SUJET 12 – THEORIE DES JEUX ET MACHINE LEARNING

Badr LYAZIL et Joaquin MENENDEZ

GRP – 04 FINANCE

Sommaire

I. Présentation du sujet

1) THEORIE DES JEUX

La théorie des jeux est l'optimisation de l'utilisation de ressources limitées en une série d'événements qui illustrent comment ce problème détermine l'interaction humaine. On peut l'utiliser pour mieux comprendre les modèles économiques (au sens non financier du terme), mais veux pas dire que nous rend plus efficaces dans le monde des affaires ou notre vie personnelle. Le fait que beaucoup des modèles sociaux se retrouvent dans toute la nature nous rappelle que les sociétés les plus complexes sont basées sur des comportements et des choix qui ne sont pas si éloignés du monde naturel.

Afin d'introduire la théorie des jeux, nous devons décrire les concepts de base de la théorie des jeux.

Information complète et incomplete :

Sont des termes largement utilisés, on dit qu'il y a information complète lorsque chaque agent connait la fonction d'utilité de l'autre agent et les règles du jeu. « Chaque jouer est pleinement conscient des règles du jeu et des fonctions d'utilité de chacun des joueurs ». Mais ça ne veut pas dire qu'il a une connaissance commune où chaque joueur est conscient que les autres joueurs connaissent les règles et chaque fonction d'utilité.

Connaissance commune (Common Knowledge) :

La connaissance commune est une condition habituellement requise en théorie des jeux. Exige que tous les joueurs doivent être conscients de la conscience des autres joueurs concernant les règles et les fonctions d'utilité de chacun.

Perfect and imperfect information :

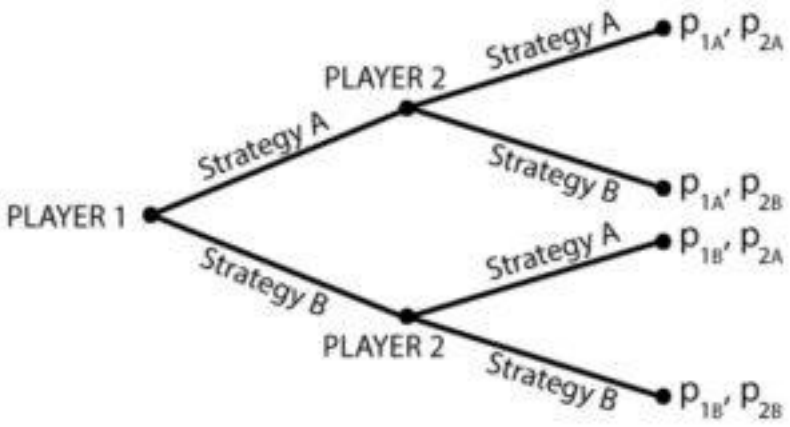
L'information parfaite est une notion importante en théorie des jeux l'lorsqu'on considère les jeux séquentiels et simultanés. Il s'agit d'un concept clé lorsqu'on analyse la possibilité de stratégies de punition dans les accords de collusion. Chaque joueur dispose des mêmes informations que celles qui seraient disponibles à la fin du jeu. Chaque joueur connait ou peut voir les mouvements des autres joueurs comme par exemple dans les échecs.

L'information imparfaite apparait lorsque des décisions doivent être prises simultanément, et que les joueurs doivent mettre en balance tous les résultats possibles lorsqu'ils prennent une décision. Un exemple est un jeu des cartes où les cartes de chaque joueur sont cachées au reste des joueurs.

Information	Perfect -knows full history /decision	**Imperfect:** Simultaneous games
Complete: Knows the rules, **payoff**	Tennis, soccer (amateur-level) Not interesting game! Just select an obvious best!	Rock - Paper – Scissors Tennis, soccer (pro-level) Sealed bid auction
Incomplete	Price negotiation of used cars	Hiring talents

Deux des façons de représenter un "jeu" sont la forme extensive et la forme stratégique.

La forme extensive consiste à décrire un jeu à l'aide d'un arbre de jeu. On utilise un arbre pour montrer que les choix sont faits à différents moments. Elle est généralement utilisée pour décrire des jeux séquentiels grâce à la représentation des décisions à différents moments. Les jeux séquentiels impliquent de prendre des décisions à différents moments pour chaque joueur, l'information est parfaite puisque chaque joueur peut voir la décision prise par le joueur précédent et les règles du jeu sont connus par tous.



