

Rapport : Gagnant Euromillions



Etudiants Managers :
Florian Lherbeil
Thibault Condemine
Thierry Fernandez

Encadrants :
Jean-Michel Richer
Benoît Da Mota

Sommaire

I - Présentation du projet	3
a - Contexte	3
b - Besoin du client	3
c - Objectifs	3
II - Préparation du projet	4
a - Constitution du cahier des charges	4
b - Choix des technologies	4
c - Architecture du projet	5
III - Gestion du projet	7
a - Formation des développeurs	7
b - Méthodes de communications	8
IV - Etat des lieux du projet	9
a - Livraison de décembre	9
b - Evolutions	12
V - Post-mortem	18
VI - Conclusion	19
Glossaire	20
Sitographie	21
Annexes	22

I - Présentation du projet

a - Contexte

Dans le cadre du cours de Management de projet dispensé lors de notre dernière année de Master Informatique à l'Université d'Angers, nous avons été amenés à encadrer une équipe de quatre développeurs, étudiants en première année de Master Informatique. L'objectif de ce module est de nous faire mener à bien un projet. Dans notre cas, ce projet, appelé "Gagnant Euromillions", concerne le développement d'une toute nouvelle application.

b - Besoin du client

L'EuroMillions est un jeu de hasard bihebdomadaire fonctionnant sur le principe de la loterie. Pour gagner à l'EuroMillions, il faut trouver, parmi les cinquante numéros et les douze étoiles, les cinq numéros et les deux étoiles qui seront tirés au sort au prochain tirage. La demande de notre client, M. Jean-Michel Richer, est de créer un prédicteur du prochain tirage grâce à l'intelligence artificielle et aux réseaux de neurones.

En plus du simple aspect pratique du projet, ces travaux de recherche ont pour objectif de répondre à la problématique suivante : "Les réseaux de neurones ont-ils un rôle à jouer dans les jeux de hasard ?".

L'application qui sera produite permettra donc d'obtenir une approximation des cinq nombres et des deux étoiles constituant le prochain tirage de l'Euromillions.

c - Objectifs

Gagner à l'Euromillions n'est pas le principal objectif de notre projet. S'il était possible de prédire correctement les numéros d'un tirage de l'Euromillions, cela signifierait que ce dernier n'est pas vraiment un jeu de hasard. De ce fait, on peut considérer que le but de ce projet est avant tout pédagogique.

Le Deep Learning est un domaine d'activité populaire et prometteur. L'objectif de notre projet est donc d'apprendre à modéliser un problème pour qu'il puisse être utilisable par un réseau de neurones et de voir ce que nous pouvons en tirer.

Néanmoins, les objectifs concrets suivants ont pu être déterminés avec l'aide de notre client :

- Créer un tutoriel permettant d'installer tout ce qui est nécessaire à l'exécution de l'application finale sur la machine du client. Ce tutoriel permettra plus globalement de configurer une machine pour faire des réseaux de neurones avec Tensorflow et/ou Keras
- Réaliser une application contenant au moins un réseau de neurones permettant de prédire le prochain tirage de l'Euromillions
- Créer un guide utilisateur

II - Préparation du projet

a - Constitution du cahier des charges

Dans le but de répondre au mieux aux besoins du client, un cahier des charges fonctionnel a été réalisé. Ce cahier a pour but de déterminer les outils de gestion utilisés, le périmètre et l'objectif du projet. Pour cela, il a fallu faire une première étude sur les profils d'utilisateurs de l'application. Dans ce cadre, notre projet vise le domaine de la recherche et se limite donc à une utilisation par des chercheurs et des étudiants en informatique.

A l'intérieur de ce cahier des charges, on peut y trouver une liste exhaustive des tâches à réaliser pour mener à bien ce projet. Ces tâches se répartissent sur les semaines imposées à l'équipe de développement, de septembre à décembre.

Un diagramme de gantt (voir annexes) permettant de déterminer les "deadlines" du projet a été proposé en même temps que ce cahier des charges. Ce diagramme nous a permis d'avoir des repères et de réguler la charge de travail de notre équipe en fonctions des différents jalons.

b - Choix des technologies

Pour le choix des technologies, nous avons dû prendre en compte la demande du client ainsi que son environnement actuel de travail. Dans cet optique, plusieurs variables ont été prises en compte : son système d'exploitation, la configuration de sa machine et les ressources de l'équipe de développement.

1) Système d'exploitation

Concernant le système d'exploitation, deux versions différentes d'Ubuntu ont été retenues :

- **Ubuntu 18.04 LTS**, version présente sur la machine du client
- **Ubuntu 16.04 LTS**, version présente sur la machine des développeurs

Ce sont des systèmes adaptés aux développement et à la recherche en informatique.

2) Frameworks de réseaux de neurones

Deux frameworks ont été retenus pour construire des réseaux de neurones :

- **Tensorflow** : un framework développé par Google, ce dernier a été choisi car il permet de créer facilement des réseaux de neurones. De plus, cette technologie est populaire auprès de la communauté scientifique. Cette popularité résulte en une documentation très fournie et de nombreuses sources disponibles en ligne. Ce framework, disponible en version CPU et GPU, a été utilisé en mode GPU pour des raisons de performances ainsi que pour respecter la demande du client.

- **Keras** : une surcouche de Tensorflow qui est censée faciliter son utilisation. Keras est utilisé pour créer d'autres réseaux de neurones afin de comparer les deux solutions.

Enfin, nous avons aussi la possibilité d'utiliser Pytorch. C'est une technologie assez similaire à Tensorflow et elle est tout autant utilisée. Même si Pytorch est reconnu comme un outil simple d'utilisation, nous avons trouvé plus d'exemples et d'application de cours pour Tensorflow, ce qui nous a conforté dans notre choix.

3) Langage de programmation

Pour utiliser les frameworks de réseaux de neurones, nous avons le choix entre le C++ et Python. Python a été retenu pour sa praticité, sa flexibilité et sa facilité d'usage par rapport au C++. De plus, la plupart des démonstrations utilisant Tensorflow et Keras sont faites en Python. La version 3.7 de Python a donc été retenue (la version conseillée par les équipes en charge des frameworks cités plus haut).

4) Gestionnaire de paquets

Pour simplifier la mise en place de l'environnement d'exécution du projet et par conséquent l'intégration des dépendances et des bibliothèques, nous avons fait le choix d'utiliser Anaconda. Il s'agit d'un gestionnaire de paquets et d'un système de gestion d'environnement de développement, de plus il est open-source et largement utilisé par la communauté Python.

5) Outils & choix divers

En terme d'outils, nous avons fait quelques suggestions aux développeurs. L'utilisation d'un IDE comme Pycharm, leur permettant d'être plus efficace en développement Python.

Il a été convenu que malgré tout, chaque développeur pouvait travailler à leurs manières tant que le code fourni est uniforme et compatible avec le projet.

