

# APPRENTISSAGE PROFOND SUR DES DONNÉES DE JEU VIDÉO

Rubenach Théo

Projet de recherche personnel - Session d'hiver 2018



Encadré par Michel Desmarais

# Contents

<b>1 Présentation générale</b>	<b>3</b>
1.1 Introduction	3
1.2 Un exemple	3
1.3 Les MOBAs	4
<b>2 Revue des solutions existantes (sur League of Legend et DotA)</b>	<b>4</b>
2.1 Dota 2	4
2.2 Autre Moba	6
2.3 Choix des technologies	6
<b>3 Api Steam et récupération des données</b>	<b>6</b>
3.1 package dota2api	7
3.2 Constitution de la base de données	7
<b>4 Prédiction de l'issue du match</b>	<b>7</b>
4.1 Définition d'une baseline	8
4.2 Première approche : réseau neuronal simple	9
4.2.1 Impact du drop-out	9
4.2.2 Etude de la taille du modèle	9
4.3 Deuxième approche : ajout d'un embedding	11
4.3.1 Rôles dans l'équipe	11
4.3.2 Keras embedding	12
4.3.3 Word2Vec	12
4.4 Implémentation des <i>embeddings</i> et tests	13
4.4.1 Impact des différents paramètres	13
4.4.2 Vérification de la pertinence du modèle Word2Vec	13
4.4.3 Impact sur la prédiction de parties	14
<b>5 Limitations et pistes d'élargissement</b>	<b>15</b>
5.1 Approche utilisateur	15
5.2 Utilisation du niveau de jeu	15
5.3 Utilisation de l'ordre des picks, et bans	16
<b>6 Obstacles rencontrés</b>	<b>16</b>
6.1 Instabilité de l'API	16
6.2 Cuda	17
6.3 Limitations de l'envergure du projet	17
6.4 Résultats obtenus	17
<b>7 Apprentissages durant le projet</b>	<b>17</b>
7.1 Apprentissages méthodologiques	17
7.2 Apprentissages techniques	18
<b>8 Conclusion</b>	<b>18</b>
<b>9 Annexes</b>	<b>18</b>
9.1 Code développé	18
9.2 Données renvoyées par l'API	18
9.2.1 Informations sur les matches	18
9.2.2 Informations sur les joueurs	20

# 1 Présentation générale

## 1.1 Introduction

Les dernières années ont vu surgir de nombreux sites web et outils logiciels destinés aux amateurs de jeux vidéo, du joueur occasionnel du dimanche aux joueurs professionnels. Bien que les forums, encyclopédies ou même plug-ins – et avant eux les magazines spécialisés – fournissent une aide au joueur depuis plusieurs décennies, les outils disponibles ont longtemps été artisanaux, développés et maintenus par des joueurs passionnés. La démocratisation du jeu vidéo, en particulier sur ordinateur – les consoles étant par nature moins voire pas accessibles au développement d'outils annexes – ainsi que l'émergence de l'e-sport en tant que marché ont fourni l'impulsion nécessaire au développement de cet écosystème. Les éditeurs de jeux vidéo encouragent pour la plupart cette dynamique en mettant à disposition des données relatives aux joueurs et au jeu lui-même, à travers des APIs disponibles pour tout entrepreneur passionné ou développeur curieux. On ne compte donc plus le nombre de jeux disposant d'applications ou de sites dédiés compilant tout ce qu'il y a à savoir sur les donjons de *World of Warcraft* ou sur la culture de la betterave dans *Farming Simulator 2018*.

Plusieurs entreprises se sont d'ailleurs situées sur ce marché, que ce soit en mettant en relation des joueurs désirant échanger des objets acquis dans l'univers du jeu (<https://loot.farm/>, <https://www.lootmarket.com/>), en agrégeant des statistiques sur le jeu et les joueurs sur un portail web (<https://lolprofile.net/fr/>), ou encore en fournissant une assistance directe au joueur durant les sessions de jeu (<http://www.eblur.co.uk/>).

Il y a donc une manne de données à exploiter d'un côté, et beaucoup d'applications possibles de l'autre. Nous nous proposons donc de construire notre propre application, dont le but sera de recommander des héros aux joueurs de DotA2 (voir ci-dessous) à l'aide d'algorithmes d'apprentissage profond. En effet, DotA2 est un jeu complexe, sur lequel d'importantes masses de données sont disponibles, ce qui fait de l'apprentissage profond un outil intéressant. Après avoir passé en revue les outils existants, nous nous pencherons dans un premier temps sur la constitution d'une base de données suffisante. Nous essaierons ensuite de construire un modèle prédisant la victoire à partir de la composition des équipes d'un match, ceci dans le but de déterminer s'il existe une corrélation entre choix du héros et chances de victoire (indépendamment du joueur). Notons au passage qu'un tel outil serait déjà extrêmement utile, car il pourrait par exemple être utilisé pour les paris en ligne (autre domaine en plein essor dans le domaine de l'e-sport) Enfin, en se basant sur nos résultats, nous évoquerons les limitations et pistes possibles pour continuer ce travail.

## 1.2 Un exemple

L'exemple le plus emblématique des outils actuellement disponibles est peut-être [Hearthstone Deck Tracker](#) (voir figure 1). Hearthstone est un jeu vidéo de cartes à jouer édité par Blizzard, au succès mondial depuis sa sortie en 2014. Les joueurs construisent leur deck en choisissant 30 cartes parmi des centaines de cartes différentes, et s'affrontent ensuite en duel. Chaque joueur dispose à son tour d'une certaine quantité de ressources, qu'il peut dépenser pour jouer des cartes depuis sa main et ainsi espérer prendre l'avantage sur son adversaire.

Dans la mesure où il existe plusieurs centaines de cartes différentes (plus de 1000 à l'heure actuelle), il est nécessaire pour progresser de non seulement connaître toutes les cartes et leurs effets, mais encore de connaître les tendances du moment : quelles sont les cartes à la mode, et comment peut-on les contrer. Enfin, une partie dure en moyenne un quart d'heure, il est nécessaire de se rappeler les cartes déjà jouées afin d'adapter ses choix en conséquences.

*Hearthstone deck tracker* permet de réaliser ces trois fonctions en même temps : en plus d'une base de données sur les cartes disponibles, le logiciel agrège les parties jouées par de nombreux joueurs autour du globe et en tire des statistiques permettant d'adapter son jeu. De plus, un *overlay* (affichage par dessus celui du jeu) permet au joueur de savoir durant sa partie les cartes que lui et son adversaire ont déjà joué, entre autres.

On notera cependant que *Hearthstone Deck Tracker* se limite à agréger des statistiques et faciliter la prise d'information pendant le jeu.



Figure 1: Figure représentant les possibilités offertes durant le jeu par Hearthstone Deck Tracker. Image tirée du site officiel.

### 1.3 Les MOBAs

On a vu avec l'exemple de *Hearthstone Deck Tracker* comment un logiciel pouvait venir en aide au joueur en lui donnant des clés pour améliorer ses choix en amont et pendant la session de jeu. Cependant, les joueurs intéressés par de tels outils sont le plus souvent des joueurs réguliers, qui souhaitent s'améliorer. Cela implique que ce genre d'outil n'est intéressant que dans la mesure où le jeu présente une part de compétition importante.

C'est le cas par exemple des MOBAs (Multiplayer Online Battle Arena). Dans ce type de jeu, dont League of Legend et DotA2 sont les deux plus grands représentants, deux équipes de 5 joueurs s'affrontent dans des matchs de 30minutes à une heure. Lors d'un match, chaque joueur choisit un personnage unique, aux caractéristiques et compétences propres, qu'il incarnera pendant toute la partie. Le but du jeu est de détruire le bâtiment principal de la base ennemie. Cette base étant défendue, les joueurs doivent dans un premier temps augmenter la puissance de leur héros, en gagnant des points d'expérience et des pièces d'or (en tuant des héros adverses ou des monstres dans l'arène) (voir figure 3).

Le choix du héros (voir figure 2) est donc déterminant et influe grandement – toutes choses égales par ailleurs – sur l'issue de la partie.

