Rapport du projet GAN (Generative Adversarial Networks)

# Description du projet

Ce projet consiste à réaliser l’entrainement d’un GAN en utilisant la librairie tensorflow.

L’entrainement des modèles sont conçu pour s’entrainer sur n’importe quelle source de catégorie d’images. Cependant, l’entrainement est particulièrement adapté pour les images de catégorie ‘handwriting’ (plus précisément les images contenues dans ce lien <https://www.kaggle.com/scolianni/mnistasjpg>). Pour d’autres source d’images il faudrait peut-être modifier les hyperparamètres des modèles, modifier la définition des modèles, etc… pour avoir de bons résultats.

Il est fortement recommander d’exécuter le projet à travers un notebook (comme main.ipynb) pour que l’affichage graphique fonctionne correctement.

# Fonctionnement des GAN

Les GAN utilisent 2 modèles qui compétitionnent entre eux. L’un généralement appelé ‘Generator’ et l’autre généralement appele ‘Discriminiator’.

Le ‘Generator’ est utilisé pour générer un resultat (une image dans notre cas).

Le ‘Discriminiator’ est utilisé pour classifier les fausses images (générer par le ‘Generator’) et les vraies images.

Le point crucial des GAN est que les 2 modèles base en partie leurs performances dépendamment de **la performance inverse l’autre modèle**.

La performance du ‘Generator’ se basée sur la performance inverse du ‘Discriminiator’ à prédire si l’image est une fausse image (**la performance inverse l’autre modèle**).

La performance du ‘Discriminiator’ est basée sur à quel point il est capable de prédire si l’image est une vraie image et a quel point il est capable de prédire si l’image est une fausse image (**la performance inverse l’autre modèle**).

C’est cette évaluation de performance qui génère une ‘compétition’ entre les deux modèles puisque si la performance de l’un devient meilleure, la performance de l’autre se dégrade. Cependant, si la performance d’un modèle se dégrade, le modèle s’adaptera pour augmenter sa performance a une intensité plus élevée. Lorsque sa performance augmentera, la performance de l’autre se dégrade et le cycle continu.

# Particularités de la réalisation d’un GAN

La définition des modèles des GAN n’a rien de particulier aux autres modèles standard. En revanche, puisque l’évaluation des performances lors de l’entrainement des modèles diffère de ce qui est normalement utilisé, il faut donc redéfinir la façon dont les modèles sont entrainés. C’est ce qui explique l’existence du module Train/train.py dans le projet.

Nous pouvons voir dans ce module 2 fonctions particuliere qui ont pour but de redéfinir les ‘loss functions’ standard de tensorflow :

Text

Description automatically generated

Nous pouvons voir :   
 - **cross\_entropy(tf.zeros\_like(fake\_output), fake\_output)** pour le ‘Discriminiator’  
 - **cross\_entropy(tf.ones\_like(fake\_output), fake\_output)** pour le ‘Generator’

Ces deux parties sont la performance inverse de l’autre modèle.

Ensuite, nous définissons la méthode pour entrainer le modèle ( train() ). Vous pouvez allez y jeter un coup d’œil mais cette méthode est moins pertinente pour bien comprendre le fonctionnement du GAN.

# Structure du projet

* **Datas :**   
  Dossier contenant un module avec des fonctions pour charger les données.
* **Displayers** :  
  Dossier contenant un module avec des fonctions pour afficher de l’information des objets utilisés dans le projet.
* **Models :**Dossier contenant 2 module avec des fonctions pour charger le model ‘Discriminiator’ et ‘Generator’.
* **Train :**Dossier contenant un module avec des fonctions pour entrainer les modèles.
* **GAN.py :**Classe pour définir et exécuter tous les éléments nécessaires pour la réalisation d’un GAN.

# Bibliographie

* <https://www.tensorflow.org/tutorials/generative/dcgan>
* <https://youtu.be/8L11aMN5KY8>

# Quelques résultats

## MNIST Dataset (handwriting) :