Par contre la forme stratégique est une façon de décrire un jeu à l'aide d'une matrice. Le jeu est défini en exposant de chaque côté de la matrice les différents joueurs, chaque stratégie ou choix qu'ils peuvent faire et les ensembles de gains qu'ils recevront chacun pour une stratégie donnée

		PLAYER 2	
		Strategy A	Strategy B
PLAYER 1	Strategy A	p_{1A}, p_{2A}	p_{1A}, p_{2B}
	Strategy B	p_{1B}, p_{2A}	p_{1B}, p_{2B}

On utilise souvent la forme stratégique pour les jeux simultanés, où les deux joueurs choisissent simultanément, par opposition aux jeux séquentiels pour lesquels il est préférable de décrire le jeu en utilisant la forme extensive. Les jeux simultanés ont des informations complètes et imparfaites, et que les règles du jeu ainsi que les gains de chaque joueur sont connues par tous.

Nash Equilibrium :

Les équilibres de Nash sont définis comme la combinaison de stratégies dans un jeu de telle sorte que les joueurs ne sont pas incités à dévier de leur choix. Il s'agit de la meilleure option qu'un joueur puisse faire, en tenant compte de la décision des autres joueurs et où un changement dans la décision d'un joueur ne conduira qu'à un résultat pire si les autres joueurs s'en tiennent à leur stratégie. L'un des équilibres de Nash les plus connus se trouve dans le dilemme du prisonnier. Ce concept appartient à la théorie des jeux, plus précisément aux jeux non coopératifs, et a été nommé d'après John Nash qui l'a développé.

Il existe quelques exigences de cohérence qui doivent être prises en compte lorsqu'on traite des équilibres de Nash. L'une d'entre elles est connue sous le nom de connaissance commune, qui étend la nécessité d'une information complète. Par conséquent, les attentes concernant les stratégies des autres joueurs doivent être rationnelles.

Un équilibre de Nash est donc une combinaison de croyances sur les probabilités des stratégies et les choix de l'autre joueur. Il est assez facile de comprendre cela à l'aide d'un exemple, en l'occurrence le dilemme du prisonnier tel qu'il est représenté dans la matrice de jeu ci-contre.

		PRISONER 2	
		Confess	Lie
PRISONER 1	Confess	-8, -8	0, -10
	Lie	-10, 0	-1, -1

Le prisonnier 1 (P1) doit se faire une idée du choix que P2 va faire, afin de choisir la meilleure stratégie. Si P2 avoue (P2C), il obtiendra soit -8 soit 0, et s'il ment (P2L), il obtiendra soit -10 soit -1. On peut facilement voir que P2 choisira d'avouer, puisqu'il sera mieux. Par conséquent, P1 doit choisir la meilleure stratégie étant donné que P2 choisira d'avouer : P1 peut soit avouer (P1C, qui rapporte -8), soit mentir (P1L, qui rapporte -10). La chose rationnelle à faire pour P1 est d'avouer. En procédant de manière inverse, nous analysons les croyances de P2 sur les stratégies de P1, ce qui nous amène au même point : la chose rationnelle à faire pour P2 est d'avouer. Par conséquent, [P1C, P2C] est l'équilibre de Nash de ce jeu (souligné en rouge).

2) MACHINE LEARNING

Tout d'abord, il ne faut pas confondre intelligence artificielle et machine learning. En effet, il y a une nuance. Le rôle de l'intelligence artificielle est de concevoir des machines capables d'imiter l'intelligence humaine. Le machine learning, lui, permet aux ordinateurs d'utiliser les expériences passées, des données passées et statistiques pour apprendre à exécuter des tâches pour lesquelles ils ne sont pas programmés à la base.

Historiquement, ce concept de machine learning a été initié par Alan Turing en 1936 avec « la machine universelle ». Le neurophysiologiste Warren McCulloch et le mathématicien Walter Pitts sont, eux, à la base théorique du concept des réseaux neuronaux, grâce à leur article en 1943.

En machine learning, il existe trois catégories. Il y a le « Supervised learning », le « Unsupervised learning » et « Reinforcement learning ».

Le Supervised learning (apprentissage supervisé), c'est le cas où l'on fournit à la machine un ensemble de données avec leurs solutions. Une fois que la machine est « entrainée », c'est-à-dire une fois que l'ordinateur a pris en compte toute les données et leurs solutions, alors ce dernier sera capable de donner la solution d'une donnée non entrée au préalable. Pour ce cas, il existe deux types de classification. En effet, il y a la classification linéaire et la non-linéaire. Il y a une différence entre les problèmes de régression et les problèmes de classification. En effet, les problèmes de prédiction d'une variable quantitative sont des problèmes de régression alors que les problèmes permettant de prédire 'une variable qualitative sont, quant à eux, des problèmes de classification.

