Apprentissage Multi-Agent

Marc-Olivier LaBarre
Département d'informatique et
de recherche opérationnelle
Université de Montréal
Montréal, QC
labarrem @ iro.umontreal.ca

Abstract

L'apprentissage multi-agents est un domaine relativement restreint, mais présentement très dynamique. Le présent document a pour but de faire le point sur cette méthode d'apprentissage, de montrer dans quel domaine on agit, ses bases théoriques et ses buts. On montrera une définition sommaire de plusieurs algorithmes déjà existants.

1. Introduction

Il y a eu récemment dans le monde de l'intelligence artificielle une mode des "agents", au point où on pouvait considérer tout élément d'intelligence artificielle comme un agent. De fait, la mode a passé, mais les agents sont restés. Si le vocable "agent" a relativement reculé, un sujet relié, les multi-agents, est resté comme un centre d'intérêts viable. Ce modèle permettait la coexistence simultanée d'éléments d'agents divers et de leurs liens possibles entres eux et sur l'environnement. Ces objectifs sont particulièrement importants dans les domaines de la théorie des jeux ou encore de la robotique (soit directement physique, soit sur Internet - aussi appelés bots).

Parmi les éléments importants des agents est que leur environnement est variable, et comme plusieurs éléments d'intelligence artificielle, on désire que notre agent soit capable de s'adapter aux changements. Il s'agit ici d'un problème d'apprentissage relativement commun, cependant, la logique même des multi-agents veut non seulement que l'environnement soit variable, mais les autres agents ont également un effet sur les données d'apprentissage d'un agent. On peux séparer les types de relations d'agents en 3 catégories : les agents indépendants, les agents collaboratifs, les agents compétitifs.

La présence d'agents indépendants est la moins importante pour l'apprentissage. Comme leurs buts ne sont pas liés, l'effet d'un agent indépendant sur notre apprentissage se trouve alors similaire à un bruit qu'on pourrait assimiler aux variations de l'environnement.

Les agents collaboratifs posent entre eux certains problèmes pour l'apprentissage. Il arrive qu'on se trouve face à un effet de "classe". Les bénéfices du système peuvent se donner pour tous les agents en même temps, ce qui rend l'apprentissage à un niveau local difficile. Comment l'agent peut-il savoir quelle partie du bénéfice lui appartient - quelle partie il mérite.

Les agents compétitifs sont un problème plus sérieux. Les agents sont en compétition et ont des buts opposés, où la réussite peut très bien passer par la "défaite" des autres agents. C'est une catégorie déjà connue par les méthodes de la théorie des jeux - et c'est de cette catégorie que se basent la plupart des algorithmes d'apprentissage multi-agents. Les jeux compétitifs acceptent plus de deux joueurs, mais l'apprentissage devient alors plus complexe. Avec seulement deux joueurs, l'adversaire obtient généralement un bénéfice inverse au joueur, ce qui fait qu'on sait naturellement que ce qui est bien pour soit est mauvais pour l'adversaire et vice-versa.

Cet article veut faire un résumé de ce que signifie l'apprentissage dans le domaine des multi-agents. Nous présenterons d'abord les bases théoriques des modèles que l'on tente d'apprendre, ensuite certaines qualités recherchées et les buts (généraux et particuliers) des algorithmes d'apprentissage. La section 4 portera sur une courte définition d'algorithmes d'apprentissage multi-agents déjà connus et finalement la section 5 fera des liens avec l'apprentissage machine plus commun.

2. Bases théoriques

Les algorithmes d'apprentissage multi-agents se basent sur deux modèles déjà connus, les modèles de décisions de Markov et les jeux matriciels. De ces deux modèles, on construit un autre modèle, celui des jeux stochastiques, qui sont une réunion des deux idées précédentes. Une stratégie pour un joueur est une manière de décider quelle action; une stratégie pure implique un seul choix déterministe; une stratégie mixte implique un choix probabiliste entre les actions.

2.1 Jeux matriciels

Les jeux matriciels, comme leur nom l'indique, se représentent comme des matrices. On peut les définir comme un tuple (n, $A_{1..n}$, $R_{1..n}$) où n est le nombre de joueurs, A_i est l'ensemble des actions que le joueur i peux poser et R_i la matrice n-dimensions A_1*...*A_n qui donne les bénéfices possibles pour chacune des combinaison possible des actions des joueurs.

