Compte rendu

Théorie des jeux : Principe et Application

Souheib Ben Mabrouk

Hassen Chtara

INDP2E

7 novembre 2021

Introduction

La théorie des jeux se propose d'étudier des situations (appelées « jeux ») où des individus (les « joueurs ») prennent des décisions, chacun étant conscient que le résultat de son propre choix (ses « gains ») dépend de celui des autres.[1] C'est pourquoi on dit parfois de la théorie des jeux qu'elle est une « théorie de la décision en interaction ».

Les décisions ayant pour but **un gain maximum** elles relèvent d'un **comportement rationnel**, elles peuvent se prêter au traitement mathématique : calcul d'extremums, approche probabiliste.

La théorie des jeux est de ce fait parfois présentée comme une branche des mathématiques , il est vrai que des mathématiciens (Émile Borel et John von Neumann) sont à son origine, et qu'elle demeure essentiellement le fait de mathématiciens.
Pour que ceux-ci puissent utiliser leurs techniques, il faut toutefois que le contexte dans lequel les décisions sont prises soit spécifié avec précision. D'où le recours à des hypothèses extrêmement fortes, notamment en ce qui concerne l'information dont dispose chacun, qui conduisent parfois à des conclusions fort peu intuitives. Ces conditions imposées par le traitement mathématique font que les analyses de la théorie des jeux se prêtent mal à une présentation peu formelle, « littéraire », s'appuyant sur l'intuition, qui souvent en donne une vision erronée.

L'objectif de ce rapport est de présenter ce concept dans le cadre du cours de l'optimisation ainsi que implémenter quelques cas d'étude en se basant sur l'exemple de l' **Algorithme de Réseaux Génératifs Antagonistes (GAN) basé sur la Théorie des jeux.**

Pour ce faire, Nous avons choisi de travailler avec **Python 3.8** disponible en ligne par le recours à **Google Colaboratory** qui nous facilite le travail grâce à sa simplicité et son efficacité en termes de temps d'exécution.
D'autre part, Nous utiliserons le système de composition de documents **LATEX** afin d'avoir un rapport bien conçu avec les formulations mathématiques nécessaires.

Table des matières

Table des figures

Chapitre 1

Principe

1.1 Présentation

La théorie des jeux est une discipline théorique qui permet de comprendre (formellement) des situations dans lesquelles les joueurs, les preneurs de décision, interagissent.[2]

Un jeu est alors définit comme un univers dans lequel chaque preneur de décision possède un ensemble d'actions possibles déterminé par les règles du jeu. Le résultat du jeu dépend alors conjointement des actions prises par chaque preneur de décision.
Cette discipline possède de nombreuses applications permettant notamment de comprendre des phénomène économiques, politiques ou même biologiques. Parmi ces phénomènes, voici une liste de situations dans lesquelles la théorie des jeux peut être appliquée :

- La concurrence entre entreprises
- La concurrence entre hommes politiques
- Un jury devant s'accorder sur un verdict
- Des animaux se battant pour une proie
- La participation à une enchère
- Le vote d'un législateur soumis à la pression de lobbies

Comme toute discipline théorique, la théorie des jeux consiste en une collection de modèles. Ces modèles sont alors des abstractions utilisées pour comprendre ce qui est observé ou vécu. Ils permettent de prédire l'évolution d'un jeu ou de conseiller le ou les joueurs sur **le meilleur coup à jouer.**

1.1.1 Modélisation d'un jeu

Afin de décrire ou de modéliser un jeu (c'est-à-dire une situation d'interaction décisionnelle) quatre éléments doivent être précisés :

- L'ensemble des joueurs (ou preneurs de décisions) N = 1, .., N
- Les actions possibles pour chaque joueur S1, . . ., SN
- Les règles du jeu spécifiant notamment l'ordre dans lequel les joueurs jouent et quand le

jeu se termine
— Le résultat du jeu pour chaque fin possible et son implication en termes de ”fonction de paiement” (pour cela il faut connaître les préférences des joueurs) ui, i=1, . . ., n.

1.1.2 La théorie du choix rationnel

La théorie du choix rationnel est une composante majeure de la plupart des modèles de théorie des jeux. Cette théorie stipule qu'étant données ses préférences, un joueur choisit la meilleure action parmi celles disponibles. On n'impose toutefois aucune restriction qualitative sur les préférences du joueur. La ”rationalité” tient dans la cohérence des actions d'un joueur avec ses préférences, pas de la nature de ces préférences.

On définira, la fonction de paiement u déterminant la ”satisfaction” des joueurs. La fonction de paiement u représentera alors les préférences d'un joueur si pour toute actions a et b (dans l'ensemble des actions possible) : u(a) > u(b) si et seulement si le joueur préfère a à b

1.2 Exemples d'applications de la théorie des jeux dans le domaine des Télécommunications

Il existe une grande pertinence de l'utilisation de la théorie des jeux dans la modélisation et la résolution de certains problèmes liés au routage de l'information dans les réseaux informatiques.