## 2 Revue des solutions existantes (sur League of Legend et DotA)

### 2.1 Dota 2

De nombreux sites tels [Dotabuff](#) ou [Yasp](#) agrègent les données disponibles via l'API de Valve (société éditrice de DotA2) et en extraient des statistiques utiles ou intéressantes pour les joueurs. Ces sites fournissent souvent également des outils sommaires pour savoir quel héros est efficace contre qui. Cependant leur fonctionnement est sommaire : pour savoir si le héros X est efficace contre le héros Y, l'algorithme se contente de filtrer les matches où les deux héros sont présents (chacun dans une équipe) et de calculer ensuite le taux de victoire moyen sur cet ensemble de matches. C'est le cas par exemple de l'outil présent sur Dotabuff, qui est pourtant l'utilitaire le plus utilisé par les joueurs.

D'autres sites sont spécialisés dans la recommandation de héros proprement dite. Tous ne donnent pas accès à leur algorithme de recommandation, cependant on peut citer les sites suivants :

- <http://dotateam.me/> se concentre sur la composition d'une équipe équilibrée, mais ne prend pas en compte les choix de l'équipe adverse. De plus, l'outil n'est plus développé depuis 2014 et n'est plus donc du tout à jour. Il possède cependant une caractéristique intéressante :



Figure 2: Ecran du choix du héros. En haut, les choix des joueurs des deux équipes (pour l'instant, seul le joueur bleu a préselectionné un héros), ainsi que le temps restant pour faire son choix. Sur la droite, un résumé du héros préselectionné. Au centre, l'ensemble des héros disponibles. En bas, une aide à la composition de l'équipe indiquant dans quels domaines la composition actuelle est la plus forte.



Figure 3: Une capture d'écran d'une partie de DotA2. Dans la partie basse, en 1, la mini-carte montrant le terrain. En 6, la miniature du héros choisi par le joueur, ses statistiques (barre de vie, attaque, vitesse de déplacement, recharge des capacités en cours...). En 7 et en 8, respectivement, le niveau et la quantité d'or amassé par le joueur. On note également les objets achetés par le joueur juste à gauche du chiffre 8. Sur la partie centrale de l'écran, en 2 et 4, les héros des deux équipes s'affrontant. En 3 et 5, les miniatures des héros des deux équipes indiquant le temps restant pour les héros morts avant de pouvoir revenir dans la partie. En 10, un bâtiment d'une des deux équipes. En 9, des monstres de l'arène.

les recommandations du système sont affinées par le vote des utilisateurs, ce qui, une fois atteint une masse critique, devrait filtrer les erreurs. Cependant ce filtrage semble être fait *a posteriori* plutôt qu'utilisé comme feedback pour l'algorithme.

- <https://truepicker.com/en/> a l'avantage de fournir des explications sur les synergies possibles entre différents héros. Ces synergies sont nécessairement entrées à la main (tout le site est d'ailleurs tourné vers le système collaboratif, avec possibilité pour qui veut de laisser une guide ou un conseil sur tel ou tel héros).
- <https://dota2.becomegamer.com/> dispose d'un *overlay* et d'une application mobile, ce qui en fait un outil plus facile à utiliser. Cependant leur algorithme est souvent pris en défaut, bien que le site affirme qu'il est basé sur une grosse base de matches.
- [dotapicker.com](http://dotapicker.com) est certainement l'outil le plus complet et le plus puissant. Il fournit, en plus des recommandations, beaucoup d'informations supplémentaires telles que la courbe de probabilité de victoire au fil du temps (afin de savoir si l'équipe est composée de héros fonctionnant mieux en fin de partie ou pas).

Les deux seuls outils proposant une approche basée sur le *machine learning* à proprement parler sont [Feedless](#) et [Dota+](#). Les deux sont bien plus poussés que les solutions listées ci-dessus, mais elles ont le désavantage de ne pas être open-source contrairement à certains des outils cités plus haut. De plus, les deux sont payantes : Feedless est commercialisé par eblur et Dota+ par Valve, société détentrice des droits de DotA2. Les deux outils combinent la plupart des fonctionnalités possibles en termes de recommandation : recommandations de choix de héros, d'objets et de compétences notamment. Notons que Feedless semble à l'abandon depuis plusieurs mois, et que Dota+ n'est disponible que depuis le mois de mars.

## 2.2 Autre Mobas

En ce qui concerne les autres principaux Mobas, à savoir League of Legend et Heroes of the Storm, il existe également des outils semblables, tels que [pickforme](#), <http://matchup.gg/> pour League of Legend ou encore [heroescounters](#) pour Heroes of the storm. Cependant, à ma connaissance, tous les outils dont j'ai pu tester le fonctionnement pour ces jeux se limitent à calculer le héros maximisant le *winrate* (taux de victoire) d'après la base de données de matches. De plus, n'étant pas joueur suffisamment régulier de ces jeux, j'ai préféré me concentrer sur Dota2 sur lequel ma maîtrise est plus que suffisante pour pouvoir juger des recommandations de différents modèles. Notons toutefois que Riot Games, l'éditeur de League of Legends, fournit une API bien mieux documentée que celle de Valve pour Dota 2, mais nous y reviendrons.

## 2.3 Choix des technologies

Le choix du langage de programmation s'est naturellement porté vers python dans la mesure où :

- c'est un langage très courant, en particulier dans le domaine du *machine learning*,
- Je possépais déjà une certaine connaissance de Python et des librairies de *deep learning* en Python,
- Il existe un wrapper Python simplifiant les envois de requêtes à l'API Steam (même si celui-ci n'est plus activement développé)

En ce qui concerne le choix de la librairie pour développement de modèles neuronaux, j'ai choisi Keras, qui est une librairie haut niveau pouvant fonctionner avec 3 backends : Theano, CNTK ou Tensorflow. Ce dernier étant le plus répandu (et également développé par Google), j'ai préféré celui là.

# 3 Api Steam et récupération des données

La première étape, qui est souvent bloquante, dans une approche de type *apprentissage profond*, est d'obtenir une base de données suffisamment importante pour pouvoir entraîner les réseaux de neurones. Dans notre cas, la constitution d'une base de données de taille suffisante est rendue possible par l'existence d'une API fournie par Steam (éditeur de Dota 2). Un descriptif des informations disponibles pour un match ou un joueur donnés est disponible en annexe.

### 3.1 package dota2api

Après recherches, il existe des wrappers open source de l'API dans de nombreux langages, ce qui facilite la récupération de données. Le choix pour le développement s'étant porté sur Python, nous avons utilisé le package dota2api, installable via pip et dont le code est visible sur [Github](#). Ce package se charge de construire et d'envoyer la requête. Le serveur renvoie les données en format JSON. Elles sont ensuite converties en un dictionnaire.

Une autre solution aurait pu être de construire l'URL "à la main", puis de l'envoyer au serveur à l'aide d'un package tel que `urllib2`.

### 3.2 Constitution de la base de données

Bien que cela ne soit pas indiqué dans la documentation officielle, le serveur limite le taux de requêtes à environ 1 par seconde. Un taux de requête supérieur finit par déclencher différentes erreurs allant du *timeout* au blocage temporaire de connexion. L'API est également régulièrement indisponible pendant plusieurs heures, instabilité dont souffrent même les plus gros sites utilisant cette API (par exemple [ce post](#) sur Dotabuff).

Chaque requête se fait par un appel à `get_match_history_by_seq_num`. Cet appel renvoie les données principales (non détaillées) des matches à partir d'un match donné, en ordre chronologique. Bien que la documentation n'indique pas de limite au nombre de matches dont on requiert les données, le cas pratique est bien différent : le JSON est vide au delà du centième match, quel que soit le nombre de matches demandés. Il faut donc réaliser de nombreux appels avant d'obtenir une base de taille correcte. De plus, le résultat renvoyé à l'utilisation de certaines clés (comme par exemple la clé permettant de sélectionner les matches dans une certaine fenêtre temporelle) est incohérent ou carrément inexistant (le problème semblant parfois dater de plusieurs années d'après divers forums). Enfin, l'API est particulièrement instable; ainsi, le même appel échoue souvent à un instant  $t$  pour réussir à l'instant  $t + 1$ ; une rapide estimation statistique sur le résultat d'une heure de requêtes (à raison d'une requête par seconde) a donné un pourcentage de réponses valides de 42% environ.