Examples for supervised learning (1): classification

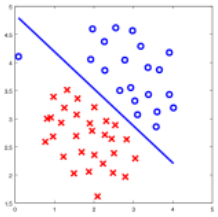


Figure: Linear classification

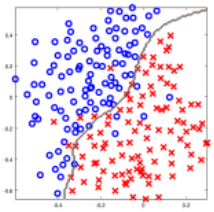
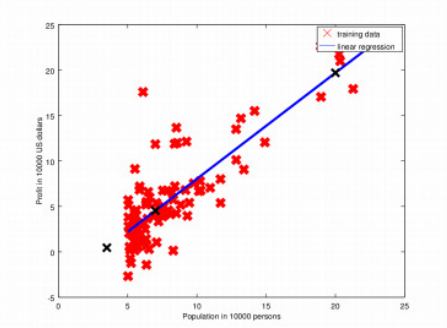


Figure: Nonlinear classification

Examples for supervised learning (2): regression



Le Unsupervised learning (apprentissage non-supervisé) c'est lorsque l'on intègre à la machine des données mais sans leurs solutions. Ce type d'algorithme ne sait pas au départ ce que représente les données que l'on lui a fourni. En effet, on dit que les données ne sont pas étiquetées, ne sont pas labélisées. Pour résoudre les problèmes non supervisés, la technique qui est utilisée est le regroupement ou le clustering. En effet, il s'agit de séparer, de répartir l'ensemble des données en différents sous-groupes de données se ressemblant le plus.

Examples of unsupervised learning (1)

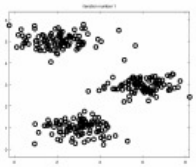


Figure: Unlabeled data

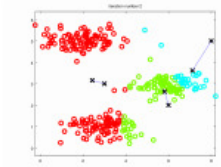


Figure: 3-means (1 iteration)

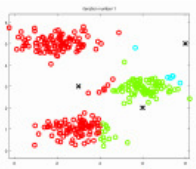


Figure: 3-means (initial)

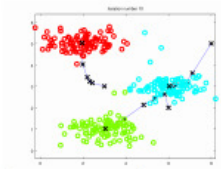


Figure: 3-means (10 iterations)

Le Reinforcement learning (apprentissage par renforcement) consiste pour un agent autonome (ex : robot) a apprendre comment se comporter, comment exécuter certaines tâches dans un environnement, à partir d'expériences passées, afin d'optimiser, de maximiser des noti ons de récompenses quantitative au fil du temps.

En bref, le machine learning, grâce aux mathématiques et aux statistiques, permet à des machines, à des ordinateurs, à partir de données, d'apprendre à exécuter des tâches pour lesquelles ils n'ont pas été programmés. En d'autres termes, le machine learning concerne la conception, l'analyse, l'optimisation, le développement et l'implémentation de ces méthodes qui permettent aux ordinateurs d'être entrainés par des données préalables afin d'effectuer des nouve lles actions. Un des aspects les plus importants du machine learning, c'est justement ces problèmes d'optimisation que nous verrons plus en détail dans la suite du rapport.

II. Partie théorique

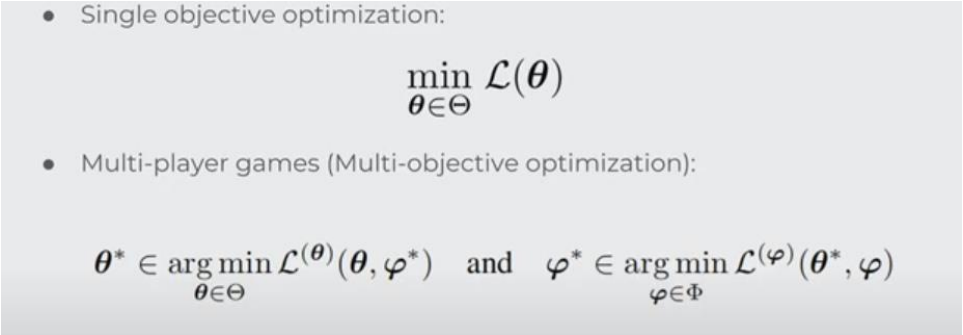
Depuis quelques années, deux branches de l'IA, le deep-learning et la théorie des jeux, semblent se rapprocher de plus en plus :

•On peut, par exemple, parler de l'intégration des jeux au machine learning qui a permis de repousser des limites (GANs). En effet, elle semblerait valider le passage progressif de problèmes d'optimisations multiples à ceux plus généraux de minimisations de pertes

•On peut, aussi, évoquer les récents progrès dans l'appréhension des jeux à informations imparfaite qui est largement renforcé par l'adjonction du deep learning dans les moteurs de résolution (Deep Stack)