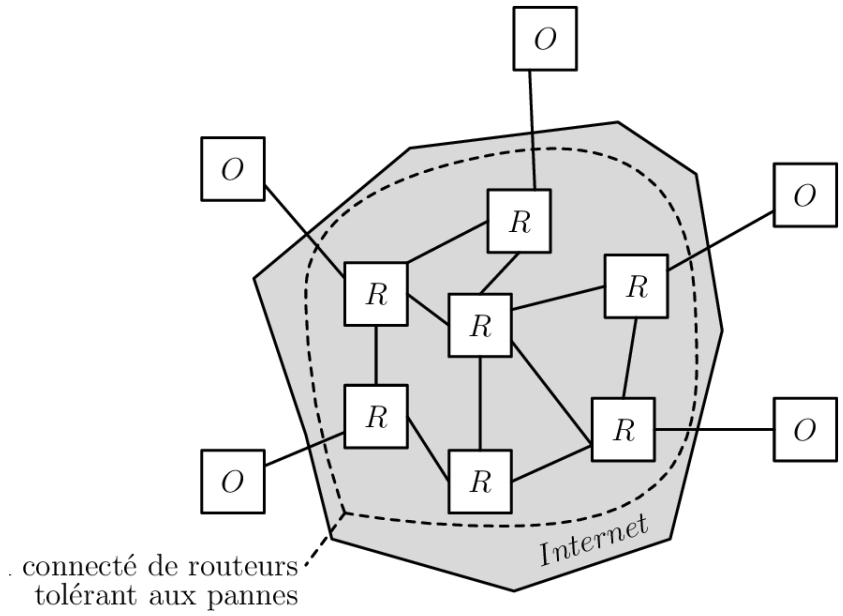


Fig. 1.1 – Modélisation de routage par la théorie des jeux

1.2.1 Le modèle de Wardrop

C'est un modèle de routage réseau conçu pour représenter des problèmes de routage avec une infinité d'utilisateurs, chacun étant responsable d'une partie infinitésimale du trafic. Chaque utilisateur est assigné à une commodité, dans chacune de ces commodités, une certaine quantité de trafic doit étre routée, depuis une source donnée jusqu'à une destination donnée, selon un ensemble de plus courts chemins qui induisent un sous-graphe du réseau général.

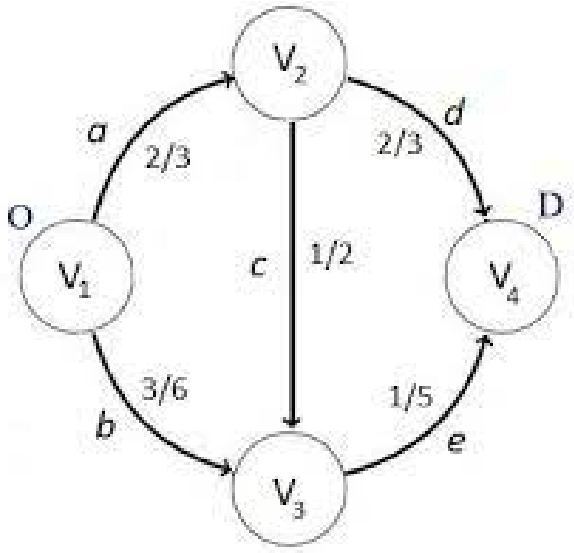


Fig. 1.2 – Modélisation du modèle de Wardrop de gestion de traffic

1.2.2 Allocation des tâches et théorie des jeux

On considère m machines (serveurs), n tâches (égoistes) ayant pour temps d'exécution w1,..,wn et ayant un ensemble de choix de serveurs Sj pour la tâche j.
Charge Li d'un serveur i = somme des temps d'exécution sur le serveur.
Coût d'une tâche = charge d'un serveur l'hébergeant.
1 joueur = 1 tâche.
L'ensemble des stratégies d'un joueur j = l'ensemble des serveurs Sj qui peut l'héberger.
L'utilité d'un joueur j : u j (M) = L i, sachant que dans la configuration M, le joueur j choisit d'être placé sur le serveur i.

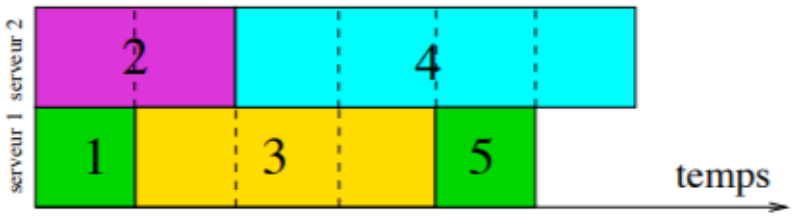


Fig. 1.3 – Modélisation du modèle d'allocation des tâches

1.2.3 Jeux de congestion

Dans un jeu de congestion, chaque agent choisit un ensemble de ressources. La fonction de coût pour chaque ressource ne dépend que du nombre d'agents qui ont choisi la ressource.

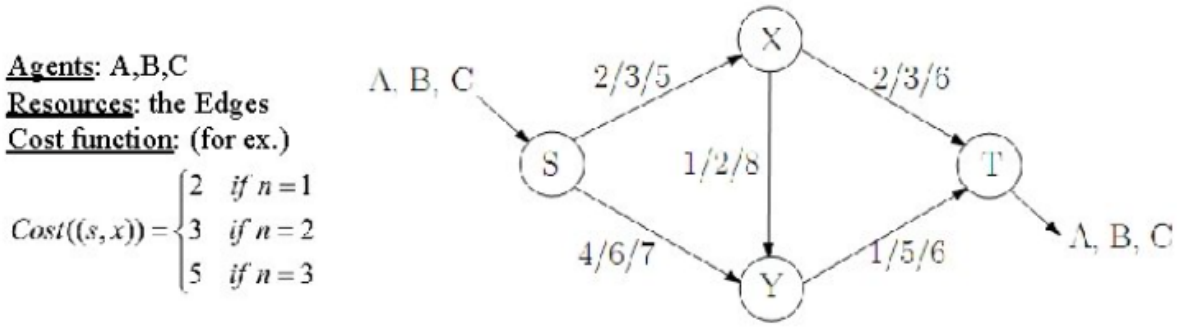


Fig. 1.4 – Modélisation du modèle de Jeux de congestion

1.3 Exemples de jeux utilisés dans la littérature

1.3.1 Un peu d'histoire

L'analyse du duopole d'Antoine Augustin Cournot publiée en 1838 dans ses Recherches sur les principes mathématiques de la théorie des richesses peut être considérée comme la première formulation, dans un cadre particulier, de la notion d'équilibre de Nash. [3]