Dans un souci de qualité et de sécurité nous leur avons imposé un outil de développement largement utilisé dans le milieu professionnel, le gestionnaire de versions GIT. Un outil indispensable qui a demandé un long moment d'adaptation aux étudiants de M1. L'ensemble du projet est donc disponible dans un repository Github situé à l'adresse suivante : <https://github.com/Thierryfe/DeepLearningEuroMillion>

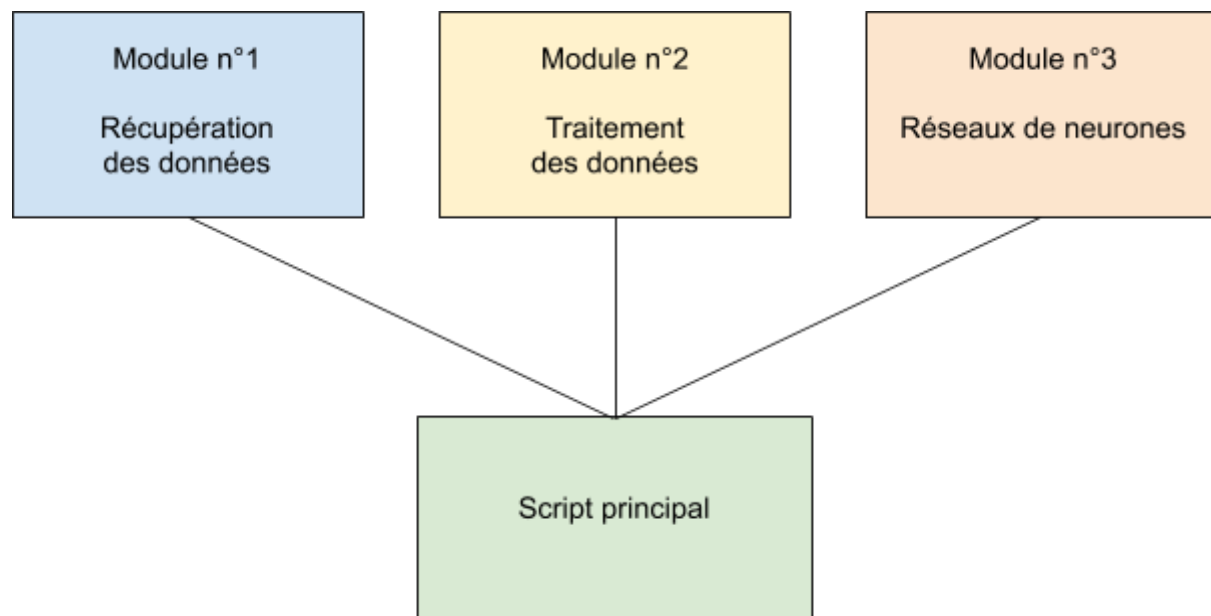
Durant la phase de développement, il a été laissé le choix aux développeurs de choisir des bibliothèques complémentaires dans le but de leur faciliter diverses étapes notamment le traitement des données.

Pour simuler le système d'exploitation de la machine client, il a été utilisé une solution de machine virtuelle sous Ubuntu 18.04.

c - Architecture du projet

Une fois l'idée générale du projet acquise et les technologies choisies nous avons pu décider d'une architecture générale.

Elle est découpée en trois modules Python, ces modules ont pour objectif de simplifier la mise en place de script exploitant les données de tirages sur des réseaux de neurones, il s'agit d'un ensemble de fonctions encapsulées :



Le module n°1 a pour objectif de récupérer les données à jour depuis un serveur web via des requêtes HTTP.

Pour le traitement des données, le module n°2 a pour rôle de formater les données afin de les rendre exploitables pour les différents moteurs de réseaux de neurones

Pour finir le module n°3 contient tous les moteurs réalisés par les développeurs.

À l'aide de ces modules, nous avons pu produire un script principal faisant appel à ces derniers, cela représente notre objectif en terme de souplesse de code.

III - Gestion du projet

a - Formation des développeurs

Dès le début du projet et en choisissant les technologies que nous allions utiliser, nous savions pertinemment que notre équipe de développeur n'avait pas particulièrement les connaissances nécessaires au projet. Pour remédier à cela, les premières semaines du projet ont été entièrement consacrées à leur formation.

Trois principaux points ont alors été abordés :

- Git
- Python
- Le Deep Learning (à travers Keras et Tensorflow)

L'ensemble des ressources évoquées ci-dessous sont accessibles via la sitographie présente en fin de ce rapport.

Le gestionnaire de version Git n'était quasiment pas connu de notre équipe de développeur et nous leur avons fourni une vidéo Youtube "Git & GitHub Crash Course For Beginners" de la chaîne Traversy Media. Néanmoins, cette formation s'est avérée insuffisante puisque nous avons dû leur fournir individuellement d'autres explications au cours du projet.

Le langage que nous avons choisi, Python, ne fait pas officiellement partie du programme de notre formation. Mais, un bon développeur doit être en mesure de s'adapter rapidement à un langage qu'il ne connaît pas alors nous leur avons juste fourni le tutoriel python de la W3schools afin qu'il puisse avoir facilement accès à tout ce qui pourrait leur être nécessaire au développement de notre application.

Enfin, pour les connaissances sur le Deep Learning et Tensorflow, nous leur avons fourni des liens de vidéos Youtube présentes sur la chaîne d'Alexander Amini. Ces différentes vidéos d'introductions aux réseaux de neurones font parties d'un cours dispensé par le MIT (Massachusetts Institute of Technology) appelé "Introduction to Deep Learning". En parallèle, ils ont aussi été invités à lire une courte introduction aux réseaux de neurones et à Tensorflow disponible sur le site towardsdatascience.com.

b - Méthodes de communications

Pour au mieux répondre à la problématique de communication inhérente à un projet, nous avons fait le choix de communiquer à deux niveaux.

Un modèle de communication rapprochée avec un outil moderne : Discord. Il permet de communiquer en instantané auprès des différents participants du projets avec moins de formalité, Discord nous a permis de transmettre des informations rapidement sur certains aspects du projets. De plus il fournit aux développeurs un espace de travail pour communiquer entre eux mais aussi un moyen d'interagir avec nous pour d'éventuels éclaircissements.

Ce modèle est donc adapté à une communication occasionnelle.

Le second niveau a pour but d'établir une communication plus formelle entre les développeurs et les chefs de projets, pour cela il a été mis en place un système de fiche hebdomadaire (voir annexes).

Ces fiches fournies chaque semaine contiennent trois parties informatives, une note introduisant des informations diverses, une liste de tâches assignés à tous les membres du projet et un talon contenant une partie à remplir par les développeurs pour nous faire remonter leur avancement et éventuelles difficultés.

Aussi nous avons profité de chaque créneau de management de projet pour réaliser des échanges avec les développeurs via la fiche hebdomadaire sous forme d'une réunion dans laquelle cette fiche était détaillée par l'un des managers. La fin de ces réunions était une occasion pour les développeurs d'interagir avec nous.

À noter que ces méthodes de communication découlent de notre expérience professionnelle.

IV - Etat des lieux du projet

a - Livraison de décembre

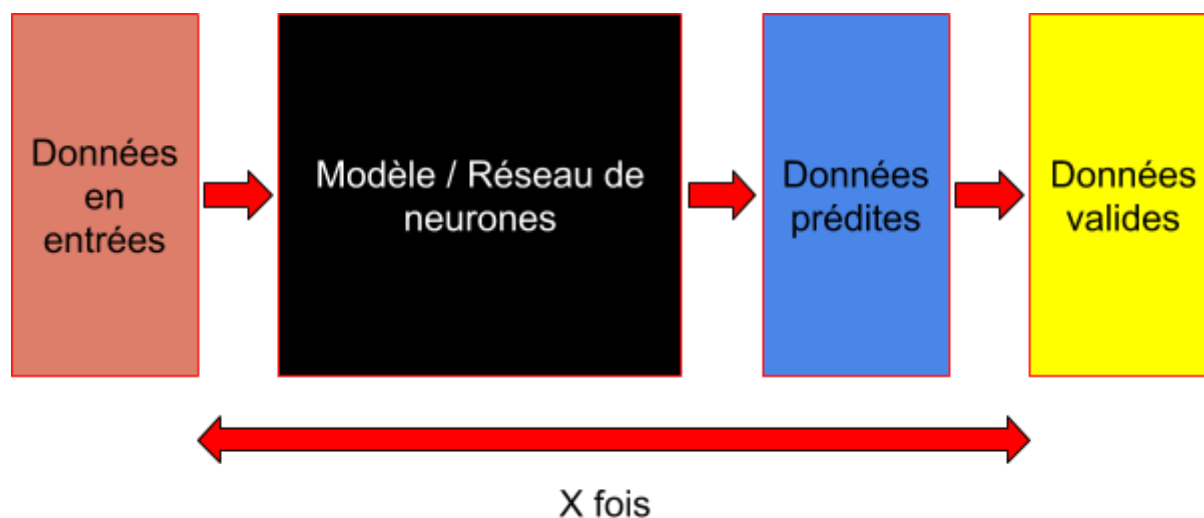
Notre équipe de développement était disponible pour ce projet pour une période de quatre mois, de septembre à décembre. Au terme de ces mois d'implication, nos développeurs ont livré :