Malgré les différents problèmes dont nous venons de citer quelques exemples, après plusieurs semaines de requêtes, nous avons fini par obtenir les données de plus de 4 millions de matches, totalisant plus de 15Go de données brutes. Notons que cela est peu comparé à l'historique de tous les matches joués depuis le premier jour d'existence du jeu, puisque la somme totale des données (en dehors des *replays*) récupérables via l'API est estimée à plus de 6To, pour plus de  $1.1e^9$  matches. Cependant, notre but étant de conseiller des héros aux joueurs, ou dans un premier temps de déterminer l'issue du match, les matches récents sont bien plus pertinents puisque le jeu a énormément évolué en 6 ans d'existence. Ainsi, beaucoup de héros ont vu leur rôle changer du tout au tout, et un certain nombre de héros ont été ajoutés au jeu. Un modèle appris sur les parties de 2012 ne serait aujourd'hui plus guère performant. Nous nous sommes donc limités aux parties jouées depuis le premier janvier 2018, dans la mesure où il n'y a pas eu de grosses mises à jour dans le jeu depuis; et en pratique, les 4 millions de matches, bout-à-bout, doivent couvrir une fenêtre temporelle d'environ une dizaine de jours.

De plus, il existe plusieurs modes de jeu, dont certains où il n'est pas possible de savoir à l'avance quels seront les choix de l'équipe adverse, voire même un mode où le choix du héros est totalement aléatoire. Ces modes de jeu tombant naturellement en dehors des objectifs de ce projet, nous nous sommes donc limités aux matches en mode dit "All pick" : tous les héros peuvent être choisis librement. Cela divise par deux le nombre de matches dans la base de données, et nous obtenons donc une base finale d'un peu plus de 2 millions de matches. Enfin, le rôle des deux équipes étant parfaitement interchangeable, nous pouvons facilement doubler la taille de notre base de matches en inversant les deux équipes et l'issue du match pour obtenir le match 'symétrique' d'un match donné. Cela nous permet donc de disposer de 4 millions de matches.

## 4 Prédiction de l'issue du match

Avant d'essayer d'utiliser les données des matches pour conseiller des héros aux joueurs, il faut tout d'abord s'assurer que les données contiennent des informations utiles : le meilleur modèle au monde ne saurait trouver des règles pertinentes si les données n'en contiennent pas ! Dans cette optique, nous avons tout d'abord essayé de prédire l'issue d'un match à partir de la composition des équipes. Il est évident qu'il est impossible d'atteindre un score parfait sur ce problème dans la mesure où

Nombre de matches considérés	Test	Validation
1000	54.2 %	52.1%
100000	56.6%	56.4 %
500000	56.4 %	57%
1000000	56.9 %	57.0 %
2000000	57.5%	57.1%

Figure 4: Pourcentage de performance de la baseline en fonction du nombre de matches considérés. 20 % de la base est choisie aléatoirement comme ensemble de validation. A partir de 100000 matches environ, on note que le nombre de matches importe peu. L'écart non significatif entre ensemble d'entraînement et de validation s'explique par le fait que cette baseline "n'apprend" rien, elle se contente d'appliquer la distribution du taux de victoire. Ceci explique aussi les plus faibles performances pour 1000 matches : il n'y a pas assez de matches pour avoir des *winrates* significatifs.

cela signifierait que seul le choix des héros détermine l'issue de la partie; cependant on espère trouver une précision supérieure à 50% : cela montrerait que le choix des héros a effectivement un impact sur l'issue de la partie, indépendamment du joueur.

Si malgré tout on ne parvient pas à dépasser les 50% de précision, cela voudra dire qu'un modèle devra obligatoirement prendre en compte le profil de l'utilisateur pour être utile (si on considère qu'utilile signifie augmenter le taux de victoire). Cependant, utiliser les données de l'utilisateur est bien plus délicat dans un cadre d'apprentissage profond, car le modèle devient bien plus complexe du fait de la plus grande hétérogénéité des données sur les utilisateurs (nombre de parties jouées, répartition égale ou non des héros joués, du type de héros joué, niveau de compétence...). De plus, les données relatives aux utilisateurs ne sont pas publiques par défaut, et le dataset serait forcément bien plus réduit. Dans un premier temps, nous laisserons donc les données spécifiques à un utilisateur de côté pour nous concentrer uniquement sur les données relatives aux matches.

#### 4.1 Définition d'une baseline

Nota Bene : Nous utiliserons indépendamment les termes "précision" et "performances" pour évaluer les résultats fournis par les différents réseaux entraînés. En effet, dans la mesure où la prédiction est binaire, calculer la précision sur les deux classes suffit à évaluer la qualité du modèle. Ceci est valable dans la mesure où les deux classes (victoire et défaite) sont également réparties, et que prédire l'une revient à prédire l'autre dans la situation symétrique (inversion des deux équipes), ce qui est le cas ici.

Afin de pouvoir comparer les résultats obtenus à l'aide des différents modèles de réseaux de neurones, il est nécessaire de construire une baseline qui nous servira de référence. En ce qui concerne la prédiction de l'issue du match, une baseline saute immédiatement aux yeux : le choix aléatoire, qui obtient un score de 50%. Cependant, cette baseline ne présente que peu d'intérêt car trop triviale. Une deuxième baseline plus complexe est donc construite. Cette deuxième baseline se calcule comme suit. En notant 1 le fait que l'équipe 1 gagne, et 0 sinon, on a :

$$\hat{y} = \operatorname{argmin}_{x \in 1,2} \sum_{h \in H_x} w_h,$$

où  $x$  désigne l'indice de l'équipe (1 ou 2),  $H_x$  est l'ensemble des héros choisis par cette équipe, et  $w_h$  est le taux de victoire moyen du héros  $h$  (usuellement nommé *winrate*). En effet, tous les héros n'ont pas le même *winrate* : ceux qui sont plus délicats à jouer sont souvent légèrement en-deçà, car seuls les bons joueurs les maîtrisent. De plus, dans la mesure où le jeu est en constante évolution, il existe toujours certains héros un peu plus puissants que les autres, indépendamment du joueur qui les manie. Ainsi, les héros les moins en vogue peuvent avoir un *winrate* aux alentours de 45%, quand les meilleurs héros du moment sont fréquemment au-delà de 55%.

La baseline réalise un score étonnamment haut (voir figure 4), dans la mesure où 7% de *winrate* au dessus de la moyenne est exceptionnel (l'immense majorité des joueurs ont un *winrate* compris entre 48% et 52%). Il semblerait donc qu'en moyenne, choisir les meilleurs héros du moment soit déjà une bonne méthode pour gagner. Cependant il y a fort à parier que cela ne fonctionne plus avec des joueurs expérimentés (et qui savent donc comment contrer les meilleurs héros).

## 4.2 Première approche : réseau neuronal simple

La première famille de modèle que nous allons tester est celle des perceptrons multi-couches. Dans ce genre de réseau, nommé également *fully connected*, chaque neurone d'une couche prend ses entrées de tous les neurones de la couche précédente. C'est le type de réseau le plus simple, mais il est ici bien adapté puisque nos données ne sont pas organisées spatialement ni temporellement. La couche d'entrée du réseau comporte 121 neurones (un par héros possible). Notons qu'il n'y a pas exactement 121 héros, mais l'indice par lequel un héros est représenté dans l'API va de 1 à 121. Utiliser 121 neurones est donc un pur artifice, dans la mesure où certains neurones ne seront jamais activés.

Le vecteur représentant un match comporte donc 122 valeurs pour les 121 héros 'possibles'. La valeur  $i$  du vecteur d'entrée vaudra 0 si le héros numéro  $i$  n'a pas été choisi du tout, 1 s'il a été choisi par la première équipe et -1 s'il a été choisi par la deuxième équipe. On gardera également la trace de l'équipe gagnante par un *float* valant 1 ou 0.

La sortie du réseau sera une valeur comprise entre 0 et 1, en fonction de l'équipe gagnante prédite. L'erreur utilisée est une entropie classique :

$$e = \frac{1}{N} \sum_i^N y_i * \ln(\hat{y}_i),$$

où  $N$  est la taille du mini-batch,  $y$  la vérité et  $\hat{y}$  la prédiction.