Dans son ouvrage de 1938, Applications aux Jeux de Hasard, Émile Borel développe **un théorème du minimax pour les jeux à somme nulle à deux joueur**, c'est-à-dire les jeux dans lesquels ce que gagne l'un est perdu par l'autre. John von Neumann.
La théorie des jeux devient un champ de recherche à part entière avec la publication de Theory of Games and Economic Behavior (Théorie des jeux et du comportement économique) par John von Neumann et Oskar Morgenstern en 1944. Cet ouvrage fondateur détaille **la méthode de résolution des jeux à somme nulle.**

Vers 1950, John Forbes Nash formalise une notion générale d'équilibre qui portera **le nom d'équilibre de Nash**. Cette notion généralise les travaux de Cournot en incluant en particulier la possibilité de randomisation des stratégies.

1.3.2 Exemple de jeux de référence

Pile ou face :

- Deux joueurs
- Chaque joueur dispose de deux choix : Pile ou Face
- Règles du jeu :

Choix simultanés (en temps)
Joueur 1 gagne si les choix sont identiques
Joueur 2 gagne si les choix sont différents
Joueurs en opposition : si un joueur gagne la partie, l'autre joueur perd.

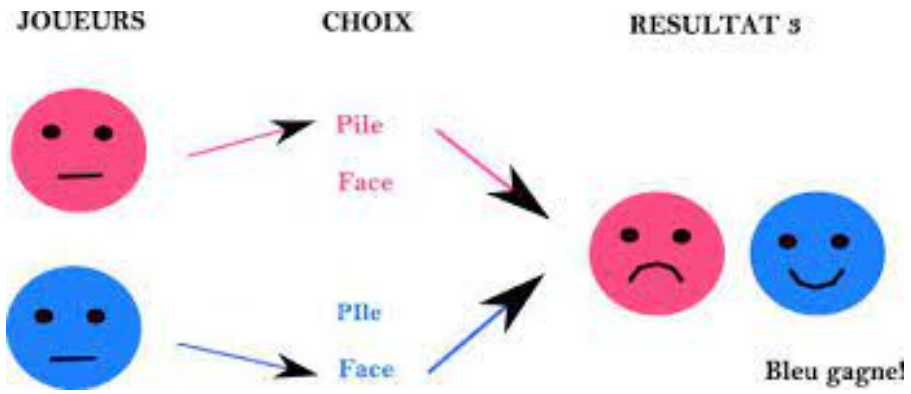


Fig. 1.5 – Modélisation du jeu pile ou face

Dilemme du prisonnier :

Deux suspects sont arrêtés par la police. Mais les agents n'ont pas assez de preuves pour les inculper, donc ils les interrogent séparément en leur faisant la même offre. « Si tu dénonces ton complice et qu'il ne te dénonce pas, tu seras remis en liberté et l'autre écopera de 10 ans de prison.
Si tu le dénonces et lui aussi, vous écoperez tous les deux de 5 ans de prison. Si personne ne dénonce l'autre, vous aurez tous deux 6 mois de prison. »

1 \ 2	Le suspect n° 2 se tait	Le suspect n° 2 dénonce
Le suspect n° 1 se tait	Les deux font 6 mois de prison	1 fait 10 ans de prison ; 2 est libre
Le suspect n° 1 dénonce	1 est libre ; 2 fait 10 ans de prison	Les deux font 5 ans de prison.

Fig. 1.6 – Modélisation du dilemme du prisonnier

Dilemme du voyageur :

Une compagnie aérienne perd deux valises appartenant à deux voyageurs différents. Les deux valises se trouvent être identiques et contiennent des antiquités identiques.
Un responsable de compagnie aérienne chargé de régler les réclamations des deux voyageurs explique que la compagnie aérienne est responsable d'un maximum de 100 USD par valise - il est incapable de connaître directement le prix des antiquités.

Pour déterminer une valeur estimée honnête des antiquités, le responsable sépare les deux voyageurs afin qu'ils ne puissent pas se concerter et leur 4 demande d'écrire le montant de leur valeur entre pas moins de 2 dollards et plus de 100 dollards. Il leur dit également que si les deux écrivent le même numéro, il traitera ce numéro comme la valeur réelle en dollars des deux valises et remboursera ce montant aux deux voyageurs.

Toutefois, si l'un écrit moins que l'autre, ce dernier sera considéré comme la valeur réelle et les deux voyageurs recevront ce montant avec un bonus / malus : 2 dollards supplémentaires seront versés au voyageur qui a écrit la valeur inférieure et une déduction de 2 dollards seront prélevées sur la personne qui a consigné le montant le plus élevé. Le défi est le suivant : quelle stratégie les deux voyageurs devraient-ils suivre pour décider de la valeur à écrire ?

Les deux joueurs tentent d'optimiser leurs gains, sans se soucier des gains de l'autre joueur.