1) GANs

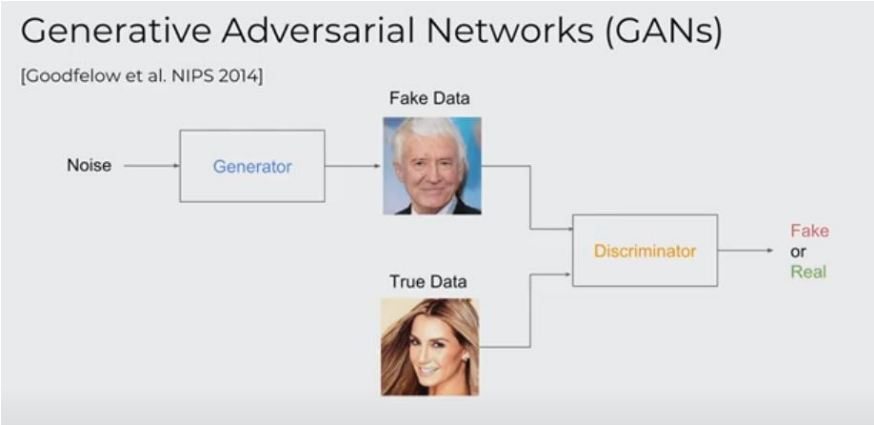
Tout d'abord, il faut différencier les problèmes d'optimisation mono-objectif et les problèmes d'optimisation multiobjectif. En effet, le but dans les problèmes à un seul objectif est d'obtenir le paramètre permettant de minimiser la fonction coût, les pertes en bref. Les problèmes d'optimisation à plusieurs objectifs peuvent, quant à eux, être considérés comme des jeux avec plusieurs joueurs. Par exemple, pour un problème à deux objectifs, on peut considérer que c'est un jeu avec deux joueurs, où chaque joueur contrôle un paramètre.



Le but du jeu est de trouver un équilibre de Nash, c'est-à-dire une situation où chaque joueur, en prévoyant correctement le choix de l'autre, maximise son gain, et donc minimise sa fonction coût. On peut parler des jeux à somme nulle où, par exemple, la fonction coût du premier joueur est égale à moins la fonction coût du deuxième joueur. On peut alors interpréter ce problème comme un problème de min max. En effet, on cherche le paramètre qui minimise la première fonction coût et le paramètre qui maximise la deuxième. On appelle cela le « Saddle point » (le point col). C'est là où réside la différence entre les problème d'optimisation mono-objectif et multiobjectif. En effet, dans les problèmes à plusieurs objectifs, donc les problèmes min max, on cherche à converger vers ce point col, tandis que pour les problèmes à un seul objectif on cherche à les éviter.

On va, désormais, parler des GANs (Generative Adversarial Networks = Réseaux antagonistes alternatifs) et du lien que l'on peut faire entre la résolution de problème min max et ces derniers. Un GAN peut être considéré comme un jeu entre deux joueurs. Le premier joueur est appelé le générateur et son but est de générer, comme son nom l'indique, une fausse image à partir du « noise » (le bruit). Dans ce GAN, il y a un autre un autre élément, le discriminateur. Le

discriminateur va recevoir la fausse image du générateur, et une vraie image. Son but sera de détecter quelle est la vraie image et quelle est la fausse. C'est modélisable par un jeu avec deux joueurs, le discriminateur et le générateur. Mathématiquement, le but du générateur est de maximiser la probabilité logarithmique d'une mauvaise classification de la fausse image générée (que le discriminant ne détecte pas que c'est une fausse image). Le but du discriminateur est, quant à lui, de maximiser la probabilité logarithmique d'une bonne classification des données qu'on lui fournit. On est donc bien face à un problème min max.



Si l'on résume, ces problèmes min max, ces problèmes d'optimisation multiobjectif, sont modélisables par des jeux multi-joueurs. On remarque, aussi, que ces problèmes peuvent, finalement, également, être considérés comme des problèmes avec des inéquations variationnelles.