L'algorithme utilisé pour la rétro-propagation dans le réseau est Adam. C'est l'algorithme le plus utilisé dans le cas général. Il permet d'avoir un taux d'apprentissage (*learning rate*) adapté à chaque poids en fonction des moments d'ordre 1 et 2 (moyenne et variance) des erreurs des époques d'entraînement passées. Cet algorithme intègre une notion de *momentum*, qui permet d'accélérer l'entraînement, et réduit également progressivement son *learning rate* en fonction de la progression du réseau (haut taux au début, quand le réseau apprend vite, plus bas réduit au fur et à mesure que l'on affine l'entraînement).

Enfin, pour accélérer l'entraînement, on utilisera des mini-batches de taille 128, car nos données sont peu volumineuses; on peut donc utiliser sans problème des batches de cette taille. On pourrait prendre une taille encore bien supérieure, de l'ordre de  $10^3$ , mais on prend alors le risque de "lisser" les données d'entrée. Cela se comprend comme suit : dans le cas extrême où le mini-batch fait la taille de l'ensemble d'entraînement, le réseau aurait de grande chances de tomber dans un minimum local qui correspondrait à la situation "apprendre la réponse parfaite pour la moyenne de l'ensemble d'entraînement".

### 4.2.1 Impact du drop-out

Il est parfois utile d'utiliser la technique du drop-out lors de l'entraînement d'un réseau pour augmenter sa capacité à généraliser. En effet, un phénomène souvent rencontré est celui du sur-apprentissage, où le réseau apprend la distribution spécifique à l'ensemble d'entraînement plutôt que la distribution des données dont est issu cet ensemble. Ce phénomène se caractérise par des performances sur l'ensemble d'entraînement qui continuent à augmenter alors que les performances sur l'ensemble de validation diminuent. Le drop-out consiste à "éteindre" de manière aléatoire une proportion fixe des connexions du réseau (typiquement entre 20% et 50%).

Cependant, en conduisant nos tests, si le drop-out diminue le sur-apprentissage (l'écart entre précision sur l'ensemble d'entraînement et précision sur l'ensemble de validation est plus faible), il n'augmente pas de manière significative la précision du réseau sur l'ensemble de validation. Comme on peut le voir sur la figure, les performances sur l'ensemble de validation de la plupart des réseaux testés connaissent un pic avant de décroître. L'utilisation du drop-out diminue ce phénomène, mais n'améliore pas la précision maximale atteinte sur l'ensemble de validation.

On n'utilisera donc pas de drop out.

### 4.2.2 Etude de la taille du modèle

Nous allons tester 3 modèles construits de la même manière mais de tailles différentes. La structure du réseau peut être trouvée sur la figure 5. Le plus petit réseau comporte 7 couches pour 173000 paramètres, le réseau intermédiaire en possède 9, plus grosses, pour un total de 1028000 paramètres et enfin le plus gros réseau totalise 3600000 paramètres répartis sur 12 couches.

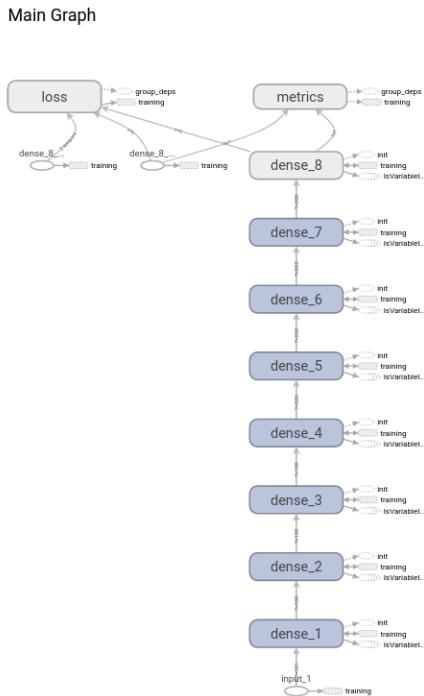


Figure 5: Structure du réseau *fully connected* utilisé. Image générée à l'aide de tensorboard. Il s'agit ici du réseau le plus petit, mais les autres sont construits sur le même schéma.

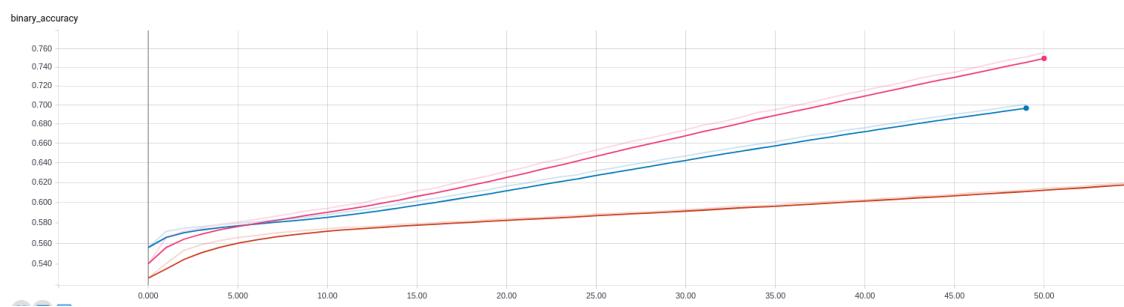


Figure 6: Précision des trois modèles (petit, moyen, grand) sur l'ensemble de test en fonction du nombre d'époques. Les 3 courbes estompées sont les vraies courbes; celles en couleurs vives sont légèrement lissées. En magenta la courbe du plus petit réseau; en bleu celle du réseau de taille moyenne et en brun la progression du plus gros réseau. Le plus petit réseau s'entraîne logiquement plus vite. Notons que si les trois réseaux ont été entraînés une cinquantaine d'époques, cela correspond pour le petit réseau à 5 minutes et à 24h pour le gros. Image générée à l'aide de tensorboard.

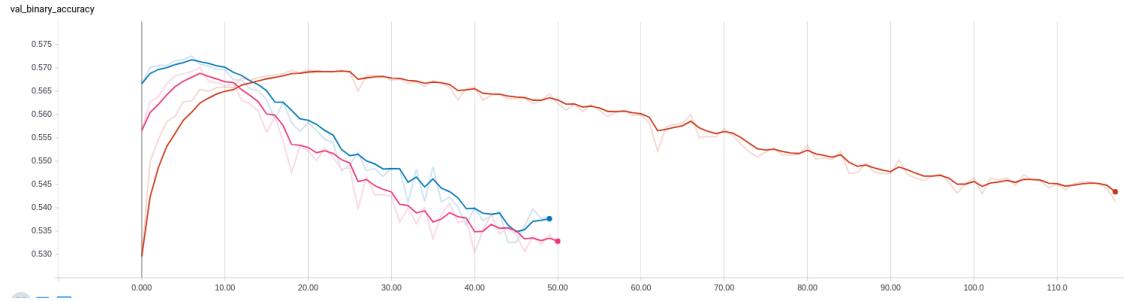


Figure 7: Précision des trois modèles (petit, moyen, grand) sur l'ensemble de validation en fonction du nombre d'époques. Même code couleur que précédemment. Notons qu'après un pic atteint assez rapidement à 57% environ, le réseau entre en phase de sur-apprentissage. Notons que le gros réseau connaît ce pic plus tard, ce qui est logique puisqu'il s'entraîne plus lentement également. Image générée à l'aide de tensorboard.

On constate (voir figures 6 et 7) un comportement exactement semblable pour les trois modèles, quelle que soit leur taille. Notre première approche nous permet donc déjà de conclure qu'il n'y a pas de corrélation cachée dans les données autre que celle, évidente, utilisée pour la baseline; ou du moins s'il y en a une, qu'un modèle aussi simple que celui *fully connected*, quel que soit sa taille, ne parvient pas à la trouver.

### 4.3 Deuxième approche : ajout d'un embedding

On a vu que la première famille de modèles était globalement équivalente à la baseline choisie, qui était pourtant relativement simple. Si ce résultat n'est pas pour autant mauvais dans la mesure où la baseline est déjà assez haute (passer de 50 à 57% de taux moyen de victoire serait déjà exceptionnel), il est un peu décevant.

Nous allons donc essayer de complexifier l'approche pour tenir compte des spécificités propres à chaque héros. En effet, une équipe bien composée doit être équilibrée, et choisir 5 héros au hasard ne donnera que très rarement une bonne composition. Nous allons maintenant expliquer rapidement les différents rôles dans une équipe.