1.4 Les principaux types de jeux

1.4.1 Jeu de potentiel

En théorie des jeux, un jeu de potentiel est un jeu où il existe une fonction globale décrivant les conséquences d'un changement de stratégie pour chaque joueur. Cette fonction est appelée **fonction de potentiel**. Cette fonction permet de garantir l'existence d'équilibres de Nash purs.
Ces jeux ont des propriétés intéressantes du point de vue des équilibre de Nash, et ont des applications, par exemple dans les jeux de congestion en théorie algorithmique des jeux, qui peuvent représenter le trafic routier.

1.4.2 Jeux satbles

De même que l'équilibre chimique est l'état dans lequel les réactifs et les produits n'ont plus tendance à changer avec le temps, un ensemble d'actions est considéré comme stable lorsqu'aucun joueur - ou, dans le cas des jeux coopératifs, un ensemble de joueurs - ne changerait son choix d'action s'il en avait l'occasion.

1.4.3 Jeux min-max

L algorithme MinMax est un algorithme qui s'applique à la théorie des jeux pour les jeux à deux joueurs à somme nulle (et à information complète) consistant à **minimiser la perte maximum** (c'est-à-dire dans le pire des cas).
Pour une vaste famille de jeux, le théorème du minimax de von Neumann assure l'existence d'un tel algorithme, même si dans la pratique il n'est souvent guère aisé de le trouver.
Le jeu de hex est un exemple où l'existence d'un tel algorithme est établie et montre que le premier joueur peut toujours gagner, sans pour autant que cette stratégie soit connue. Il amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son adversaire.

Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'adversaire cherche au contraire à les maximiser (le jeu est à somme nulle). Il existe différents algorithmes basés sur MinMax permettant d'optimiser la recherche du meilleur coup en limitant le nombre de nœuds visités dans l'arbre de jeu, le plus connu est l'élagage alpha-bêta.
En pratique, l'arbre est souvent trop vaste pour pouvoir être intégralement exploré (comme pour le jeu d'échecs). Seule une fraction de l'arbre est alors explorée.

On va noter par MAX l'agent qu'on cherche à faire gagner et son adversaire par MIN. Les deux joueurs désirent gagner le jeu. On suppose que le joueur MIN joue logiquement et qu'il ne va jamais rater une occasion de gagner. Si pour gagner le joueur MAX essaie de maximiser son score, le joueur MIN désire aussi maximiser son propre score (ou de minimiser le score du joueur MAX).[4]

L'algorithme MINIMAX, dû à Von Neumann, à comme but l'élaboration d'une stratégie optimale pour le joueur MAX.

À chaque tour le joueur MAX va choisir le coup qui va maximiser son score, tout en minimisant les bénéfices de l'adversaire. Ces bénéfices sont évalués en termes de la fonction d'évaluation statique utilisée pour apprécier les positions pendant le jeu.
Pour des raisons de temps de calcul et de mémoire, l'analyse est faite pour un horizon de jeu de P, le nombre de coups en avant (la profondeur de l'arbre de jeu examiné).

On a vu que l'évaluation statique des positions terminales de jeu est simple et précise : il faut apprécier selon les règles du jeu s'il s'agit d'une victoire, d'une défaite ou d'une égalité. Pour les postions intermédiaires cette fonction est imprécise, à cause des critères plus au moins subjectifs qui ont été utilisé pour l'évaluation. Le manque d'exactitude de la fonction d'évaluation statique est compensée par l'analyse rigoureuse des conséquences de chaque coup de deux joueurs.

L'algorithme réalise une évaluation de la position courante, représentée par la racine de l'arbre de jeu, en partant des nœuds terminaux. Dans ce but l'ensemble des nœuds de l'arbre de jeu est divisé en deux classes : les nœuds MAX sur les niveaux où ce joueur va choisir un coup et les nœuds MIN pour les niveaux de décision du joueur MIN.

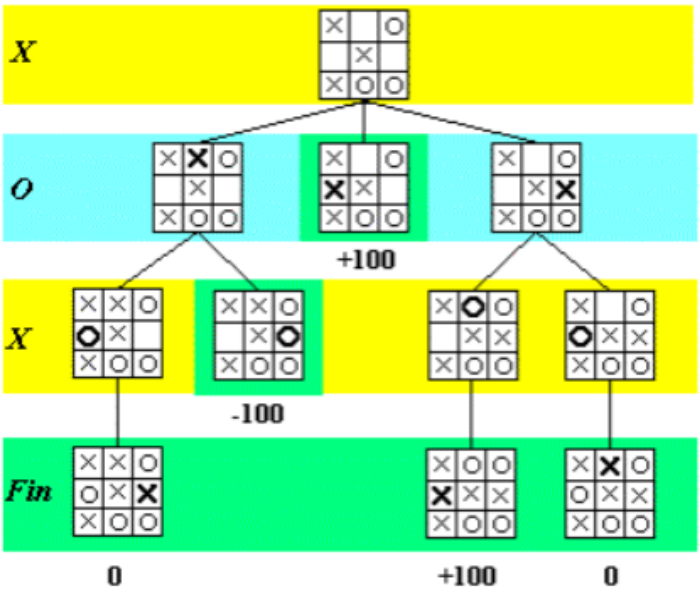


Fig. 1.7 – Modélisation de l'algorithme minimax par tic tac toe

1.5 Equilibre de Nash

1.5.1 Présentation

En théorie des jeux, un équilibre de Nash est une situation où :
1. Chaque joueur prévoit correctement le choix des autres ;
2. Chaque joueur maximise son gain, compte tenu de cette prévision.
Autrement dit, un profil de stratégie $S^* = (S_i^*)_{i \in K}$ est un équilibre de Nash si chaque joueur i joue une stratégie optimale S^* (qui maximise son gain u) compte tenu des stratégies des autres joueurs S^*

$$\forall (i,j) \in K * K, \forall s_i \in ((s_i^*)_{i \in K}), u(s_i^*, s_j^*) \geq u(s_i, s_j^*) \tag{1.1}$$

L'équilibre de Nash est donc tel qu'aucun joueur ne regrette son choix (il n'aurait pas pu faire mieux) au vu du choix des autres, les choix étant, comme toujours en théorie des jeux, simultanés. Souvent l'équilibre de Nash est présenté comme une situation où chacun adopte la meilleure réponse « compte tenu » du choix des autres, ce qui peut laisser croire que ce choix est connu — alors qu'il n'en est rien, pour des raisons évidentes.