Le but d'un joueur dans un tel jeu est de maximiser son bénéfice, mais le calcul de la meilleure stratégie n'est pas chose facile, car on a besoin pour cela de connaître les matrices de bénéfices de tous les joueurs, ce qui n'est pas nécessairement connu au niveau d'un agent lui-même. En général, il n'existe aucune stratégie optimale qui peut parer à tous les cas de jeu des adversaires. Cependant, il existe un ensemble des meilleures réponses (***best-response***) qui inclut toutes les stratégies qui sont optimales si l'adversaire joue une stratégie σ_i (on encore, si les adversaires jouent selon une collection de stratégies σ_i: chaque adversaire joue une stratégie différente).

De là, on peut introduire le concept d'équilibre de Nash (***Nash Equilibrium***), qui représente une collection de stratégies σ_i comme quoi pour chaque joueur, sa stratégie σ_i se trouve être une meilleur réponse aux stratégies σ_j des autres joueurs. Un équilibre de Nash montre donc une situation stable, on un mouvement léger ne peux bénéficier à personne - il s'agit donc d'un point stable à atteindre pour un algorithme d'apprentissage puisqu'il possède la qualité d'être une meilleure réponse et est souvent stable (l'algorithme d'apprentissage convergerait vers un équilibre de Nash). La grande qualité des équilibres de Nash est que chaque jeu matriciel possède un équilibre de Nash, et peux même en posséder plusieurs.

2.2 Problèmes de décision de Markov

Un problème de décision de Markov (ou MDP pour *Markov Decision Problem*) est un modèle où un agent (un seul) peut poser diverses actions, mais que les bénéfices qu'il retire dépendent de l'état dans lequel il se trouve - et ses actions ont une influence sur l'état dans lequel se trouvera le joueur. On peut formaliser un MDP par un tuple (*S, A, T, R*) ou *S* est l'ensemble des états, *A* l'ensemble des actions, *T* la matrice *S*A*S* des probabilités de transition et *R* la matrice *S*A* des bénéfices. Ces MDP sont très importants dans ce qu'on appelle l'apprentissage par renforcement.

2.3 Jeux stochastiques

Les jeux stochastiques (aussi appelés jeux stratégiques ou encore jeux de Markov) sont l'unification des deux concepts précédents. Ce sont des jeux matriciels avec plusieurs états, ou encore des MDP avec plusieurs joueurs. De fait, il est simple de voir qu'un jeu stochastique avec un seul état (ou des états tous identiques), on se retrouve avec un jeu matriciel, et que d'un autre côté, si on n'a qu'un seul joueur avec une stratégie variable (les autres joueurs ont des stratégies fixes), on se retrouve avec un MDP.

Cette extension de deux concepts déjà connus à travers les recherches en théorie des jeux et en apprentissage par renforcement est d'autant plus pratique qu'elle amène avec elle plusieurs des éléments des deux méthodes dont elle vient, en particulier, l'existence d'équilibres de Nash. Les jeux stochastiques sont le framework qui est grandement utilisé pour les algorithmes d'apprentissage des systèmes multi-agents.

Il est à noter que la somme totale des bénéfice donnés par le jeu est un élément important dans l'analyse des algorithmes d'apprentissage, bien qu'il n'en sera pas discuté explicitement ici. Les jeux dont la somme est à zéro (*zero-sum game*) sont en général plus faciles à analyser que les autres (*general-sum game*).

2.4 Exemples

Pair-Impair: deux joueurs, un état, deux actions possibles pour chaque joueur : pair ou impair (ce jeu est également appelé *matching pennies* dans la littérature)

$$R_1 = \begin{matrix} \\ pair \\ impair \end{matrix} \begin{matrix} \begin{matrix} pair & impair \end{matrix} \\ \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \end{matrix} \qquad R_2 = \begin{matrix} \\ pair \\ impair \end{matrix} \begin{matrix} \begin{matrix} pair & impair \end{matrix} \\ \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \end{matrix}$$

