Atelier 2 d'optimisation: Algorithme de Réseaux Génératifs Antagonistes (GAN) basé sur la Théorie des jeux

Introduction :

Dans cet atelier, nous allons nous intéresser à des modèles génératifs qui modélisent traditionnellement la probabilité jointe P(X,Y) et donc la vraisemblance des données X.

En particulier, les Réseaux Génératifs Antagonistes (Generative Adversarial Networks (GAN)) sont considérés dans ce contexte, qui basés sur des modèles génératifs. Depuis leur introduction en 2014, les GAN ont un impact important sur l'apprentissage non-supervisé. Leur capacité à modéliser les structures sous-jacentes aux données les place au cœur des techniques état de l'art en génération, complétion et modification d'images

Travail demandé :

L'objectif d'un modèle GAN est d'être capable de générer une image $\tilde{x}$ à partir d'une entrée z tirée selon un distribution pré-définie, et ce quel que soit le z choisi parmi la distribution.

Ceci peut être modélisé par l'équation mathématique suivante :

$$\tilde{x} = G(z), z \sim P(z). \qquad (1)$$

L'objectif du GAN est de générer des données réalistes, c'est-à-dire plausibles selon une distribution de probabilité des données réelles P(X). Évidemment, cette distribution est inconnue et on possède simplement des exemples de données tirées selon cette distribution

Data = $\{x_i \sim P(X)\}_{i=1,2,..N}$, : les exemples d'apprentissage du dataset.

1. **Générateur:** Pour résoudre cet problème, on va définir un premier réseaux de neurones, le générateur G, dont l'objectif est de transformer n'importe quel entrée z tiré selon une distribution fixées – généralement $U_{[-1,1]}$ ou $N(0,1)$ en une image $\tilde{x}$ réaliste, telle que :

   $$\tilde{x} = G(z), z \sim P(z). \qquad (2)$$

   Question : Ecrire un programme sur Python qui permet de générer une dataset.
2. **Discriminateur :** La fonction de coût permettant l'apprentissage de ce générateur n'est cependant pas évidente, car la distribution P(X) n'est pas connue. Pour contourner ce problème, on propose d'apprendre un second réseau de neurones, le discriminateur D.

Ce réseau permet de prendre une image en entrée et doit prédire si cette image est une image réelle x^*du jeu de données ou si c'est une image x^* produite par le générateur.

A la sortie du discriminateur, on a les deux cas suivants à traiter

$$D(x) \in [0,1], \begin{cases} D(\tilde{x}) = 0 \; si: \tilde{x} = G(z) \\ D(x^*) = 1, si: x^* \in Data. \end{cases} \quad (3)$$

Question : Ecrire un programme sur Python qui permet de générer un discriminateur suivant ce modèle.

3. **Apprentissage adversaire :** L'apprentissage d'un GAN est particulier, et peut être assimilé à un "jeu" (au sens de la théorie des jeux) où deux adversaires sont en compétition (connu aussi sous le nom de problème des attaques adversariales) : le générateur apprend continuellement à tromper le discriminateur alors que le discriminateur apprend continuellement à s'adapter aux modifications du générateur pour toujours réussir à distinguer les exemples réels des exemples générés.

   Le coût de classification du discriminateur sera une « binary cross-entropy ».

   On cherche donc à optimiser le problème suivant :

$$\min_G \max_D \mathrm{E}_{x^* \in Data} [logD(x^*)] + E_{\mathrm{z} \sim \mathrm{P(z)}}[\log (1 - D(G(z)))], \quad (4)$$

   où, E[X] désigne l'espérance de X.

   Questions :

   1. Définir le jeu utilisé dans cette partie.
   2. En déduire pourquoi les GANs peuvent être modélisés par des problèmes min-max.
   3. Interpréter l'équation (4) par rapport au jeu déjà mentionné.
   4. Tester les deux blocs : générateur et discriminateur pour un ensemble d'images, en précisant à chaque fois les paramètres utilisés.