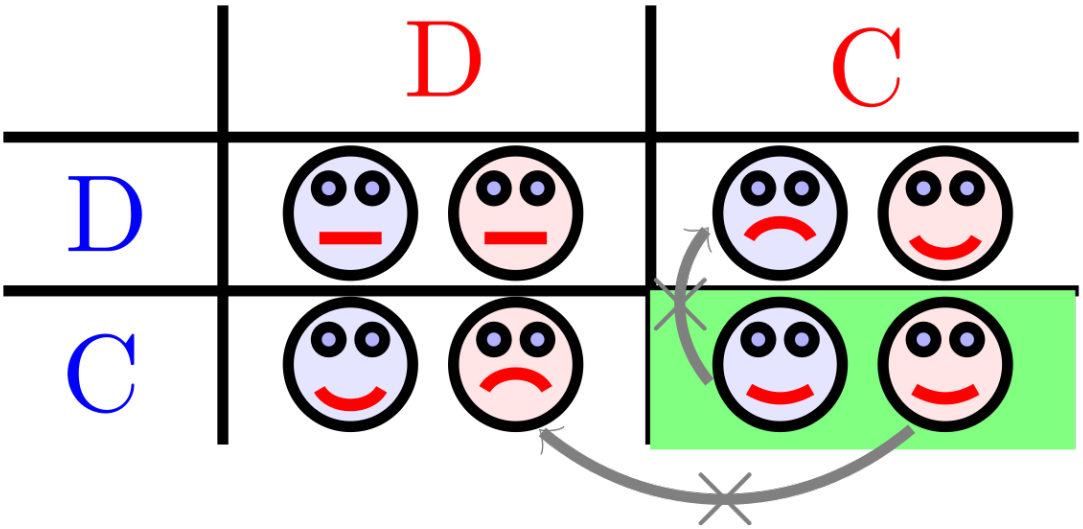


Fig. 1.8 – Modélisation de l'équilibre de Nash

1.5.2 Etude d'unicité

Tout jeu peut avoir de nombreux équilibres de Nash ou aucun (c'est le cas du jeu consistant à écrire simultanément un entier, le gagnant étant celui dont l'entier est le plus grand). Un autre jeu populaire est : Deviner 2/3 de la moyenne qui possède exactement un équilibre.
Néanmoins, Nash est parvenu à démontrer que tout jeu avec un nombre fini de joueurs ayant un nombre fini de stratégies admet au moins un équilibre de Nash en stratégie mixte (c'est-à-dire si

l'on considère comme une stratégie possible de tirer aléatoirement entre plusieurs stratégies, avec des probabilités fixées).

1.5.3 Etude d'unicité pour un problème de minimax

Soit k le nombre de stratégies pures disponibles pour le joueur 1 et n le nombre de stratégies pures disponibles pour le joueur 2. On numérote de 1 à k les stratégies à la disposition du J1 et de 1 à n celles du J2.
Prenons un exemple facile de jeu où n = k = 3.
Voici la matrice de gain :

J1/J2	joue sa stratégie 1	joue sa stratégie 2	joue sa stratégie 3
joue sa stratégie 1	-1	1,5	0
joue sa stratégie 2	2	2	1
joue sa stratégie 3	2	-3	0

Fig. 1.9 – Matrice de gain correspondante

Le joueur 1 se dit : « Si je joue 1, je risque de perdre 1 unité, et si je joue 3 d'en perdre 3 ; mais si je joue 2, je gagne une unité dans le pire des cas. »
Le joueur 2 se dit : « Si je joue 1, je risque de perdre 2 unités, et si je joue 2 d'en perdre 2 ; mais si je joue 3 mes pertes sont limitées à une unité dans le pire des cas.
Le joueur 1, reconstituant le raisonnement du joueur 2 à choisir sa stratégie 3 pour limiter ses pertes, est conforté dans son choix.
Le joueur 2, reconstituant le raisonnement du joueur 1 à choisir sa stratégie 2, est conforté dans son choix (il perdrait 2 s'il choisissait autrement).

Aucun des joueurs n'est prêt à changer sa stratégie car ce sont les choix les plus rationnels.

Par exemple, si le joueur 1 devient gourmand et change sa stratégie pour gagner 1,5 ou 3 unités, sachant que le joueur 2 joue intelligemment et donc sa stratégie 3, le joueur 1 ne gagnerait plus que 0 unité, au lieu de 1. Il n'a donc aucun intérêt à changer sa stratégie. Ce raisonnement est réciproque pour le joueur 2.
En effet, selon la proposition, **nous trouvons bien un équilibre :**

$$\max_{1\leq i\leq k}(\min_{1\leq i\leq n} a_{ij}) = max(1,-1,-3) = 1 \tag{1.2}$$

Tandis que

$$\min_{1\leq i\leq n}(\max_{1\leq i\leq k} a_{ij}) = min(1{,}1.5{,}2) = 1 \tag{1.3}$$

Le minimax et le maximin sont égaux ; la ligne où le minimax est atteint et la colonne où le maximin est atteint indiquent un point d'équilibre.