2) Deepstack

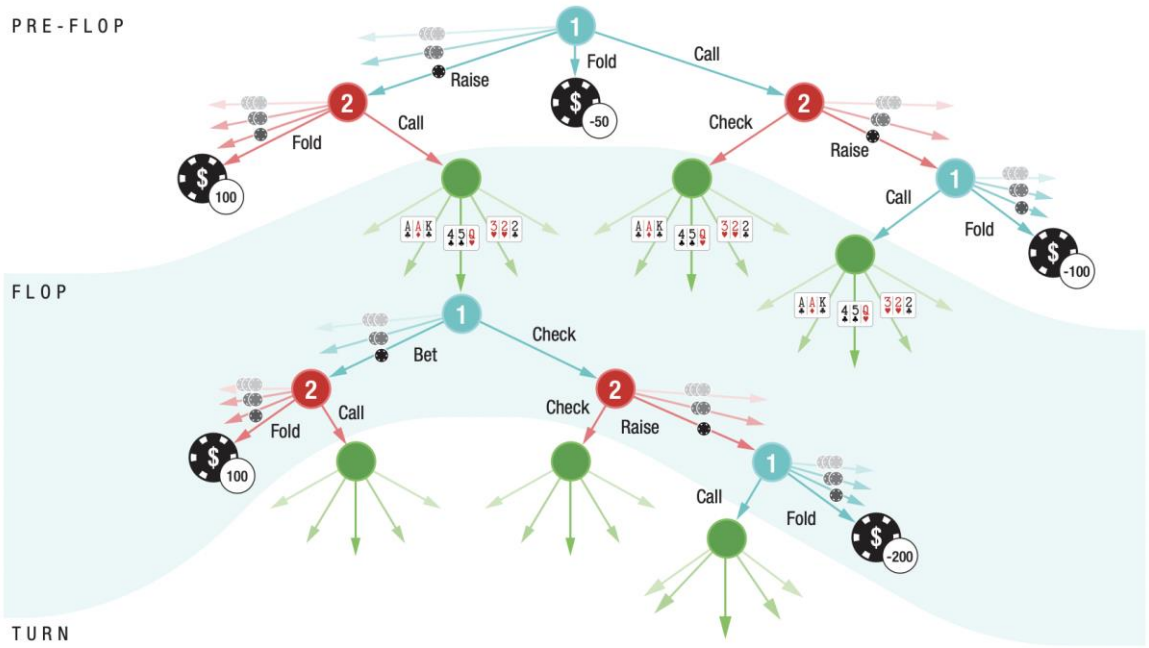
L'intelligence artificielle a avancé beaucoup ces dernières années et les jeux servant pour les essayer. Mais les jeux où on utilise souvent l'IA ont comme caractéristique commune qu'ils disposent d'une information parfaite. Comme par exemple le backgammon, checkers, chess, Jeopardy ! Atari video games etc. Le Poker par contre dispose des informations imparfaites et constitue un défi pour l'IA. DeepStack est un algorithme pour les situations d'information imparfaite.

Il combine le raisonnement récursif, c'est-à-dire un algorithme qui résout un problème en calculant des solutions d'instances plus petites du même problème, pour gérer l'asymétrie d'information, la décomposition pour concentrer le calcul sur la décision pertinente, et une forme d'intuition qui est automatiquement apprise à partir de l'auto-jeu en utilisant deep learning. Dans une étude portant sur 44 000 mains de poker, DeepStack a battu, avec une signification statistique, des joueurs de poker professionnels dans un heads-up no-limit Texas hold'em. L'approche est solide sur le plan théorique et il est démontré qu'elle produit des stratégies plus difficiles à exploiter que les approches précédentes.

Le Heads-up no-limit Texas hold'em (HUNL) est une version du poker à deux joueurs dans laquelle deux cartes sont initialement distribuées face cachée à chaque joueur, et des cartes supplémentaires sont distribuées face visible au cours des trois tours suivants. Aucune limite n'est imposée à la taille des mises, bien qu'il y ait une limite globale au montant total misé dans chaque partie. Les techniques d'IA ont déjà fait leurs preuves dans le jeu plus simple du Texas hold'em limite heads-up, où toutes les mises sont d'un montant fixe, ce qui donne un peu moins de 10^{14} points de décision. En comparaison, les ordinateurs ont dépassé les performances des experts humains au go, un jeu à information parfaite avec environ 10^{170} points de décision. Le jeu d'information imparfaite HUNL est comparable en taille au go, le nombre de points de décision dépassant 10^{160}.

Les jeux à information imparfaite nécessitent un raisonnement plus complexe que les jeux à information parfaite de taille similaire. La décision correcte à un moment donné dépend de la distribution de probabilité sur les informations privées que détient l'adversaire, qui sont révélées par ses actions passées. Cependant, la façon dont les actions de l'adversaire révèlent ces informations dépend de sa connaissance de nos informations privées et de la façon dont nos actions les révèlent. Ce type de raisonnement récursif est la raison pour laquelle on ne peut pas facilement raisonner sur les situations de jeu de manière isolée, ce qui est au cœur des méthodes de recherche heuristique pour les jeux à information parfaite. Les approches d'IA compétitives dans les jeux à information imparfaite raisonnent généralement sur l'ensemble du jeu et produisent une stratégie complète

avant de jouer. La minimisation du regret contrefactuel (CFR) est une de ces techniques qui utilise l'auto-jeu pour effectuer un raisonnement récursif en adaptant sa stratégie contre elle-même au cours d'itérations successives. Si le jeu est trop grand pour être résolu directement, la réponse commune est de résoudre un jeu plus petit, abstrait. Pour jouer au jeu original, on traduit les situations et les actions du jeu original au jeu abstrait.