#### 4.3.1 Rôles dans l'équipe

Les rôles dans une équipe se décomposent (grossièrement) comme suit :

- Le *carry* est (sont) le(s) héros le(s) plus important(s) dans l'équipe, il est d'ailleurs nommé "position 1" (ou "position 1" et "position 2"). C'est en général un héros particulièrement fort en fin de partie, car il a besoin de niveaux et d'équipement pour être efficace. Il sera alors la condition sine qua non qu'un combat réussi contre l'équipe adverse. Comme ce genre de héros est faible en début de partie, il faudra l'entourer de héros qui eux sont efficaces en début de partie pour qu'il puisse gagner ces niveaux et cet équipement convenablement. Ces héros davantage efficaces en début de partie sont nommés
- *supports*. Généralement au nombre de deux, ce sont des héros qui n'ont pas besoin de beaucoup d'équipement pour être efficaces. Ils peuvent donc se concentrer sur les objets utiles à toute l'équipe, et sur la survie du ou des *carry*. En ordre d'importance, ils généralement en position 4 et 5.
- Entre ces deux extrêmes se trouvent les positions 2 et 3 (s'il y a un *carry* et deux supports) ou 3 et 4 (s'il y a deux *carry* et un support). Ces héros sont plus versatiles et peuvent être efficaces avec ou sans équipement, en adaptant bien évidemment leur style de jeu en conséquence.

De plus, outre la position du héros, certains héros ont des capacités les rendant extrêmement efficaces en duo; d'autres sont davantage taillés pour être joués seuls en début de partie...

On voit donc que notre première approche négligeait totalement cet aspect du jeu.

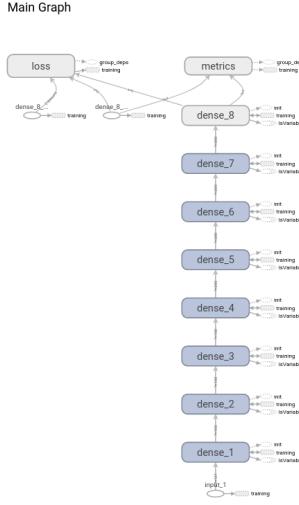


Figure 8: Structure du réseau de base avec embeddings. La structure avec els embeddings de Keras ou de Gensim a la même structure.

Nous allons donc essayer d'utiliser un *embedding* (ou plongement en français) de type *Word2Vec*, c'est-à-dire de projeter chaque héros (qui était jusque là représenté par un vecteur de 120 zéros avec un seul 1) dans un espace de haute dimension, où sa représentation sera apprise de manière automatique.

#### 4.3.2 Keras embedding

Keras propose déjà une couche appelée "embedding" qui semble réaliser ce que l'on souhaite, c'est-à-dire que cette couche projette l'entrée qu'on lui passe dans un espace de dimension donnée (voir figure 8).

Cependant les tests préliminaires n'ont pas été concluants (les différents réseaux n'apprenaient rien), et pour cause : cette couche étant intégrée au réseau telle quelle, elle est sujette à la rétro-propagation du réseau, et non pas à une erreur de type softmax calculée en sortie de la couche d'embedding comme c'est le cas pour la technique *Word2Vec*. Les *embeddings* calculés n'ont donc rien à voir avec la logique de *Word2Vec*.

#### 4.3.3 Word2Vec

Word2Vec est une méthode ( Efficient Estimation of Word Representations in Vector Space, 2013) conçu au départ pour apprendre une représentation algébrique (sous forme de vecteur) d'un ensemble de mots à partir d'un corpus de texte (d'où son nom). L'idée est de construire la représentation de chaque mot en considérant la relation entre la représentation d'un mot et la représentation de son contexte. Cela est réalisé grâce à un modèle de réseau de neurones à 3 couches dont une couche cachée. Deux structures de réseau sont possibles : skip-gram ou CBOW (pour Continuous Bag Of Words). L'un essaie de prédire la représentation du mot à partir de son contexte, l'autre est exactement l'inverse puisqu'il prédit la représentation du contexte à partir du mot. Une rétro-propagation de l'erreur permet classiquement au réseau d'apprendre les représentations. Nous utiliserons ici CBOW puisque nous cherchons à prédire le héros à partir du contexte. A partir de la représentation d'un vocabulaire, beaucoup de choses deviennent possibles. L'exemple le plus connu est qu'en notant  $v$  la représentation d'un mot du vocabulaire dans cet espace algébrique, on peut ensuite avoir des égalités du type

$$v("reine") = v("roi") - v("homme") + v("femme")$$

Ici, l'idée est de considérer chaque héros comme un mot, et chaque partie (10 héros choisis) comme une phrase, c'est à-dire comme le contexte de ce mot. On devrait ainsi obtenir une représentation de chaque héros, et on pourra ensuite vérifier que les plus proches voisins d'un héros de type 'carry' est un autre héros de type 'carry', par exemple.

Une subtilité apparaît cependant ici : dans la mesure où on souhaite utiliser les 10 choix de héros, il convient de différencier les héros des deux équipes. En effet, ils n'ont pas du tout le même 'rôle' dans notre 'phrase' ! Garder les 10 choix de héros (c'est-à-dire les deux équipes) est nécessaire car ne garder que les héros de la même équipe reviendrait à essayer de calculer les synergies, mais sans prendre en compte l'adversaire, ce qui serait une erreur ! Pour utiliser les 10 héros, on dédouble également chaque héros, selon qu'il soit dans la première ou dans la deuxième équipe. Notre vocabulaire aura donc 2 mots par héros selon l'équipe dans laquelle il est.

On devrait alors obtenir deux représentations pour chaque héros, celle construite à partir de ses occurrences dans la première équipe, et celle construite à partir de ses occurrences dans la deuxième. Ces deux représentations devraient être symétriques l'une de l'autre.

## 4.4 Implémentation des *embeddings* et tests

Au vu de la simplicité du réseau utilisé pour construire une représentation Word2Vec, il aurait tout à fait été possible de le développer en Keras (ou d'en trouver une implémentation Keras sur Internet). Cependant, (échaudés par les *embeddings* proposés par Keras), il existe une librairie python très efficace et spécialisée dans ce genre d'algorithmes, nommée *gensim*. Comparativement à une approche Keras, *gensim* permet d'utiliser directement des mots au format 'humain' et non des vecteurs, les résultats sont donc plus facilement exploitables sans avoir à développer une interface entre un format compréhensible par l'homme et un format compréhensible par Keras. Enfin, si *gensim* n'utilise pas d'accélération GPU, cela n'est pas dérangeant puisque j'ai également renoncé à utiliser ces accélérations sur Keras.

### 4.4.1 Impact des différents paramètres

Les deux principaux paramètres pertinents lors de la construction de la représentation Word2Vec sont le nombre de parties utilisées pour construire la représentation et la dimension de l'espace de plongement. En effet, le troisième paramètre primordial pour Word2Vec est la taille du contexte à prendre en compte, mais ici dans la mesure où toutes nos 'phrases' font la même taille, il suffit de régler la taille de la fenêtre à la taille de la 'phrase' soit 10 (si on utilise la représentation de base) ou 240 (2\*120, si on utilise la représentation utilisé pour les MLP mais 'dédoublée').

Concernant le nombre de parties, notons tout d'abord qu'il doit être suffisamment important pour que tous les héros soient joués au moins une fois. Dans la mesure où les héros les moins joués ont un taux de *pick* (c'est-à-dire une probabilité d'être présent dans une partie choisie au hasard) largement inférieur à 1% –à comparer au 30% de taux de *pick* pour les héros les plus populaires du moment –, cela porte déjà le nombre minimal de partie à plus d'un millier, pour être raisonnablement sûr de voir chaque héros. Cependant, une seule occurrence est bien trop léger pour construire une représentation fiable; il faudra donc utiliser des ensembles bien plus gros pour être certains d'avoir une bonne représentation de tous les héros. De manière générale, plus gros est le corpus de parties, meilleure sera la représentation du héros construite. Typiquement, nous avons réalisé nos tests avec au moins 10000 parties, le problème étant qu'utiliser davantage de parties entraîne logiquement une augmentation du temps de calcul, celui ci devenant rapidement non négligeable (de plusieurs minutes à plusieurs heures) à cause du temps d'entraînement du modèle Word2Vec.