- Un programme permettant de prédire le prochain tirage de l'Euromillions. Ce programme contient les trois réseaux de neurones suivants :
 - Un RNN développé en Keras
 - Un CNN développé en Keras
 - Un RNN développé en Tensorflow
- Deux tutoriels d'installation (en version Ubuntu 16.04 et 18.04) permettant de créer un environnement de développement pour utiliser Keras/Tensorflow avec Python et exécuter notre programme. Un de ces tutoriels est disponible en annexe.

Nous allons maintenant pouvoir passer à l'illustration des mois de développement de notre équipe, pour cela nous allons vous expliquer le réseau de neurones le plus abouti qui a été livré.

1) Introduction rapide aux réseaux de neurones

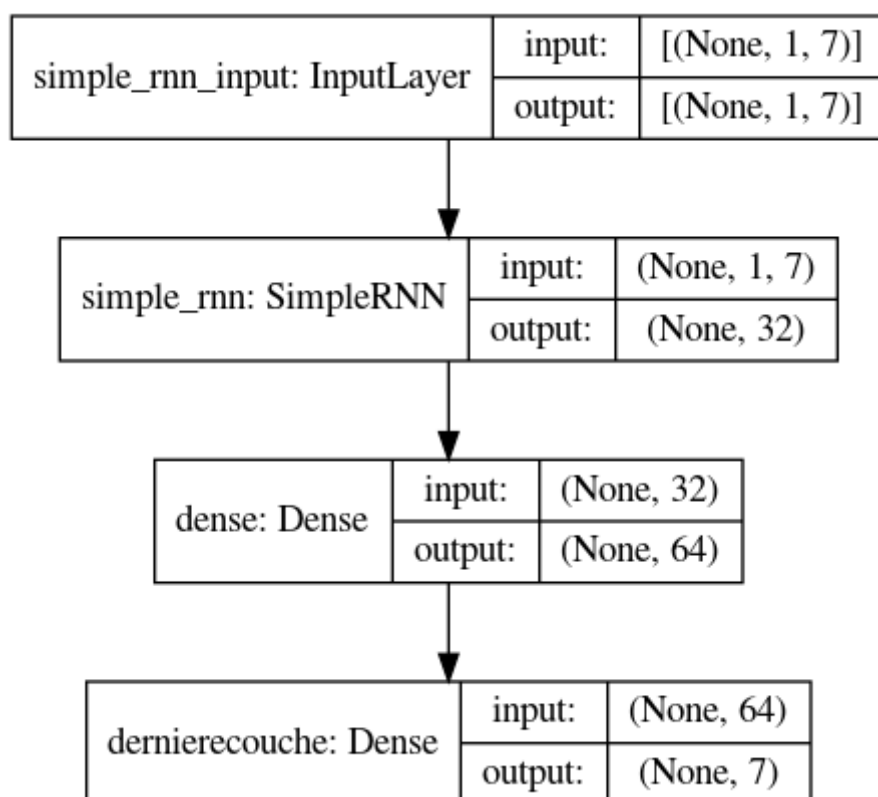
Afin de comprendre plus facilement la partie qui suit, il est essentiel d'avoir une vision schématique de ce que peut être un réseau de neurone en informatique :



Dans le cas des réseaux de neurones, on dit qu'on va entraîner un modèle (la boîte noire dans le schéma) en lui fournissant notre jeu de données valide en entrée (marron) et en sortie (jaune). A chaque passage le modèle va prédire une "réponse" et s'ajuster en fonction de la cohérence entre la donnée prédites (bleu) et la données valides (jaune).

2) Réseaux réalisés par les développeurs.

Comme un graphe est plus explicite qu'un long discours, voici le réseau que nous souhaitons vous présenter :



Il a la particularité d'être récurrent, c'est à dire que dans notre réseau, des neurones ont une "mémoire" des étapes précédentes qu'ils ont réalisées, afin de proposer une "prédiction" à la couche de neurone suivante.

Ce modèle est particulièrement adapté à notre jeu de données séquentiel (la première boule tirée influe sur la deuxième et ainsi de suite dans le cas de l'euromillions).

Notre équipe de développeurs a donc choisi d'implémenter un réseau à trois couches de neurones (layers). La première couche du réseau est la couche récurrente (SimpleRNN), c'est elle qui donne la particularité à notre réseau.

Les deux dernières couches sont de régulation (Dense). La première couche de régulation est obligatoire avec la bibliothèque keras car la couche récurrente renvoie des données qui ne seront pas comparable avec notre jeu valide en sortie. Enfin la deuxième couche de régulation est purement arbitraire, l'équipe a choisi de l'utiliser pour expérimenter.

Concernant le nombre de neurones présent dans notre réseau. Grâce au graphe, il est facile de faire le compte. Nous avons sept neurones en entrée dans la couche récurrente afin de transmettre les données au réseau sous forme brute ([1,8,45,34,12,4,9],...). Chaque neurone prend donc un chiffre en entrée.

La première couche de régulation en possède trente-deux et la seconde soixante-quatre. Aucune de ces deux valeurs sont fixe et il peut être judicieux de les modifier pour un apprentissage plus rapide par exemple.

Enfin de la même façon qu'en entrée, il y a sept neurones en sortie afin d'avoir un chiffre par neurone.

Une fois le modèle entraîné, il prédit le prochain tirage à partir du dernier tirage recensé.

Dans le projet, la fonction qui encapsule ce réseau offre différents arguments :

- Date1 et date2 qui représentent l'intervalle entre deux années afin de réduire ou d'augmenter le nombre de tirages en entrée du réseau
- Nb_epoch qui permet de choisir le nombre d'itération d'entraînement du réseau.

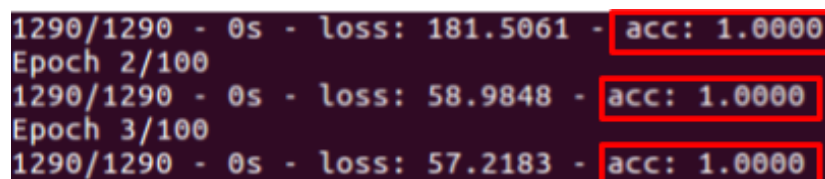
`NetworkRNN(Nb_epoch,date1,date2)`

3) Limites du livrable développé par les étudiants de Master 1

Le livrable est donc fonctionnel mais ne répond pas véritablement à la question du client.