III. Partie technique

1) GAN

Pour le code, nous avons utilisé un programme sous Python pour implémenter un GAN. Nous nous sommes servis de la bibliothèque tensorflow qui est un outil très utilisé dans le machine learning et le deep learning. Notre GAN est censé générer des chiffres entre 0 et 9.

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import os


def sample_Z(m, n):
    return np.random.uniform(-1., 1., size=[m, n])

def plot(samples):
    fig = plt.figure(figsize=(4, 4))
    gs = gridspec.GridSpec(4, 4)
    gs.update(wspace=0.05, hspace=0.05)

    for i, sample in enumerate(samples):
        ax = plt.subplot(gs[i])
        plt.axis('off')
        ax.set_xticklabels([])
        ax.set_yticklabels([])
        ax.set_aspect('equal')
        plt.imshow(sample.reshape(28, 28), cmap='Greys_r')

    return fig
```

Nous avons, ensuite, X qui est l'image que l'on va fournir au discriminant et Z qui est le « bruit » que l'on va fournir au générateur pour qu'il puisse générer la fausse image.

```
X = tf.placeholder(tf.float32, shape=[None, 784])


Z = tf.placeholder(tf.float32, shape=[None, 100])
```

On implémente, ensuite, les fonctions générateur et discriminateur.

```
def generator(z):
    with tf.variable_scope("generator", reuse=tf.AUTO_REUSE):
        x = tf.layers.dense(z, 128, activation=tf.nn.relu)
        x = tf.layers.dense(z, 784)
        x = tf.nn.sigmoid(x)
    return x

def discriminator(x):
    with tf.variable_scope("discriminator", reuse=tf.AUTO_REUSE):
        x = tf.layers.dense(x, 128, activation=tf.nn.relu)
        x = tf.layers.dense(x, 1)
        x = tf.nn.sigmoid(x)
    return x
```

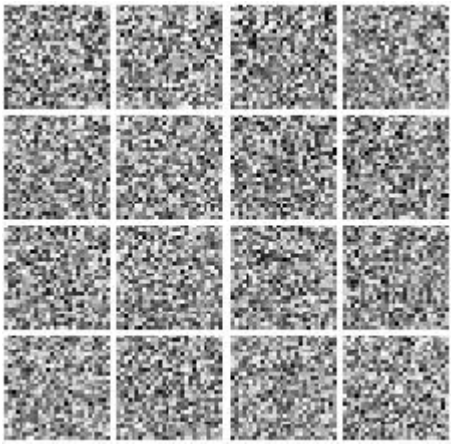
On crée ensuite les fonctions coûts/pertes et les optimiseurs.

```
G_sample = generator(Z)


D_real = discriminator(X)
D_fake = discriminator(G_sample)



D_loss = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
G_loss = -tf.reduce_mean(tf.log(D_fake))


disc_vars = [var for var in tf.trainable_variables() if var.name.startswith("disc")]
gen_vars = [var for var in tf.trainable_variables() if var.name.startswith("gen")]


D_solver = tf.train.AdamOptimizer().minimize(D_loss, var_list=disc_vars)
G_solver = tf.train.AdamOptimizer().minimize(G_loss, var_list=gen_vars)
```

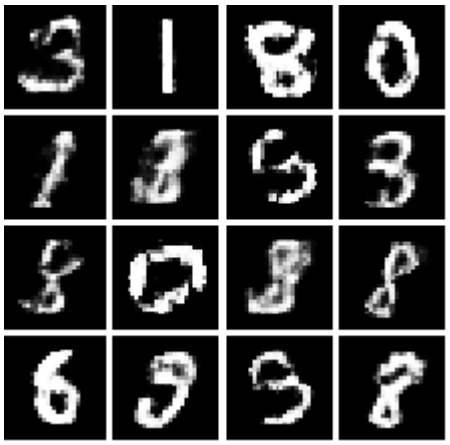
Nous passons, désormais, à l'optimisation de notre code. Le générateur et le discriminateur sont optimisé d'une fois à chaque itération. On fait le point toutes les 100 itérations afin de nous rendre compte de notre progression, de à quel point le programme est entrainé.