### 4.4.2 Vérification de la pertinence du modèle Word2Vec

Pour vérifier que le modèle Word2Vec fonctionne correctement, et que l'approche de plongement à partir des *picks* fonctionne correctement, nous pouvons vérifier quels sont les plus proches voisins d'un héros donné.

Le problème est de pouvoir comparer deux héros entre eux. En tant que joueurs, il nous est possible d'avoir une bonne idée de la pertinence des voisins proposés, mais il serait intéressant d'avoir une approche quantitative. L'approche proposée ici est simple : l'[encyclopédie](#) des héros disponible pour les nouveaux joueurs classe un héros dans plusieurs catégories (figure 9). On peut donc construire une sorte de vecteur de catégories, où chaque valeur vaudra 1 si le héros fait partie de cette catégorie ou 0 sinon.

Dans la mesure où le vecteur doit être construit à la main (puisque ces catégories n'apparaissent que sur le site d'aide aux débutants), nous ne comparerons que sur un héros. Par exemple, pour le premier héros dans l'ordre alphabétique, Anti-Mage, les catégories indiquées sont Corps-à-Corps,



Figure 9: Exemple de catégories dans lesquelles est classé un héros donné. La liste de toutes les catégories est : Carry, Corps-à-corps, Disabler (empêche les héros ennemis d'agir), Distance, Durable (efficace en fin de partie), Escape, Initiator, Jungler (efficace dans la *jungle*, une partie de l'arène), Nuker (gros dégâts en peu de temps), Pusher (utile pour détruire les bâtiments adverses) et Support.

Nom	Carry	Cac	Disab	Dist	Dur	E	I	J	N	P	S	cos dist
Anti-Mage	X	X			X				X			0
Sven	X	X	X		X		X		X			0.18
Spectre	X	X			X	X						0.25
Weaver	X			X		X						0.71
Gyrocopter	X		X	X					X			0.50
Lifestealer	X	X	X		X	X		X				0.38
Shadow Demon				X	X		X		X		X	0.77
Lion					X	X		X	X		X	0.77
Disruptor					X	X		X	X		X	0.77
Storm Spirit	X			X	X		X	X		X		0.59
Rubick				X	X				X		X	0.75

Figure 10: On note une distance bien moindre pour les 5 plus proches voisins, ce qui confirme la méthode et la validité des résultats. Le seul résultat surprenant est celui de Storm Spirit qui, étant un héros de type *carry*, aurait été attendu à minima dans le milieu du tableau mais certainement pas à la fin ! En ce qui concerne les autres héros, tous font sens : les voisins les plus lointains sont des supports, au style de jeu aux antipodes d'un héros comme Anti-Mage. Weaver possède également une distance relativement élevée, cependant Anti-Mage est un héros extrêmement élusif et aurait pu être faire partie de la catégorie Escape, Weaver quant à elle reste un choix tout à fait valable en fin de partie, même si ce n'est pas la caractéristique principale du héros.

Carry, Escape et Nuker. Les 5 voisins les plus proches, pour 100 itérations sur 1 million de parties, sont dans l'ordre Spectre, Weaver, Gyrocopter, Lifestealer et Ursula. Les 5 héros les plus éloignés sont dans l'ordre Shadow Demon, Lion, Disruptor, Storm Spirit et Rubick. Les résultats sont visibles sur la figure 10.

Une piste pour améliorer ce modèle pourrait être de ne considérer que les compositions d'équipe ayant amené à la victoire; cependant après test les résultats se sont révélés identiques ou presque, les quelques différences constatées dans les plus proches voisins étant très probablement à mettre sur le compte de la part aléatoire du modèle Word2Vec puisqu'elles ne reposaient sur aucune logique apparente. Cette hypothèse s'est vue constatée en traçant le graphe des similarités d'un héros avec les autres (figure 11), puisque l'on y constate que la similarité décroît relativement lentement; il est donc raisonnable de penser que le deuxième voisin le plus proche se retrouve en 4 ou cinquième position sur une autre expérience, dans la mesure où l'écart de score de similarité entre les positions 2 et 5 est relativement réduit.

#### 4.4.3 Impact sur la prédiction de parties

Maintenant que nous disposons d'une représentation pertinente et plus complexe des héros, nous pouvons l'utiliser comme entrée de notre réseau *fully connected*. Le souci est qu'il faut pour cela passer de la représentation d'un héros à la représentation d'une partie (soit 2x5 héros). Notre idée a été d'additionner les représentations des héros d'une équipe. L'idée sous-jacente était de considérer que les héros d'une même équipe se complètent les uns les autres, alors qu'une équipe déséquilibrée aurait ainsi une représentation reflétant ce problème. Pour regrouper ensemble les représentations des deux équipes, deux choix sont ensuite possibles : soit soustraire la représentation d'une équipe à l'autre, en suivant une logique qui est que chaque équipe essaie de contrer l'autre; soit nous pouvons également simplement concaténer les représentations des deux équipes. La deuxième

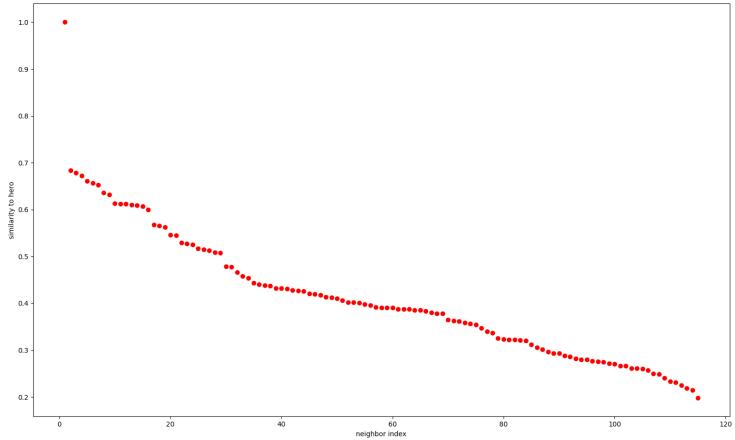


Figure 11: Courbe de la similarité des plus proches voisins en fonction de leur rang pour un héros (Anti-Mage). On note que le plus proche voisin a un score de similarité de 1, ce qui est logique puisque c'est le héros lui-même (qui est donc son plus proche voisin).

approche paraissant plus sûre, c'est celle-là que nous avons privilégiée. Cependant il s'est vite avéré que cela n'est pas concluant du tout, le réseau n'apprenant quasiment rien (figure 12)...

Il apparaît donc que la représentation apprise par le modèle Word2Vec, si elle est pertinente, n'a pas permis de révéler d'éventuels relations dans les données que la baseline n'avait pas déjà utilisées. A ce stade, peut-être serait-il plus pertinent de se diriger vers d'autres données que simplement celles des matches.

## 5 Limitations et pistes d'élargissement

### 5.1 Approche utilisateur

Plusieurs éléments des données, a priori disponibles, n'ont pas été utilisés pendant le projet et aurait certainement pu l'être. Premièrement, il était au début prévu d'utiliser les données des utilisateurs. Cependant, les appels API tels que *getPlayerDetails* sont trompeurs car ils ne renvoient en rien des statistiques relatives aux matches auquel un joueur a participé. La documentation indique par ailleurs qu'il est possible de filtrer les matches en fonction d'un joueur donné, ce qui rendrait cette tâche tout à fait abordable, mais cette fonctionnalité ne fonctionne hélas pas (et semble-t-il n'a jamais fonctionné d'après certains messages trouvés en ligne).

Pour obtenir un profil utilisateur, il faudrait donc filtrer tous les matches pour récupérer ceux où est présent un joueur donné. Dans la mesure où il y a plus de 10 millions de joueurs uniques mensuels, il faudrait parcourir (et donc avoir téléchargé) à minima plusieurs centaines de millions de matches pour obtenir au mieux quelques dizaines de matches pour un joueur donné, et cela en considérant qu'il a été actif sur la période choisie. A titre de comparaison, l'ensemble des matches téléchargés au cours du projet couvre environ deux semaines (de manière discontinue), pour plus de 4 millions de matches.

Il a donc été difficile dans les limites de ce projet d'utiliser une approche utilisateur, et les idées ne manquant pas par ailleurs, d'autres pistes ont été préférées; mais cela reste une dimension à explorer.