Plusieurs problématiques sont liées à cela :

- Premièrement le jeu de données en entrée est identique à celui de validation fournit en sortie. Le réseau apprend donc à copier un tirage dans une autre tirage. On le vérifie grâce à la rapidité que le réseau obtient un taux de réussite de 100% à chaque prédiction.



```
1290/1290 - 0s - loss: 181.5061 - acc: 1.0000
Epoch 2/100
1290/1290 - 0s - loss: 58.9848 - acc: 1.0000
Epoch 3/100
1290/1290 - 0s - loss: 57.2183 - acc: 1.0000
```

- Deuxièmement, les données auraient dû être exploitées sous un autre format.
- Ensuite, les développeurs auraient dû découper ce jeu de données en deux instances. Un jeu pour entraîner le modèle et un jeu de test afin d'analyser l'efficacité du modèle.
- Dernièrement, il aurait pu être intéressant de modifier le nombre de neurones sur les couches intermédiaires en paramètre et le nombre d'échantillons avant la mise à jour d'une valeur dans le réseau (gradient)

Chacune de ces remarques ne sont pas exclusivement à mettre sur le compte de l'implication des développeurs. Notre manque de connaissances dans ce domaine au début du projet y a également contribué. C'est pour cela que nous avons décidé de réaliser une implémentation plus adéquate et mettant en valeur le travail réalisé par les développeurs.

b - Evolutions

Depuis la livraison de la production de notre équipe de développement, nous avons effectué quelques modifications permettant de rendre l'application utilisable par notre client. Le code a dû être réorganisé afin d'offrir une expérience utilisateur cohérente. En effet, en l'état, il était impossible de lancer tous les réseaux de neurones à l'aide du programme principal et seul un réseau par défaut était lancé. Cela nous a permis de constater que l'application n'avait pas été développée dans le but d'être utilisée sans modifier directement le code source de notre programme. Notre objectif a donc été de modifier l'application pour qu'elle puisse être utilisée et paramétrée uniquement en lançant le programme principal.

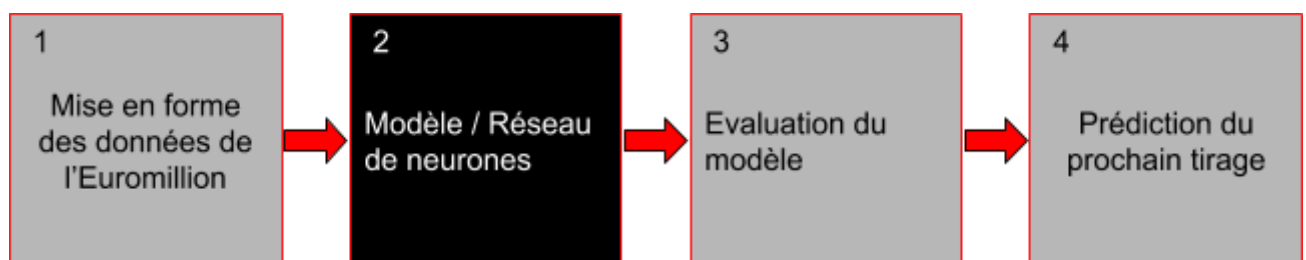
Il est à présent possible de paramétrer notre réseau de neurones via deux méthodes :

- Pour la première méthode, il suffit de rajouter des arguments au lancement de l'application
- Pour la deuxième méthode, il suffit de répondre aux quelques questions posées au lancement de l'application. Cette méthode se lance uniquement lorsqu'aucun argument n'est inséré au lancement de l'application.

Cette modification a donné lieu à la création d'un guide utilisateur (disponible en annexe) qui permet de comprendre comment utiliser l'application.

Parallèlement à l'ajout de ces fonctionnalités, nous avons donc souhaité développer un nouveau réseau de neurones (RNN) qui répond cette fois-ci à la demande du client. L'ajout de ce réseau a entraîné une modification des tutoriels d'installation puisque de nouvelles bibliothèques comme Scikit-learn, pydot et matplotlib ont été nécessaires à sa réalisation.

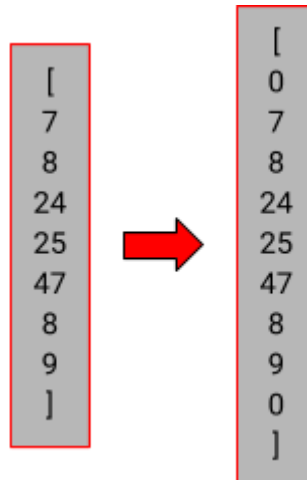
Pour expliciter notre nouvelle implémentation, prenons un peu de recul afin d'appréhender l'ensemble du script que nous avons produit. Il est composé de quatre grandes fonctions principales illustrées ci-dessous :



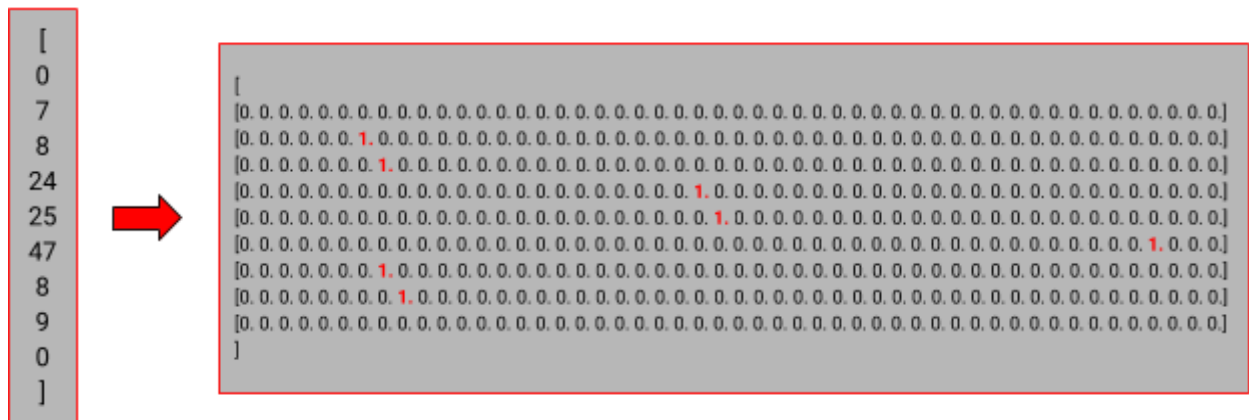
1) Mise en forme des données de l'Euromillion

Première fonction que nous avons réalisé, nous transformons les données brutes des tirages de l'Euromillions afin de ne pas répéter le problème du précédent réseau. Pour cela plusieurs étapes sont nécessaire :

Dans un premier temps, nous ajoutons un zéro au début et à la fin de chaque tirage.



Ensuite chaque nombre de chaque tirage se voit transformé en une liste de cinquante zéro. Un des zéro est remplacé par un 1 afin de savoir à quel chiffre nous avons affaire.



Troisième et avant dernière étape : il est nécessaire de différencier le jeu de données en entrée et en sortie pour ne pas reproduire le cas incohérent du précédent réseau. En entrée, l'ensemble de nos tirages convertis se verront leur zéro de fin retiré et celui du début sera retiré du jeu de données final.