```
mb_size = 128


Z_dim = 100

mnist = input_data.read_data_sets('../../MNIST_data', one_hot=True)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

if not os.path.exists('out2/'):
    os.makedirs('out2/')

i = 0

for it in range(1000000):


    if it % 1000 == 0:
        samples = sess.run(G_sample, feed_dict={Z: sample_Z(16, Z_dim)})

        fig = plot(samples)
        plt.savefig('out2/{}.png'.format(str(i).zfill(3)), bbox_inches='tight')
        i += 1
        plt.close(fig)



    X_mb, _ = mnist.train.next_batch(mb_size)



    _, D_loss_curr = sess.run([D_solver, D_loss], feed_dict={X: X_mb, Z: sample_Z(mb_size, Z_dim)})


    _, G_loss_curr = sess.run([G_solver, G_loss], feed_dict={Z: sample_Z(mb_size, Z_dim)})

    if it % 1000 == 0:
        print('Iter: {}'.format(it))
        print('D loss: {:.4}'. format(D_loss_curr))
```

Les résultats que l'on a pour la première itération c'est des cases avec « le bruit » :



Ensuite, au bout d'une centaine d'itération, on voit que le programme a commencé à apprendre et qu'il commence à être entrainé. On remarque que les chiffres commencent à se former.



2) Deepstack

Nous allons utiliser Deepstack pour la détection d'objets dans les images.

```
import numpy as np
import cv2
import requests
```

Les trois lignes de code on les utilise pour importer numpy (une bibliothèque numérique python) sur la première ligne, sur la deuxième ligne nous avons importé opencv (une bibliothèque de vision par ordinateur comme cv2), et sur la troisième ligne nous avons importé request (une bibliothèque python utilisée pour communiquer avec deepstack)

```
cap = cv2.VideoCapture(0)
```

La fonction VideoCapture() est utilisée par opencv pour capturer les images vidéo. L'argument 0 est pour choisir la caméra qu'on utilise.

```
out = cv2.VideoWriter("object_detection_from_camera_to_file.avi", cv2.VideoWriter_fourcc('M', 'J', 'P', 'G')
                                                ,24,((int(cap.get(3)), int(cap.get(4)))))
```

La fonction VideoWriter() est utilisée pour écrire des vidéos. Cette méthode prend quatre arguments qui sont le nom de la vidéo à sauvegarder comme "face_detection_from_camera_to_file.avi", le codec pour sauvegarder la vidéo qui est fait avec la méthode cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'), le nombre d'images à sauvegarder par seconde . Le dernier argument est la dimension pour laquelle la vidéo doit être sauvegardée donnée par ceci ((int(cap.get(3)), int(cap.get(4)))), la méthode cap.get(3) et cap.get(4) renvoie la dimension horizontale et verticale des images vidéo capturées.

```
progress_tracker = 0
prediction_json = {}
skip_frame = 20
```

On déclare les variables progress_tracker, _json (un dictionnaire qui stocke les coordonnées de la boîte englobante retournées par Deepstack) et skip_frame (stocke le nombre d'images à sauter avant que la détection d'objet soit effectuée par Deepstack). Si vous voulez que la détection d'objet soit effectuée sur chaque image, utilisez la valeur 1.

```
while(cap.isOpened()):
    valid, frame = cap.read()
```

cap.isOpened() renvoie la valeur booléenne 1 si la capture vidéo a été initialisée. La boucle while continuera à fonctionner tant que la valeur booléenne sera égale à 1.

cap.read() renvoi 2 valeurs, un booléen qui peut être 1 ou 0 selon qu'une image vidéo a été lue ou non, et les valeurs booléennes sont stockées dans la variable valid. La deuxième valeur renvoyée par cap.read() est l'image vidéo qui est alors stockée dans la variable frame.

```
if valid == True:
    progress_tracker += 1

    if(progress_tracker % skip_frame == 0):
        retval, new_frame = cv2.imencode('.jpg', frame)
        response = requests.post("http://localhost:80/v1/vision/detection",
                                 files={"image":new_frame}).json()
        prediction_json = response['predictions']
        print(prediction_json)
```

Si la valeur booléenne de la première ligne est 1, le bloc de code if situé sous la ligne sera exécuté. Sur la deuxième ligne, la variable progress_tracker sera incrémentée de 1. if(progress_tracker % skip_frame == 0) vérifie si le nombre requis d'images vidéo a été sauté, puis le bloc de code dans l'instruction if sera exécuté. Sur la deuxième ligne, nous avons retval, new_frame = cv2.imencode('.jpg', frame) et ici la méthode cv2.imencode('.jpg', frame) compresse l'image vidéo stockée avec l'argument frame dans un memory buffer qui est ensuite passé à new_frame. Cette méthode prend également un paramètre comme .jpg qui est l'extension qui décrit le format de sortie. Sur la troisième ligne, nous avons response = requests.post("http://localhost:80/v1/vision/detection", files={"image":new_frame}).json() qui envoie une requête post au serveur deepstack AI, et la réponse est enregistrée dans la variable response. Sur la quatrième ligne, les prédictions de la réponse response['predictions'] sont stockées dans la variable prediction_json. Sur la cinquième ligne, le json prédit qui contient les coordonnées de la boîte de délimitation retournées par l'API de détection d'objets de deepstack est imprimé sur le terminal avec print(prediction_json).