Chapitre 2

Implémentation de l'algorithme de Réseaux Génératifs Antagonistes (GAN)

Les Réseaux Antagonistes Génératifs (GAN) sont un nouveau type de réseau de neurones profonds (deeplearning). Leur objectif est de pouvoir estimer et représenter une distribution de données.[5]

Ce type de réseau a été introduit par Ian Goodfellow en 2014 .

Il présente un framework composé de deux modèles : un générateur et un discriminateur. Le générateur (que nous appellerons G) a pour objectif de produire des données artificielles semblables aux données réelles.
Le discriminateur (que nous appellerons D) a pour objectif de différencier les données générées par le générateur G des données d'apprentissage.

Les GANs sont des réseaux utilisant un apprentissage non supervisé. C'est-à-dire que l'apprentissage du réseau se fait sans étiquette. De ce fait, leur apprentissage nécessite uniquement un dataset contenant les données dont nous voulons apprendre la distribution.

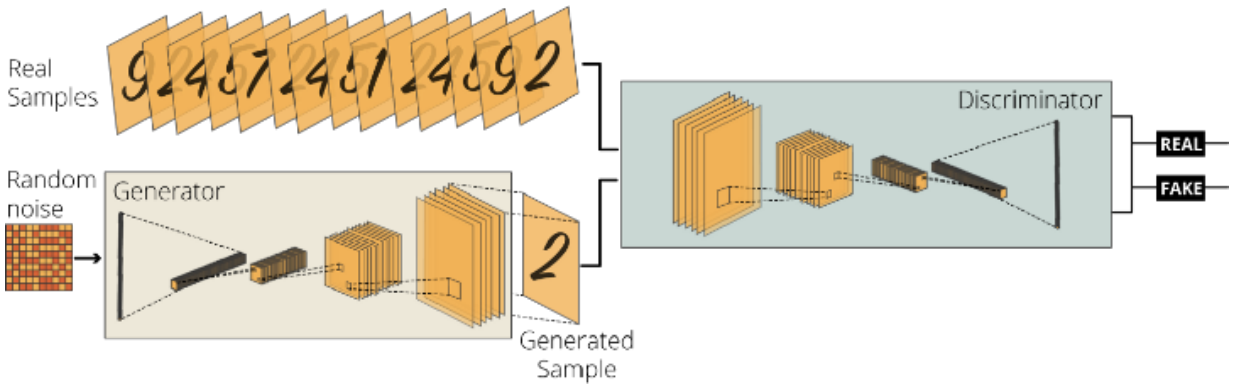


Fig. 2.1 – Modélisation de Réseaux Génératifs Antagonistes (GAN)

2.1 Importation des bibliothèques nécessaires

On commnce par importer les bibliothèques dont on aura besoin d'utiliser :

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import transforms
import numpy as np
import matplotlib.pyplot as plt
```

2.2 Implémentation du générateur

On crée la classe generator en lui donnant ses attributs et ses méthodes :

```
class generator(nn.Module):
    def __init__(self):
        super(generator, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(128, 1024),
            nn.ReLU(),
            nn.Linear(1024, 1024),
            nn.ReLU(),
            nn.Linear(1024, 784),
            nn.Tanh()
        )

    def forward(self, input):
        return self.main(input)
```

2.3 Implémentation du discriminateur

On crée la classe discriminator en lui donnant ses attributs et ses méthodes :

```
class discriminator(nn.Module):
    def __init__(self):
        super(discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(784, 256),
            nn.LeakyReLU(0.2),
```

```
            nn.Linear(256, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

2.4 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.

On importe donc la dataset et on effectue les opérations d'apprentissage sur le principe suivant :
- **Réseau génératif** : essaie de confondre le réseau discriminatif
- **Réseau discriminatif** (critique) : essaie de distinguer entre les images réels et les fausses
On ne cherche pas à modéliser explicitement la densité (manifold) mais notre approche inspirée de la théorie des jeux : **jeu minimax à 2 joueurs** [6]

$$\min_G \max_D E_{x^* \in Data}[logD(x^*)] + E_{z\ p(z)}[log(1 - D(G(z)))] \tag{2.1}$$