### 5.2 Utilisation du niveau de jeu

Il était également prévu en délimitant le projet de tester l'impact du niveau des joueurs sur les résultats, en se basant sur le champ *lobby* qui désigne habituellement dans la communauté le niveau moyen des joueurs d'un match. Cependant, il est apparu en lisant la documentation plus en détail que cela représentait simplement le type de partie, sans aucun lien avec le niveau des joueurs. Après une lecture extensive de la documentation, il s'avère qu'il est impossible d'accéder directement à une métrique qui permettrait d'estimer le niveau moyen de compétence d'un joueur ou des joueurs d'un match. Nous avons donc également dû laisser cette possibilité de côté, même

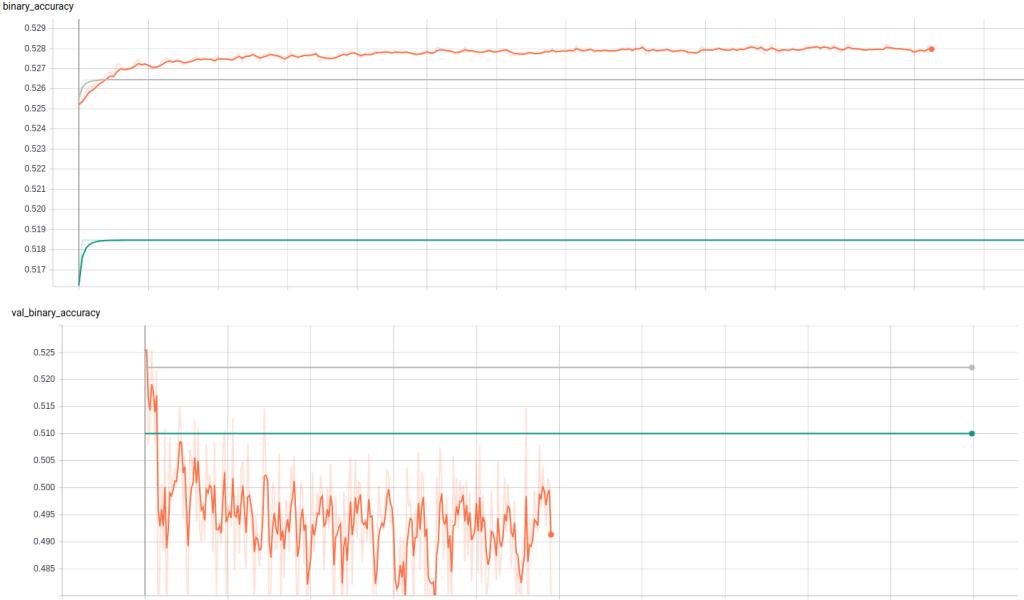


Figure 12: Précision sur les ensembles de test (haut) et validation (bas) pour un modèle 'moyen' avec en entrée les représentations fournies par gensim. En vert, gris et orange, respectivement, représentations apprises avec 10000, 100000 et 1 million de parties. Les performances avec 10000 et 100000 sont étonnamment stables, bien qu'on distingue clairement un apprentissage au début. Peut-être est un problème de chargement des données lié à tensorboard, car le rechargeement du fichier d'historique n'a rien donné. On constate cependant que les performances ne sont pas au rendez-vous : à peine 53% sur l'ensemble d'entraînement et entre 52 et 50% pour l'ensemble de validation.

s'il aurait été agréable de pouvoir confirmer ou infirmer une intuition sur la question qui est que 1- les bons choix à bas et à haut niveau ne sont pas les mêmes et 2- l'impact du choix du héros est beaucoup plus déterminant à haut niveau, où les joueurs sont de niveau plus homogène, qu'à bas niveau où le talent du joueur compte plus que la pertinence du héros qu'il choisit de jouer.

### 5.3 Utilisation de l'ordre des picks, et bans

Un autre aspect qu'il aurait été intéressant de considérer et que nous avons laissé de côté est l'ordre dans lequel les différents joueurs ont choisi leur héros : un héros choisi en premier aura plus de chances de provoquer un *counterpick* de la part de l'équipe adverse; le joueur choisissant en dernier sera sûr de ne pas se faire contrer son héros. Notons d'ailleurs qu'à partir d'un certain niveau, c'est toujours le héros *carry* (donc le plus important) qui est choisi en dernier pour éviter qu'il ne se fasse contrer. Nous l'avons laissé de côté davantage par manque de temps, le sujet étant extrêmement vaste, mais une approche par réseau récurrent de type LSTM pourrait probablement donner des résultats intéressants.

Certains modes de jeu permettent aussi de bannir quelques héros qui ne pourront être joués. Si cela est porteur d'énormément d'information sur la stratégie de choix des héros, ce qui nous aurait été très utile, ce mode de jeu est minoritaire (moins de 10% des parties jouées) et il nous aurait donc fallu récupérer 10 fois plus de données pour arriver à la même base. Nous avons donc préféré nous concentrer sur le type de partie le plus répandu.

## 6 Obstacles rencontrés

### 6.1 Instabilité de l'API

Le premier obstacle (et non le moindre) que j'ai pu rencontré a été l'instabilité de l'API. J'étais certain de pouvoir construire une base de données d'une taille suffisante pour pouvoir entraîner des réseaux de neurones, cependant je ne pensais pas qu'il me faudrait autant de temps pour y parvenir. En effet, les incohérences entre documentation et comportement constaté (voire les bugs)

m'ont obligé à procéder par essai erreur, d'autant que la seule ressource sur le sujet en ligne est un forum en sous-activité depuis 4 ans ([dev.dota2.com](http://dev.dota2.com)). De plus, entre les moments d'indisponibilité et la grande quantité de requêtes non fructueuses, au rythme de 100 requêtes par seconde, il a fallu plus d'une semaine (en cumulé) pour constituer la base de données.

## 6.2 Cuda

Un autre écueil a été l'installation du framework pour l'entraînement de réseaux de neurones. Tensorflow et Keras peuvent utiliser le GPU pour accélérer les calculs, à condition d'avoir une carte graphique Nvidia suffisamment récente. Disposant d'une carte graphique Nvidia GTX675, j'ai tenté d'installer Cuda (et cudnn qui est spécifique au réseaux de neurones), qui est nécessaire à tensorflow pour utiliser le GPU. L'installation de Cuda est connue pour être particulièrement délicate, et après plusieurs jours d'essais infructueux – notamment parce que ma version d'Ubuntu était un peu trop ancienne –, j'ai préféré me concentrer sur le code lui-même plutôt que sur son accélération. En effet, les accélérations possibles en passant sur GPU sont surtout intéressantes sur des modèles de réseaux de neurones convolutionnels (de l'ordre de 10 fois plus rapide), or je n'ai pas utilisé de couches de convolution puisque le problème ne s'y prête pas. Je suis donc resté sur la version purement CPU de Keras / Tensorflow, même s'il aurait été utile de disposer d'un Cuda fonctionnel, ne serait-ce que pour un projet futur.

## 6.3 Limitations de l'envergure du projet

Au moment de démarrer ce projet, je pensais pouvoir constituer facilement une base conséquente de données sur le jeu, puis de tester ensuite de nombreux modèles, sur différentes problématiques, pour arriver à un système de recommandation ou de prédiction performant. Je me suis donc lancé à certains moments dans des idées qui auraient nécessité un temps non négligeable pour être exploitées à fond, pour que finalement seules quelques pistes soient explorées extensivement. Ceci s'explique par le fait que contrairement à un projet dans le cadre d'un cours, par exemple, où le but est clairement défini et les données souvent déjà disponibles, j'ai dû ici partir de zéro. Or développer un pipeline allant de la récupération des données à la formulation d'une prédiction humainement compréhensible en passant par les fonctions nécessaires à l'entraînement d'un réseau de neurones prend du temps, et – à moins d'y passer encore plus de temps – donne souvent un ensemble bien moins robuste.

