [
        keras.layers.InputLayer((generator_in_channels,)),
     # We want to generate 128 + num_classes coefficients to reshape into a
     # 7x7x(128 + num_classes) map.
        layers.Dense(7 * 7 * generator_in_channels),
        layers.LeakyReLU(negative_slope=0.2),
        layers.Reshape((7, 7, generator_in_channels)),
        layers.Conv2DTranspose(128, (4, 4), strides=(2, 2),
padding="same"),
        layers.LeakyReLU(negative_slope=0.2),
        layers.Conv2DTranspose(128, (4, 4), strides=(2, 2),
padding="same"),
        layers.LeakyReLU(negative_slope=0.2),
        layers.Conv2D(1, (7, 7), padding="same", activation="sigmoid"),
```

```
    ],
    name="generator",
)
```

Entraînement du Conditional GAN
La classe ConditionalGAN gère l'entraînement adversariale entre le générateur et le discriminateur. Voici les étapes clés de l'entraînement :

1. **Initialisation** :
    - Le discriminateur et le générateur sont initialisés avec leurs architectures respectives.
    - La dimension du vecteur latent est définie (latent_dim = 128).
2. **Compilation** :
    - Les optimiseurs (`Adam`) et la fonction de perte (`Binary Crossentropy`) sont configurés.
3. **Étape d'Entraînement** :
    - Le discriminateur est entraîné pour distinguer les images réelles des images générées.
    - Le générateur est entraîné pour tromper le discriminateur en produisant des images réalistes.
    - Les pertes sont suivies et retournées pour surveiller la convergence.

Les pertes du générateur et du discriminateur sont surveillées pendant l'entraînement. Une convergence réussie se traduit par une diminution progressive des deux pertes, indiquant que le générateur produit des images réalistes et que le discriminateur apprend à les distinguer.
La classe ConditionalGAN est une implémentation robuste d'un GAN conditionnel, capable de générer des images réalistes en fonction d'étiquettes spécifiques. Son architecture et son processus d'entraînement sont conçus pour garantir une convergence stable et des résultats de haute qualité.

Génération d'Images Interpolées
Une fois le modèle entraîné, il a été utilisé pour générer des images interpolées entre deux classes spécifiques. Voici les étapes suivies :

a) **Génération du bruit latent** : Un vecteur de bruit est généré et interpolé entre deux classes.
b) **Interpolation des étiquettes** : Les étiquettes sont interpolées linéairement entre les deux classes.
c) **Génération des images** : Le générateur produit des images à partir du bruit et des étiquettes interpolées.
d) **Création d'un GIF** : Les images générées sont sauvegardées sous forme d'une animation GIF pour visualiser la transition entre les classes.

Résultats
Le modèle a été capable de générer des images réalistes de chiffres manuscrits en fonction des étiquettes conditionnelles. Les images interpolées montrent une transition fluide entre les classes, démontrant la capacité du modèle à apprendre la distribution des données.

Conclusion
Ce TP a permis de mettre en œuvre un **Conditional GAN** pour générer des images réalistes à partir du jeu de données MNIST. Les résultats montrent que le modèle est capable de produire des images de qualité en fonction des étiquettes conditionnelles. Ce travail ouvre des perspectives pour des applications plus avancées, comme la génération d'images en haute résolution ou la manipulation d'images conditionnelles.