D'ou le code suivant appliqué à l'ensemble des images des numéros :

```
plt.rcParams['image.cmap'] = 'gray'
def show_images(images):
    sqrtn = int(np.ceil(np.sqrt(images.shape[0])))

    for index, image in enumerate(images):
        plt.subplot(sqrtn, sqrtn, index+1)
        plt.imshow(image.reshape(28, 28))
def d_loss_function(inputs, targets):
    return nn.BCELoss()(inputs, targets)


def g_loss_function(inputs):
    targets = torch.ones([inputs.shape[0], 1])
    targets = targets.to(device)
    return nn.BCELoss()(inputs, targets)
```

```
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
print('GPU State:', device)

# Model
G = generator().to(device)
D = discriminator().to(device)
print(G)
print(D)

# Settings
epochs = 200
lr = 0.0002
batch_size = 64
g_optimizer = optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
d_optimizer = optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))


# Transform
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])


# Load data
train_set = datasets.MNIST('mnist/', train=True, download=True, transform=transform)
test_set = datasets.MNIST('mnist/', train=False, download=True, transform=transform)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False)
for epoch in range(epochs):
    epoch += 1

    for times, data in enumerate(train_loader):
        times += 1
        real_inputs = data[0].to(device)
        test = 255 * (0.5 * real_inputs[0] + 0.5)

        real_inputs = real_inputs.view(-1, 784)
        real_outputs = D(real_inputs)
        real_label = torch.ones(real_inputs.shape[0], 1).to(device)

        noise = (torch.rand(real_inputs.shape[0], 128) - 0.5) / 0.5
        noise = noise.to(device)
```

```
        fake_inputs = G(noise)
        fake_outputs = D(fake_inputs)
        fake_label = torch.zeros(fake_inputs.shape[0], 1).to(device)

        outputs = torch.cat((real_outputs, fake_outputs), 0)
        targets = torch.cat((real_label, fake_label), 0)

        # Zero the parameter gradients
        d_optimizer.zero_grad()

        # Backward propagation
        d_loss = d_loss_function(outputs, targets)
        d_loss.backward()
        d_optimizer.step()

        # Generator
        noise = (torch.rand(real_inputs.shape[0], 128)-0.5)/0.5
        noise = noise.to(device)

        fake_inputs = G(noise)
        fake_outputs = D(fake_inputs)

        g_loss = g_loss_function(fake_outputs)
        g_optimizer.zero_grad()
        g_loss.backward()
        g_optimizer.step()

        if times % 100 == 0 or times == len(train_loader):
            print('[{}/{}, {}/{}] D_loss: {:.3f} G_loss: {:.3f}'.format(epoch, epochs, tim

    imgs_numpy = (fake_inputs.data.cpu().numpy()+1.0)/2.0
    show_images(imgs_numpy[:16])
    plt.show()

    if epoch % 50 == 0:
        torch.save(G, 'Generator_epoch_{}.pth'.format(epoch))
        print('Model saved.')


print('Training Finished.')
```

2.5 Interprétations

Dans cette partie, on expliquera l'implémentation du modèle GAN dans le cadre de la théorie des jeux par les questions données en support de l'atelier :

2.5.1 Définir le jeu utilisé dans cette partie

Comme déja expliqué aupparavant, il s'agit d'un jeu non coopératif, non convexe implique un certain nombre de joueurs ayant des intérêts totalement ou partiellement contradictoires dans le résultat d'un processus de décision.
Dans notre cas les deux joueurs sont **le générateur et le discriminateur**.
Le générateur essaye de produire des images non réelles et trompe le discriminateur alors que le discriminateur doit distinguer les données réelles des données non réelles produits par le générateur.
Ou encore :
- Le premier joueur s'appelle un générateur et il vise à générer de nouvelles données similaires à celles attendues. Le Générateur pourrait être assimilé à un faussaire d'art humain, qui crée de fausses œuvres d'art.
- Le deuxième joueur est nommé le discriminateur. L'objectif de ce modèle est de reconnaître si une donnée d'entrée est « réelle » ou si elle est « falsifiée ». Dans ce scénario, un discriminateur est analogue à un expert en art, qui essaie de détecter des œuvres d'art comme véridiques ou frauduleuses.

2.5.2 En déduire pourquoi les GANs peuvent être modélisés par des problèmes min-max

La perte Minimax GAN fait référence à l'optimisation simultanée minimax des modèles de discriminateur et de générateur. Minimax fait référence à une stratégie d'optimisation dans les jeux au tour par tour à deux joueurs pour minimiser la perte ou le coût pour le pire des cas de l'autre joueur.
Pour le GAN, le générateur et le discriminateur sont les deux acteurs et impliquent à tour de rôle des mises à jour de leurs poids de modèle. Le min et le max font référence à la minimisation de la perte du générateur et à la maximisation de la perte du discriminateur.

2.5.3 Interpréter l'équation (2.1) par rapport au jeu déjà mentionné

Comme indiqué ci-dessus, le discriminateur cherche à maximiser la moyenne du log de probabilité des images réelles et le Log de la probabilité inverse des fausses images.
Discriminateur : maximiser log D(x) + log (1 – D(G(z)))

Le générateur cherche à minimiser le log de la probabilité inverse prédite par le discriminateur pour les fausses images. Cela a pour effet d'encourager le générateur à générer des échantillons

qui ont une faible probabilité d'être faux.
Générateur : minimiser log (1 – D(G(z)))

D'ou la fonction conjointe à optimiser :

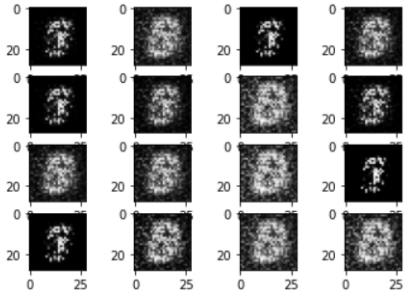
$$\min_G \max_D E_{x^* \in Data}[logD(x^*)] + E_{z\ p(z)}[log(1 - D(G(z)))] \tag{2.2}$$

2.5.4 Tester les deux blocs : générateur et discriminateur pour un ensemble d'images, en précisant à chaque fois les paramètres utilisés

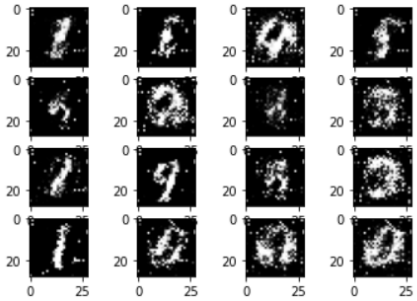
Lors de notre étude, on a fixé le pas de l'algorithme de gradient descendant (learning rate) à 0.0002 et on fait à chaque fois tracer la figure à chaque itération.