[
0
7
8
24
25
47
8
9
~~8~~
]

Exemple d'un tirage en entrée

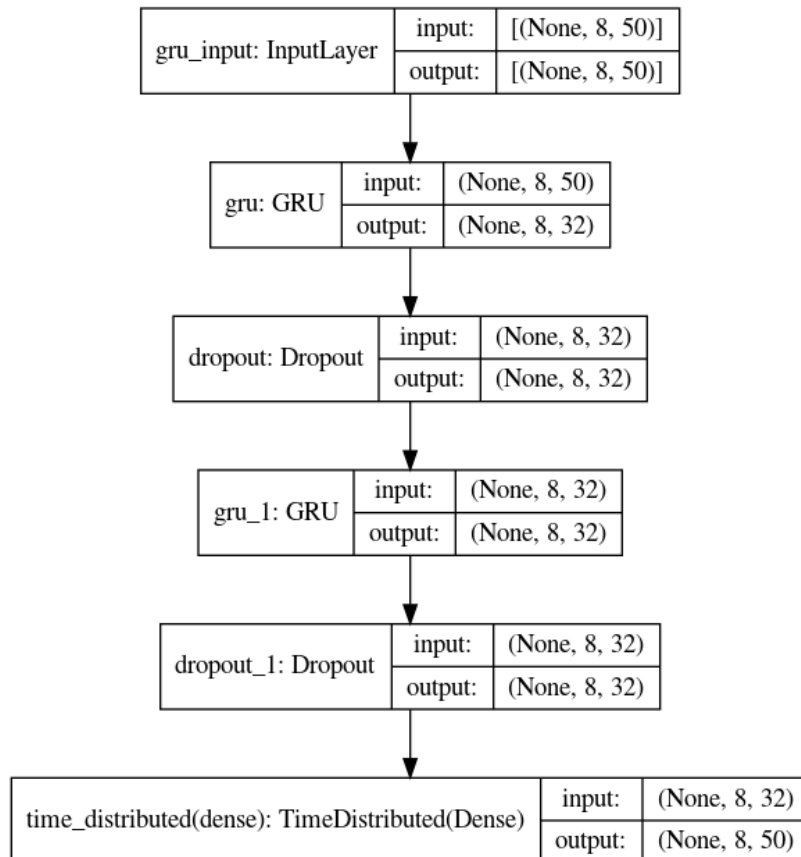
[
~~8~~
7
8
24
25
47
8
9
0
]

Exemple d'un tirage en sortie

Enfin, le jeu de données global est dupliqué pour obtenir un jeu de données en entrée et un jeu de données en sortie valide. Le jeu de données en entrée est séparé en deux groupes : l'un pour entraîner notre réseau de notre et le deuxième pour évaluer l'efficacité du réseau. On fait exactement la même chose avec le jeu de donnée de validation. Les données sont séparées à hauteur de 70% pour l'entraînement et 30% pour le test.

2) Modèle / Réseau de neurones

Une fois n'est pas coutume, nous utilisons un graphe pour présenter notre réseau :

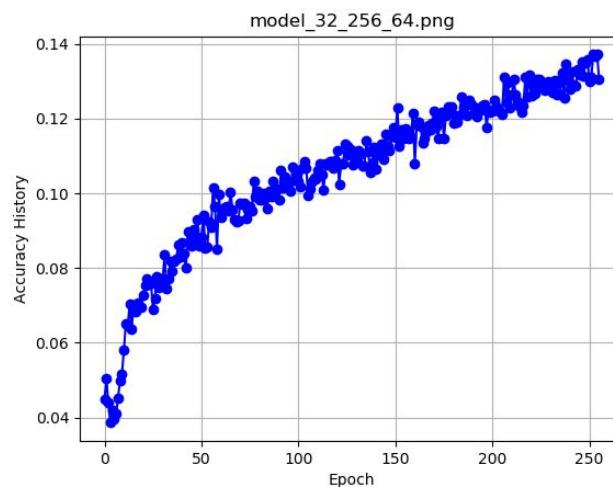


Il est facile de s'apercevoir que notre réseau est différent du précédent. Cependant il garde la même philosophie de récursivité liée à notre jeu de données.

Les principales modifications sont les suivantes :

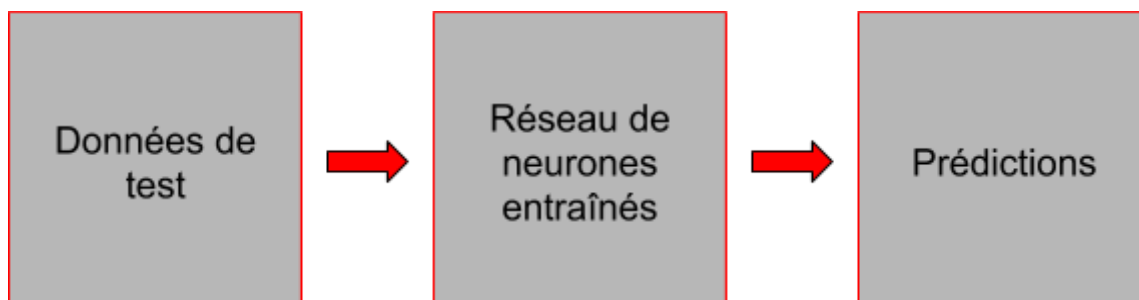
- Les nombres de neurones en entrée et en sortie sont plus élevés pour correspondre à notre formatage des données. Comme on a pu le voir sur le réseau précédente, il y avait 7 neurones en entrée qui prennent un tirage un par un. Cette fois-ci le réseau attend huit fois un nombre représenté par une liste de zéro et un seul 1.
- Le réseau est composé cette fois-ci de deux couches récurrentes (GRU) et d'une seule couche de régulation.
- On peut voir aussi l'apparition de "Dropout" entre chaque couche qui "désactive aléatoirement un petit nombre de neurones".
- Pour finir, notre couche de régulation est utilisée avec un time distributed afin de "reconstruire un tirage en sortie".

Notre fonction qui encapsule le réseau sauvegarde le modèle afin de le réutiliser plus tard et peut également générer un graphique de la courbe d'apprentissage du réseau en fonction du nombre d'itérations et la de la précision du modèle.



3) Evaluation du modèle

Troisième et avant-dernière fonction que nous avons développée : l'évaluation du modèle. Pour réaliser une évaluation cohérente d'un réseau de neurones, il faut utiliser les données de test construites dans la première étape afin d'effectuer des prédictions.



Une fois les prédictions réalisées, il ne reste plus qu'à comparer les prédictions à notre jeu de données de validation afin d'en faire ressortir le nombre de prédictions correctes.

Notre fonction va donc chercher dans un dossier de l'application l'ensemble des modèles déjà générés et les évaluer. À la fin de toutes les évaluations un fichier json est construit contenant toutes les données d'évaluation.