## 6.4 Résultats obtenus

Le dernier obstacle enfin, qui lui n'a pas été franchi, est la pauvreté des résultats obtenus : je n'ai pas réussi à dépasser la baseline (certes assez haut placée) quelle que soit la méthode; et j'ai dû laisser de côté la partie recommandation de héros pour essayer de dépasser cette baseline. Un résultat négatif reste certes, en sciences, un résultat, mais j'aurais été bien plus satisfait par des résultats positifs ! Le but du projet était de voir ce qu'il était possible d'extraire à partir des données brutes en utilisant des algorithmes d'apprentissage profond. Après 4 mois de travail, la réponse est "pas grand-chose", ce qui est tout de même un peu décevant.

# 7 Apprentissages durant le projet

## 7.1 Apprentissages méthodologiques

Les plus gros enseignements que je tire de ce projet sont d'ordre organisationnel. En effet, après être parti très enthousiaste, je me suis vite rendu compte que développer seul, sans avoir au préalable établi de spécifications précises, était le meilleur moyen pour se retrouver avec des centaines de lignes de code obscures et non réutilisables, et donc inutiles (à ma décharge, c'était mon premier projet personnel de cette envergure, et de loin). J'aurais dû quasiment dès le début davantage structurer mon projet en décidant ce qui allait où, en suivant le principe du *separation of concerns* bien connu en programmation. J'ai effet perdu beaucoup de temps à recoder des fonctionnalités déjà disponibles parce qu'une méthode en amont de la fonctionnalité avait été changée sans faire attention à qui l'utilisait. J'ai également appris à davantage aller au bout d'une idée, et d'en tirer un code propre et stable, pour pouvoir après construire par-dessus. Enfin, en l'absence de dates

limites, la gestion du temps et de l'avancée du projet tout au long des mois d'hiver a également été pour moi un défi, d'autant plus que mon stage en parallèle a été très prenant.

## 7.2 Apprentissages techniques

En ce qui concerne la partie technique, j'ai pu progresser en Python et en Keras, et me familiariser avec gensim et tensorboard. Tensorboard en particulier est un outil que je trouve extrêmement puissant car il m'a permis de remplacer toute une batterie de scripts de visualisation, et de gérer également le stockage des données pour y revenir parfois des mois après. Gensim est également puissant, mais la documentation est incomplète et il m'a donc fallu aller regarder le code source pour savoir quelles étaient les possibilités offertes par le modèle Word2Vec. Enfin, concernant Python et Keras, ce sont des technologies que je manipule déjà au quotidien durant mon stage, mais ce projet étant dans un tout autre registre, cela m'a permis d'utiliser des fonctionnalités que je n'utilise pas d'habitude (appels d'API par exemple).

# 8 Conclusion

Ce projet m'aura beaucoup apporté tant en termes de technique que d'organisation. Le sujet est plutôt inhabituel pour un projet académique, mais les données disponibles sont extrêmement riches et les applications potentielles nombreuses, et je tenais à l'idée d'explorer ce que pouvait apporter l'apprentissage profond à l'analyse de ces données disponibles en grande quantité. Malgré les résultats plutôt décevants pour l'aspect machine learning, on ne saurait conclure pour autant que des techniques statistiques de base éclairées par une bonne connaissance du jeu suffisent à extraire toute l'information pertinente qu'il y a dans les chiffres de plusieurs milliards de matches. Mais peut-être était-il naïf de ma part de penser pouvoir obtenir *ex nihilo* un système de prédiction performant, voire plus performant que les solutions qui existent déjà, le tout en 4 mois.

# 9 Annexes

## 9.1 Code développé

La grosse majorité du code développé dans le cadre de ce projet est disponible sur mon compte Github à l'adresse [.](#) Le code a subi de nombreuses itérations, et est pour l'instant fourni en l'état. Une remise au propre sera faite dans les prochains jours, avec l'ajout d'un guide pour l'installation et le test. En attendant, les seules dépendances du projet sont les librairies python keras, tensorflow, gensim et dota2api, les autres dépendances étant installées par défaut par ces 4 librairies.

## 9.2 Données renvoyées par l'API

### 9.2.1 Informations sur les matches

Ci-dessous les données disponibles via l'API pour un match donné. Quelques clés de compréhension :

- **radiant** et **dire** sont les noms des deux équipes qui s'affrontent
- **barracks** et **towers** sont les bâtiments que l'on trouve dans la base de chaque équipe
- **lobby\_type** permet de filtrer les parties publiques des parties privées (lancées manuellement par un joueur pour jouer contre ses amis par exemple).
- **radiant\_captain** et **dire\_captain** est utilisé dans certains modes de jeu où un joueur choisit tous les héros de son équipe.
- **[pick\_bans]** est la liste de tous les héros choisis ou bannis. En effet, il est possible dans certains modes de jeu de 'bannir' plusieurs héros. Ces héros ne pourront pas être choisis ni par une équipe ni par l'autre. Cela fournit un outil supplémentaire lors du choix des héros. Je ne m'en suis pas servi dans mon projet car cela aurait grandement limité le dataset et bien que cela apporte de l'information utile, elle est difficile à traiter.

```

{
  season           - Season the game was played in
  radiant_win     - Win status of game (True for Radiant win, False for Dire win)
  duration         - Elapsed match time in seconds
  start_time       - Unix timestamp for beginning of match
  match_id         - Unique match ID
  match_seq_num    - Number indicating position in which this match was recorded
  tower_status_radiant - Status of Radiant towers
  tower_status_dire - Status of Dire towers
  barracks_status_radiant - Status of Radiant barracks
  barracks_status_dire - Status of Dire barracks
  cluster          - The server cluster the match was played on, used
  in retrieving replays
  cluster_name     - The region the match was played on
  first_blood_time - Time elapsed in seconds since first blood of the match
  lobby_type       - See lobby_type table
  lobby_name        - See lobby_type table
  human_players    - Number of human players in the match
  leagueid         - Unique league ID
  positive_votes   - Number of positive/thumbs up votes
  negative_votes   - Number of negative/thumbs down votes
  game_mode        - See game_mode table
  game_mode_name   - See game_mode table
  radiant_captain  - Account ID for Radiant captain
  dire_captain     - Account ID for Dire captain
  [pick_bans]
  {
    {
      hero_id        - Unique hero ID
      is_pick        - True if hero was picked, False if hero was banned
      order          - Order of pick or ban in overall pick/ban sequence
      team           - See team_id table.
    }
  }
  [players]
  {
    account_id     - Unique account ID
    player_slot    - Player's position within the team
    hero_id         - Unique hero ID
    hero_name       - Hero's name
    item_#          - Item ID for item in slot # (0-5)
    item_#_name     - Item name for item in slot # (0-5)
    kills           - Number of kills by player
    deaths          - Number of player deaths
    assists          - Number of player assists
    leaver_status   - Connection/leaving status of player
    gold            - Gold held by player
    last_hits       - Number of last hits by player (creep score)
    denies          - Number of denies
    gold_per_min    - Average gold per minute
    xp_per_min      - Average XP per minute
    gold_spent      - Total amount of gold spent
    hero_damage     - Amount of hero damage dealt by player
    tower_damage    - Amount of tower damage dealt by player
    hero_healing    - Amount of healing done by player
    level           - Level of player's hero
    [ability_upgrades] - Order of abilities chosen by player
  }
}

```

```

{
    ability      - Ability chosen
    time         - Time in seconds since match start when ability was upgraded
    level        - Level of player at time of upgrading
}

[additional_units] - Only available if the player has a additional unit
{
    unitname      - Name of unit
    item_#        - ID of item in slot # (0-5)
}
}

// These fields are only available for matches with teams //
[radiant_team]
{
    team_name      - Team name for Radiant
    team_logo       - Team logo for Radiant
    team_complete   - ?
}

[dire_team]
{
    team_name      - Team name for Dire
    team_logo       - Team logo for Dire
    team_team_complete - ?
}
}
}

```

### 9.2.2 Informations sur les joueurs

```

{
    [player]
    {
        avatar          - 32x32 avatar image
        avatarfull       - 184x184 avatar image
        avatarmedium     - 64x64 avatar image
        communityvisibilitystate - See table below.
        lastlogoff        - Unix timestamp since last time logged out of steam
        personaname       - Equivalent of Steam username
        personastate      - See table below.
        personastateflags - ?
        primaryclanid     - 64-bit unique clan identifier
        profilestate      - ?
        profileurl        - Steam profile URL
        realname          - User's given name
        steamid           - Unique Steam ID
        timecreated       - Unix timestamp of profile creation time
    }
}

```