On obtient les figures suivantes :

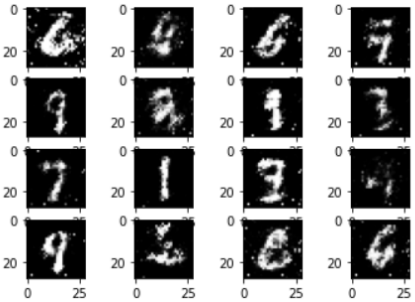
1ère itration



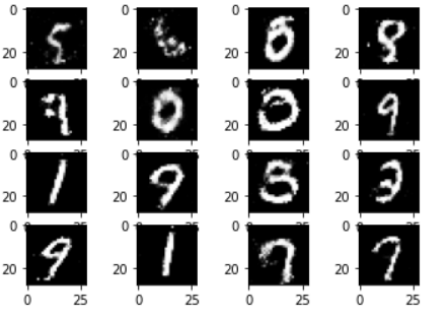
5ème itération



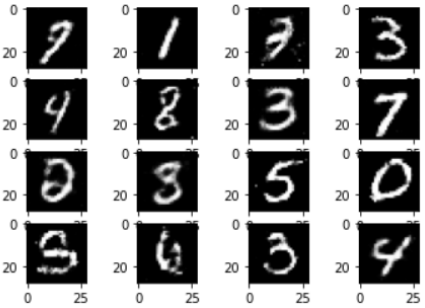
10ème itération



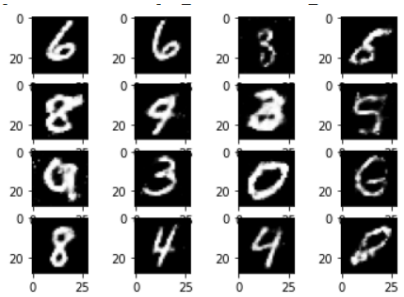
20èmee itration



50ème itération



100ème itération



on peut remarquer que l'erreur du discriminateur est très élevée au début, car il ne sait pas comment classer correctement les images comme étant réelles ou fausses. Au fur et à mesure que le discriminateur s'améliore et que son erreur diminue, l'erreur du générateur augmente, ce qui prouve que le discriminateur surpasse le générateur et qu'il peut classer correctement les faux échantillons. Au fur et à mesure que le temps passe et que l'entraînement se poursuit, l'erreur du générateur diminue, ce qui implique que les images qu'il génère sont de mieux en mieux.
Alors que Le générateur s'améliore, l'erreur du discriminateur augmente, car les images synthétiques deviennent à chaque fois plus réalistes.

On conclut que cette compétition entre nos 2 joueurs nous permet d'amériorer les performances de notre réseau en termes de génération d'images et de détection des imperfections de moment ou chaque joueur "fait de son mieux" pour gagner.

Conclusion

Nous avons réussi dans ce travail à se familiariser avec le concept de théorie des jeux et de l'explorer dans le cadre d'une implémentation appliquée dans l'étude des données.

En effet, le travail de documentation que nous avons fait nous a permis de faire face à la problématique de définir les différents types de jeux et les conditions d'équilibre de Nash (ses conditions d'existance et d'unicité) et puis d'implémenter l'exemple de l'Algorithme de Réseaux Génératifs Antagonistes (GAN) basé sur la Théorie des jeux et de discuter les fonctionnalités de ce jeu entre le générateur et le discriminateur.

Pour conclure, les GANs permettent d'économiser du temps humain puisqu'ils relèvent de l'apprentissage non-supervisé, ou d'éviter des tâches fastidieuses, comme l'étiquetage manuel des données à partir desquelles apprennent les algorithmes d'apprentissage supervisé .
Par contre, les GANs n'ont toutefois rien d'une solution miracle : ils sont difficiles à entraîner. Il arrive que **l'algorithme ne converge jamais (au sens mathématique) au cours de son apprentissage**, ce qui mène à l'impossibilité d'aboutir à des prédictions exploitables.Il sont particulièrement adaptés aux images, puisqu'ils sont plus difficiles à utiliser pour produire des données discrètes, par exemple du texte.[7]

Ainsi, cet apparent paradoxe : lorsque les GANs fonctionnent, ils fonctionnent très bien. Mais lorsqu'ils fonctionnent mal, les informaticiens peinent à identifier sur quels leviers agir... et en sont réduits à cette approche toute humaine : **le tâtonnement**

Souheib Ben Mabrouk.
Hassen Chtara.

Bibliographie

[1] Théorie des jeux. https://www.universalis.fr/encyclopedie/theorie-des-jeux/.

[2] Centrale Marseille. Game theory. http://renaud.bourles.perso.centrale-marseille.fr/Cours/Theorie_des_jeux.pdf.

[3] wikepidia : Théorie des jeux. https://fr.wikipedia.org/wiki/ThéA9orie_des_jeux.

[4] Algorithme minimax. https://turing.cs.pub.ro/auf2/html/chapters/chapter3/chapter_3_4_2.html.

[5] Gan : Vers une meilleure estimation des distributions ? https://www.aquiladata.fr/insights/gan-vers-une-meilleure-estimation-des-distributions/.

[6] Gan and game theory. http://www2.ift.ulaval.ca/~pgiguere/cours/DeepLearning/11-GANs.pdf.

[7] Comment les gans révolutionnent en profondeur l'intelligence artificielle ? https://www.heidi.news/sciences-climat/les-gan-nouvelle-technologie-star-en-intelligence-artificielle#:~:text=Les%20GANs%20ne%20sont%20qu,contrefa%C3%A7on%20la%20plus%20r%C3%A9aliste%20possible.