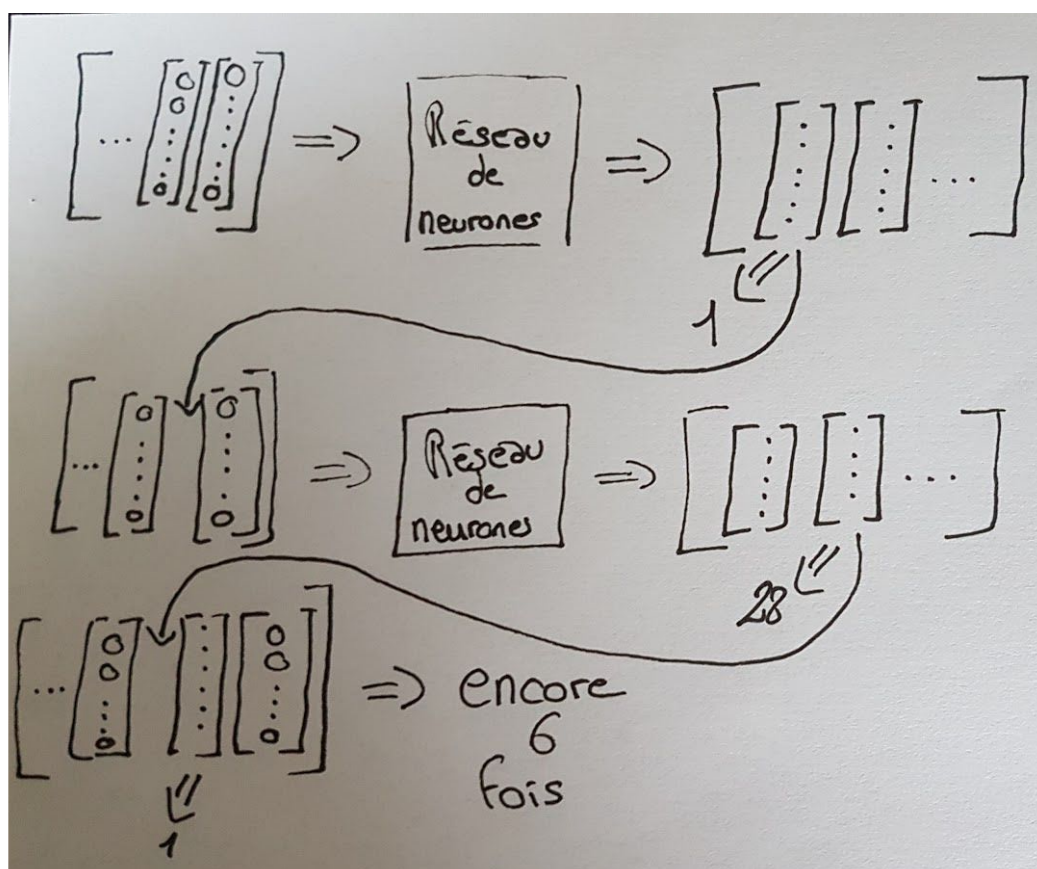
Voici une ligne du fichier qui contient plusieurs données : en premier la perte (loss) qui ne sera pas explicité ici. En deuxième le pourcentage de nos bonnes prédictions et pour finir le "model" sur lequel ces données ont été tirées.

```
"[[3.268832967570894, 0.062015504, 'model_32_128_64.h5']]"
```

4) Prédiction du prochain tirage

Dernière fonction que nous avons réalisé est celle la plus attendue par le client, cette fonction tente de répondre à la question "Quel sera le prochain tirage de l'Euromillions ?".

Son fonctionnement est le suivant :



Ainsi pour prédire le prochain tirage nous devons passer à un de nos modèles entraînés un pseudo tirage composé exclusivement de zéros afin de prédire un premier chiffre. À chaque itération on remplace un zéro par le chiffre prédit, correspondant au nombre de fois où l'opération s'est répétée.

À partir du moment où les chiffres ont été prédit, le résultat est stocké dans un fichier json.

Voici une ligne du fichier composée des 7 chiffres à jouer et du modèle qui a permis de prédire ce tirage :

```
"[[10, 18, 26, 44, 2, 8, 9, 'model_32_128_64.h5']]"
```

5) Les fonctions

Chacune des fonctions sont utilisables indépendamment les une des autres et paramétrable en python :

Formatage des données :

```
data_form(year, to_year) # Formater les tirages entre deux années
```

Réseau de neurones :

```
network(units = 32, epochs = 128, batch_size = 32, year = 2004, to_year = 2020)  
# Appelle le réseau de neurones en paramétrant le nombre de neurones sur la couche  
deux et trois, le nombre d'itérations, le nombre de tirages utilisés en même temps et les  
intervalles d'années pour la fonction data_form()
```

Evaluation du/des modèles :

```
evaluate(year = 2004, to_year = 2020) # Évalue le modèle sur les tirages compris entre les  
deux années
```

Prédiction sur un/des modèles :

```
predict() # Prédit le prochain tirage
```

V - Post-mortem

Maintenant que le management du projet est terminé, nous avons pu prendre un peu de recul vis à vis de ce dernier.

Au début du projet, nous avions des idées assez claires de comment nous voulions encadrer notre équipe de développeurs. Nous voulions principalement être là pour les orienter dans leur travail, en leur laissant un peu de liberté au niveau de leur organisation. Cependant, arrivé à la moitié du projet, nous avons dû faire le constat suivant : les libertés que nous leur avons données ne leur avaient pas été bénéfiques et cela se ressentait sur leurs productions. Pour remédier à ce problème, nous avons pris la décision de ne plus rien laisser au hasard et de mieux structurer les choses (notamment le code de l'application). Dans l'optique de recommencer un projet de ce type, il nous semblerait nécessaire de mieux préparer le terrain avant de lancer les membres de notre équipe dans leurs différentes tâches de développement. De plus il faudrait probablement une approche managériale plus personnalisée pour prendre en compte les individualités de chacun.

Un autre point important à aborder dans cette rétrospective est la communication. Entre les managers et les développeurs, cette dernière s'est plutôt bien passée dans l'ensemble. Cependant, cela n'a pas été forcément le cas de la communication entre les différents développeurs. Dès le début du projet, nous avons pris la décision de créer deux groupes de développeurs afin de mieux séparer les tâches. Si cela a bien fonctionné pour un des deux groupes, nous nous sommes vite rendu compte que les deux membres de l'autre groupe travaillaient séparément et malheureusement, nous n'avons pas été en mesure de trouver une solution pour y remédier sur le coup. Les deux personnes ne se côtoyant pas du tout hors du projet, il leur était difficile de vouloir travailler ensemble et modifier les groupes n'y aurait sûrement rien changé.

Enfin, nous pouvons terminer cette partie en soulignant que notre équipe de développement a été globalement satisfaite de notre management. Les points positifs qui ressortent plus sont le fait d'avoir mis en place une communication formelle pour remonter les difficultés rencontrées ainsi que notre capacité à nous adapter aux différentes situations.

Au final, nous avons également réussi à transmettre une part de nos compétences et de nos connaissances à notre équipe. Toutes les personnes ayant participé au projet ont partagé une expérience qui leur sera bénéfique dans la suite de leur carrière professionnelle.

VI - Conclusion

La gestion de ce projet représentait une bonne occasion de mettre à profit nos compétences professionnelles acquises durant notre formation, soit au travers de nos cours, soit au travers de nos expériences en entreprise.

Ce projet étant particulier comparé aux autres proposés pour cette unité d'enseignement, il a demandé une faculté d'adaptation à tous les membres de l'équipe de management pour pouvoir fournir aux étudiants de M1 un projet de développement au travers des demandes du client qui sont axés sur un projet de recherche.

Avec cette manière de réaliser le projet nous avons pu trouver des éléments de réponse à la problématique citée en introduction : "Les réseaux de neurones ont-ils un rôle à jouer dans les jeux de hasard ?". Instinctivement, une première réponse nous était venue à l'esprit : "non". Le développement de ce projet ne nous a pas permis d'infirmer cette intuition. De par les réseaux de neurones développés et testés, nous n'avons pas pu trouver de corrélations entre les nombres dans l'ensemble des tirages depuis 2004. Le fait est, que les réseaux de neurones soient très sensibles aux données. De plus, leurs configurations sont très variées en fonction des données à traiter, ce qui implique que les résultats sont difficilement exploitables.

Malgré tout, comme nous avons pu voir à travers les différents réseaux développés pour ce projet, il est possible de réaliser un réseau relativement cohérent par rapport aux données que nous avons collectées.