```
num_prediction_json = len(prediction_json)
```

On obtient le nombre d'objets dont les coordonnées de la boîte englobante sont renvoyées par le serveur deepstack AI avec num_prediction_json = len(prediction_json).

```
for i in range(num_prediction_json):

    color_space_values = np.random.randint(50, 255, size=(3,))
    red, green, blue = color_space_values
    red, green, blue = int(red), int(green), int(blue)

    frame = cv2.rectangle(frame, (prediction_json[i]['x_min'], prediction_json[i]['y_min']),
                (prediction_json[i]['x_max'], prediction_json[i]['y_max']), (red, green, blue), 1)
```

La première ligne de code for i in range(num_prediction_json) : cherche le nombre total de boîtes de délimitation prédites dans num_prediction_json retourné par deepstack. La deuxième ligne de code color_space_values = np.random.randint(50, 255, size=(3,))est utilisée pour générer des valeurs de couleur aléatoires entre 50 et 255. Ces valeurs représentent la couleur RVB de la boîte englobante. Dans la troisième ligne de code, les valeurs RVB sont extraites : rouge, vert, bleu = valeurs de l'espace couleur. Dans la quatrième ligne de code, les valeurs RVB sont converties en nombres entiers de type red, green, blue = int(red), int(green), int(blue). La cinquième ligne de code est la méthode cv2.rectangle() qui est utilisée pour dessiner les boîtes de délimitation sur l'image vidéo. Cette méthode prend cinq arguments qui sont l'image, les coordonnées en haut à gauche et en bas à droite pour dessiner la boîte de délimitation donnée par (prediction_json[i]['x_min'], prediction_json[i]['y_min']), (prediction_json[i]['x_max'], prediction_json[i]['y_max']), les valeurs de couleur (rouge, vert, bleu) l'épaisseur de la boîte en pixel qui est 1 dans ce cas.

```
out.write(frame)
```

Écris chaque image vidéo. Prends un argument qui est le cadre de l'image vidéo.

```
cap.release()
out.release()
```

La méthode cap.release() et out.release() ferme le fichier vidéo ou le dispositif de capture.

On va arriver a un résultat comme celle-ci

IV. Conclusion

En conclusion, nous avons bien aimé ce sujet. Nous nous sommes partagés le travail de la façon suivante : Joaco a pris la partie théorie des jeux et les parties Deepstack, tandis que Badr a pris la partie Machine learning et les parties sur GANs. Grâce à nos recherches nous avons plutôt bien saisi cette convergence entre deep learning, machine learning et théorie des jeux. En effet, nous avons compris que les notions de la théorie des jeux pouvaient servir à résoudre des problèmes d'optimisation. Nous avons, également, vu des progrès dans l'appréhension des jeux à informations imparfaite qui est largement renforcé par l'adjonction du deep learning dans les moteurs de résolution. Pour ce qui est de la partie technique, nous avons un peu galérer car nous n'avons pas eu de code exemple comme la plupart des groupes. Cependant, nous avons su nous adapter grâce à nos recherches et pu réaliser une partie technique satisfaisante.

Annexe Bibliographie

- https://fr.wikipedia.org/wiki/R%C3%A9seaux_antagonistes_g%C3%A9n%C3%A9ratifs
- https://www.youtube.com/watch?v=J1B2VCd6eTs&list=PLAbhNpVZQkbs2uAc05ebwSqWh6xMRp0AX&index=2
- https://fr.wikipedia.org/wiki/Apprentissage_automatique#Historique
- https://www.decideo.fr/Les-GANs-des-reseaux-artificiels-qui-s-entraident-pour-tromper-l-oeil-humain_a10771.html#:~:text=%C2%AB%20Un%20GAN%20est%20un%20mod%C3%A8le,sc%C3%A9nario%20de%20th%C3%A9orie%20des%20jeux.&text=Cela%20sera%20utilis%C3%A9%20dans%20le,urbanisme%2C%20les%20jeux%2C%20etc
- https://stackabuse.com/introduction-to-gans-with-python-and-tensorflow/
- https://github.com/DeepQuestAI
- https://www.deepstack.ai/downloads
- https://github.com/lifrordi/DeepStack-Leduc
- https://fr.wikipedia.org/wiki/Théorie_des_jeux
- https://www.universalis.fr/encyclopedie/theorie-des-jeux/
- http://renaud.bourles.perso.centrale-marseille.fr/Cours/Theorie_des_jeux.pdf