Le sujet de ce projet fut une bonne opportunité d'appréhender le monde du deep learning, un domaine relativement nébuleux pour notre équipe.

Glossaire

- **GPU** : Graphics Processing Unit - Processeur Graphique
- **CPU** : Central Processing Unit - Unité centrale de traitement
- **CNN** : Convolutional Neural Networks - Réseau neuronal convolutif
- **RNN** : Recurrent neural network - Réseau de neurones récurrents

Sitographie

- Traversy Media - Git & GitHub Crash Course For Beginners :
https://www.youtube.com/watch?v=SWYqp7iY_Tc&t=262s&fbclid=IwAR3130TZSuokstKrpF0ViS7DygOliHujlpueYZtllpO7IR-lrkKEVOVWpyg
- W3Schools - Python Tutorial :
<https://www.w3schools.com/python/default.asp>
- Alexander Amini - MIT 6.S191: Introduction to Deep Learning :
https://www.youtube.com/watch?v=5v1JnYv_yWs&index=1&list=PLtBw6njQRU-rwp5_7C0oIVt26ZgjG9NI
- Alexander Amini - MIT 6.S191: Recurrent Neural Networks :
https://www.youtube.com/watch?v=H-HVZJ7kGI0&list=PLtBw6njQRU-rwp5_7C0oIVt26ZgjG9NI&index=3
- Alexander Amini - Convolutionnn MIT 6.S191: Convolutional Neural Networks :
https://www.youtube.com/watch?v=H-HVZJ7kGI0&list=PLtBw6njQRU-rwp5_7C0oIVt26ZgjG9NI&index=3
- TowardsDataScience - First neural network for beginners explained :
<https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf>
- Site officiel de Tensorflow : <https://www.tensorflow.org/>
- Documentation Keras : <https://keras.io/>

Annexes

- **Fiche hebdomadaire** : page 23
- **Tutoriel d'installation Ubuntu 18.04** : pages 24, 25, 26, 27
- **Guide utilisateur** : pages 28, 29
- **Diagramme de Gantt** : page 30

Fiche hebdomadaire :

Notes M2 :

Cette semaine commence avec une bonne nouvelle : la machine du client est enfin complètement opérationnelle et connectée au réseau. Nous vous invitons donc à vous rapprocher de Mr. Richer pour qu'il vous permette d'y accéder.

Rappel des groupes :

- Groupe 1 : Brian et Corentin B
- Groupe 2 : Corentin C et Abderrahim

Rappel contacts :

- thibault.condemine@etud.univ-angers.fr
- florian.lherbeil@etud.univ-angers.fr
- thierry.fernandez@etud.univ-angers.fr

Tâches de la semaine/jours :

- Installation des outils sur la machine du client : (Groupe 1)
 - Vous devez tester votre tutoriel d'installation sur la machine du client
- Installation sur vos machines si ce n'est pas déjà fait (Tous)
- Récupération des données de l'EuroMillions : (Groupe 2)
 - Trouver les données sur internet
 - Effectuer un script Python afin de les récupérer automatiquement
- Réflexion sur la mise en forme des données pour qu'elles soient facilement exploitable pour vos réseaux de neurones (Tous)
- **BONUS** : Vous regarderez si il est possible de scripter l'installation sous Ubuntu 18.04

Avancement & notes M1 (À RENDRE PAR MAIL) :

Afin que le projet se passe au mieux pour tout le monde (nous avons été à votre place aussi...). Nous vous demandons donc chacun de répondre aux questions ci dessous :

Nom Prénom :

- Avancement (estimation en %)
- Aide requise ?
- Difficultés ?
- Remarques ?

Si vous avez rien à signaler pour une des questions merci de préciser "rien à signaler".

Instructions d'installation - Ubuntu 18.04

Instructions d'installation rapide

Vous aurez préalablement besoin d'un accès à internet, d'une machine sous Ubuntu 18.04 et de la commande `wget`.

Nous commencerons par installer le gestionnaire de paquets Anaconda. Pour l'installer, il vous suffit de copier-coller les lignes suivantes dans votre terminal :

```
1 $ wget https://repo.anaconda.com/archive/Anaconda3-2019.07-Linux-x86_64.sh
2 $ chmod +x Anaconda3-2019.07-Linux-x86_64.sh
3 $ ./Anaconda3-2019.07-Linux-x86_64.sh -b
```

Note :

Si vous rencontrez le message d'erreur suivant « conda : commande introuvable », exécutez la commande suivante :

```
1 $ export PATH=~/.anaconda3/bin:$PATH
```

Si vous rencontrez le message d'erreur suivant : « CommandNotFoundError : Your shell has not been properly configured to use 'conda activate'. » Vous devez exécuter la commande suivante :

```
1 $ conda init <SHELL_NAME>
```

Si l'installation s'est bien déroulée, vous pouvez configurer un environnement de travail avec conda. Il vous suffit de copier-coller les lignes suivantes et de les exécuter dans votre terminal :

```
1 #On désactive l'environnement de base
2 $ conda config --set auto_activate_base false
3
4 #On crée un nouvel environnement du nom d'euromillion
5 $ conda create --name euromillion python=3.7
6
7 #On active l'environnement créé
8 $ conda activate euromillion
9
10 #On installe les packages dont on aura besoin
11 $ conda install -y tensorflow-gpu
12 $ conda install -y keras-gpu
```

Normalement rendu à ce stade là vous avez un tensorflow exécutable. Pour tester votre installation, exécutez la commande suivante dans l'environnement précédemment créé :

```
1 $ wget https://raw.githubusercontent.com/Thierryfe/DeepLearningEuroMillion/master/Documents/Instructions%20d\'installation/testTF.py && python3 testTF.py
```

On va maintenant procéder à l'installation de packages supplémentaires pour utiliser le projet :

```
1 # les packages supplémentaires pour exécuter le projet
2 $ conda install -yc conda-forge numpy
3 $ conda install -yc anaconda beautifulsoup4
4 $ conda install -yc rmg xlrd
5 $ conda install -yc anaconda pandas
6 $ conda install -yc anaconda requests
7 $ conda install -yc anaconda pydot
8 $ conda install -yc conda-forge matplotlib
9 $ conda install -yc intel scikit-learn
```

Si aucune erreur n'a été détectée, l'installation s'est déroulée correctement.


```
1 #On peut désactiver l'environnement si on ne compte pas l'utiliser tout de  
   suite  
2 $ conda deactivate
```

Instructions d'installation détaillée

Nous allons ici installer les bibliothèques nécessaires à la création d'un réseau de neurones. Pour cela il vous faudra une machine ayant python, une distribution Ubuntu et une connexion internet. Nous allons utiliser les bibliothèques python, TensorFlow, Keras pour cela nous utiliserons anaconda qui nous permettra de les installer simplement.

On peut se rendre sur le site anaconda pour obtenir le fichier d'installation de anaconda, il suffit ensuite d'aller dans la section download, ensuite choisir la version à télécharger selon son système d'exploitation (Windows, Linux ou MacOS), nous choisirons Linux pour notre tutoriel.

Nous choisirons la version 3.7 car la version 2.7 de python ne sera plus maintenue en 2020 (cf Python.org). Nous prendrons la version 64-Bit (x86).

On peut aussi faire la commande suivante qui téléchargera directement le bon fichier dans le répertoire courant. :

```
1 $ wget https://repo.anaconda.com/archive/Anaconda3-2019.07-Linux-x86_64.sh
```

Une fois le fichier téléchargé il faut lui donner le droit d'exécution, en tapant la commande suivante dans votre terminal là où est situé votre fichier :

```
1 $ chmod +x Anaconda3-2019.07-Linux-x86_64.sh
```

Cela va permettre au fichier de s'exécuter et d'installer anaconda, on exécute donc le fichier avec l'option -b pour gagner du temps :

```
1 $ ./Anaconda3-2019.07-Linux-x86_64.sh -b
```

Note :

Si vous rencontrez le message d'erreur suivant « conda : commande introuvable », exécutez la commande suivante :

```
1 $ export PATH=~/.anaconda3/bin:$PATH
```

Si vous rencontrez le message d'erreur suivant : « CommandNotFoundError : Your shell has not been properly configured to use 'conda activate'. » Vous devez exécuter la commande suivante :

```
1 $ conda init <SHELL_NAME>
```

Une fois anaconda mis en place, nous allons installer des packages permettant d'utiliser les réseaux de neurones. Comme nous allons utiliser des packages et que nous ne voulons pas forcément avoir toutes ces bibliothèques sur toutes nos applications, nous allons donc créer un environnement python avec anaconda pour pouvoir installer les packages.

Les packages à installer variant d'un projet à l'autre il peut être intéressant d'avoir des environnements bien cloisonnés pour ne pas tout mélanger.

Nous allons créer notre environnement projeteuromillion :

```
1 $ conda create --name projeteuromillion python=3.7
```

Maintenant nous allons nous utiliser cet environnement :

```
1 $ conda activate projeteuromillion
```

Maintenant les packages que nous téléchargerons seront installés seulement dans cet environnement python. Nous allons maintenant installer TensorFlow pour gpu : l'option -y permet d'éviter d'avoir à répondre yes lorsque python nous le demande.

```
1 $ conda install -y tensorflow-gpu
```

Le package est installé, on installe aussi la surcouche Keras :

```
1 $ conda install -y keras-gpu
```

On va maintenant procéder à l'installation des packages supplémentaires nécessaires pour utiliser le projet. Certaines sont utiles pour le calcul, d'autres pour récupérer des fichiers et d'autres pour afficher des graphiques :

```
1 $ conda install -yc conda-forge numpy
2 $ conda install -yc anaconda beautifulsoup4
3 $ conda install -yc rmg xlrd
4 $ conda install -yc anaconda pandas
5 $ conda install -yc anaconda requests
6 $ conda install -yc anaconda pydot
7 $ conda install -yc conda-forge matplotlib
8 $ conda install -yc intel scikit-learn
```

Maintenant si on fait un list, on peut voir les packages qui ont été installés :

```
1 $ conda list
```

Pour tester votre installation, exécutez la commande suivante dans l'environnement précédemment créé :

```
1 $ wget https://raw.githubusercontent.com/Thierryfe/DeepLearningEuroMillion/
  master/Documents/Instructions%20d\'installation/testTF.py && python3
  testTF.py
```

Si aucune erreur n'a été détectée, l'installation s'est déroulée correctement. Pour quitter l'environnement dans lequel vous êtes faites :

```
1 $ conda deactivate
```

Il est possible de désactiver l'environnement de base sur le terminal :

```
1 $ conda config --set auto_activate_base false
```

```
(base) etudiant@d20j:~$
```

l'environnement base

Corentin Behuet
Bryan Garreau

Guide utilisateur (Euromillion)

Florian Lherbeil, Thibault Condemine, Thierry Fernandez

Janvier 2020

1 Introduction

Pour utiliser l'application, vous devez récupérer les sources et faire les tutoriel d'installation correspondant à votre distribution. Les sources et le documents sont disponibles sur ce repository Github :

<https://github.com/Thierryfe/DeepLearningEuroMillion>.

L'application est exécutable de deux façons différentes :

1. De façon manuelle : Une fois le script lancé, une série de questions permettant de paramétrer le réseau de neurones seront posées.
2. Avec des arguments : Il est possible d'ajouter des arguments au moment de lancer l'application pour paramétrer le réseau de neurones.

Avant de passer aux étapes suivante, il faut se placer dans le dossier **DeepLearningEuroMillion/Scripts/**

Attention : Lors de la première utilisation, il est très fortement recommandé de télécharger les données

2 Mode manuel

Pour lancer le mode manuel, il faut exécuter la commande suivante (sans aucun argument) :

Attention : les réponses sont sensibles à la casse et toutes les réponses sont obligatoires

```
1 $ python EuroMillionsTestManager.py
```

Ensuite, une série de question seront posées :

- *Voulez vous télécharger (ou mettre à jour) les données de l'Euromillion (y/n) ?*
Répondre y si oui, n sinon
- *Voulez vous utiliser tensorflow ou keras ?*
Répondre tensorflow ou keras
- *Quel réseau de neurones voulez vous entraîner ? RNN ou CNN*
Cette question n'est posée que dans le cas où keras est choisi, dans l'autre cas, le réseau sera automatiquement un RNN
Répondre RNN pour un réseau de neurones récurrents, CNN pour un réseau neuronal convolutif et RNN2 pour une meilleure version d'un RNN.
Si vous avez choisi le réseau RNN2, les questions suivantes seront posées :
 - *Combien de neurones voulez vous sur les couches 2 et 3 ?*
Permet de choisir le nombre de neurones des couches 2 et 3
 - *Combien de tirages voulez vous utiliser en même temps ? (batch size)*
Permet de choisir le nombre de tirages que vous voulez que votre réseau utilise en même temps (batch size), il doit être inférieur au nombre de tirages présents dans le jeu de données
- *Combien voulez-vous d'itérations ?*
Vous pouvez choisir le nombre de cycles d'apprentissage pour votre réseau de neurone (epoch)
- *Veuillez choisir l'intervalle (date entre 2004 et l'année courante) sur lequel vous voulez travailler*
Vous allez choisir successivement l'année de début et l'année de fin des données sur lesquels votre réseau va apprendre. L'année de début doit être antérieure à l'année de fin

3 Utilisation des arguments

Pour utiliser les arguments, il suffit d'exécuter la commande suivante en ajoutant des arguments directement à la suite. Tous les arguments sont **obligatoires et doivent être renseignés dans le bon ordre**.

Exemple :

```
1 $ python EuroMillionsTestManager.py y keras RNN2 64 32 1000 2004 2020
```

Les arguments sont les suivants (dans l'ordre) :

- *download* : Permet de choisir si on veut télécharger (ou mettre à jour) les données des tirages de l'euro-million.
Valeurs possibles : y (pour oui), n (pour non)
- *framework* : Permet de choisir la bibliothèque avec laquelle vous voulez construire votre réseau de neurones.
Valeurs possibles : keras, tensorflow
- *reseau* : Permet de choisir le type de réseau de neurones que vous voulez utiliser.
Valeurs possibles : RNN pour un réseau de neurones récurrents, CNN pour un réseau neuronal convolutif, RNN2 pour une meilleure version de RNN
- *nbNeurones* : Permet de choisir le nombre de neurones des couches 2 et 3 - **Ne sert que pour le réseau RNN2, mettre 0 sinon**
- *nbTirages* : Permet de choisir le nombre de tirages que vous voulez que votre réseau utilise en même temps (batch size), il doit être inférieur au nombre de tirages présents dans le jeu de données - **Ne sert que pour le réseau RNN2, mettre 0 sinon**
- *nbIter* : Permet de choisir le nombre de cycles d'apprentissage pour votre réseau de neurone (epoch)
- *dateDebut et dateFin* : Permet de choisir l'année de début et l'année de fin des données sur lesquels votre réseau va apprendre. L'année de début doit être antérieure à l'année de fin