Dilemme du prisonnier: deux joueurs, un état, deux actions possibles pour chaque joueur : le silence ou passer aux aveux

$$R_1 = \begin{matrix} \\ silence \\ avouer \end{matrix} \begin{matrix} \begin{matrix} silence & avouer \end{matrix} \\ \begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix} \end{matrix} \qquad R_1 = \begin{matrix} \\ silence \\ avouer \end{matrix} \begin{matrix} \begin{matrix} silence & avouer \end{matrix} \\ \begin{pmatrix} 3 & 4 \\ 0 & 1 \end{pmatrix} \end{matrix}$$

roche-papier-ciseaux: deux joueurs, un état, trois actions possibles pour chaque joueur : roche, papier ou ciseaux

$$R_1 = \begin{array}{c} \\ roche \\ papier \\ ciseaux \end{array} \begin{array}{c} \begin{array}{ccc} r & p & c \end{array} \\ \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \end{array} \quad R_2 = \begin{array}{c} \\ roche \\ papier \\ ciseaux \end{array} \begin{array}{c} \begin{array}{ccc} r & p & c \end{array} \\ \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} \end{array}$$

Ces exemples sont seulement des jeux matriciels simples (2 joueurs, peu d'actions), et bien qu'il semblent très restreint dans leur intérêt, ce sont des problèmes communs et très utilisée (encore) pour vérifier la validité des algorithmes d'apprentissage multi-agents. Si un algorithme est bon, on s'attend à ce qu'il joue bien sur ces jeux simples. Nudelman, Wortman, Leyton-Brown et Shohav ont créé un framework de test des algorithmes d'apprentissage multi-agents appelé GAMUT [6]. Ce framework sert à comparer pour plusieurs jeux connus et relativement communs.

Des jeux relativement plus complexes peuvent inclure (des fois avec certaines modifications au concept des jeux stochastiques) le tic-tac-toe ou encore les échecs, mais ici, on est à un niveau très complexe, et l'apprentissage n'est pas facile. D'autre jeux incluent les applications avec des robots, comme des robots qui jouent au soccer. Bowling a monté un petit jeu de robots pour montrer son algorithme WoLF, où un robot cherche à entrer dans un cercle tracé sur le sol et que l'autre robot cherche à l'y en empêcher.

3. Buts et critères de qualité

Le but principal de l'apprentissage dans un système multi-agent est localement de trouver pour un agent (celui qui apprend) une stratégie qui peut offrir un minimum de garantie de bénéfice peu importe son adversaire. Si notre agent est le seul dont les stratégies sont variables, alors le problème est très simplifié et devient un MDP. Cependant, il est très rare qu'on puisse faire l'assomption que les autres agents ne bougent pas. Dans le pire des cas (ceci est relatif à notre agent), les autres agents apprennent eux aussi, ce qui nous pousse à rechercher dans notre apprentissage un équilibre de Nash. Ainsi, lorsque notre agent a appris, il se trouve dans la bonne situation pour bien répondre à ses adversaires peu importe ce qu'ils sont, et si ceux-ci continuent à (bien) apprendre, ils tomberont eux aussi dans l'équilibre de Nash, et donnera une situation d'une certaine stabilité.

Cependant, l'atteinte d'un équilibre de Nash n'est pas directement un but absolu recherché. Ceci vient du fait qu'on n'assume rien sur les adversaires. Mais, en ***self-play***, il s'agit d'un bon indice qu'on apprend bien. Le ***self-play*** consiste à entraîner notre agent apprenant contre son clone, qui apprend parallèlement avec le même algorithme d'apprentissage. Ce faisant, on s'assure que l'algorithme d'apprentissage permet de trouver cette solution "positive" (ici, on ne souhaite pas voir un joueur battre les autres, mais plutôt que notre algorithme soit capable de bien se battre contre lui-même, et que les agents apprenants sur le même algorithmes atteignent un équilibre entre eux). L'analyse d'algorithmes pour l'apprentissage multi-agent nécessite toutefois d'autres critères, afin qu'on puisse donner des bases théoriques pour appuyer les résultats qu'on peut trouver empiriquement

La rationalité est une propriété qui se définit comme suit : si les stratégies des autres joueurs convergent vers des politiques stationnaires, alors l'algorithme d'apprentissage va converger vers une politique qui est une meilleure réponse aux politiques des autres joueurs. Ce critère pour l'apprentissage nous donne qu'un algorithme d'apprentissage rationnel saura trouver la meilleure réponse pour des adversaires dont les stratégies deviennent stationnaires. En d'autres

mots, cette propriété nous assure que notre algorithme pourra trouver et exploiter les faiblesses de ses adversaires lorsque les politiques de ceux-ci ne changent plus.

La convergence est une caractéristique relativement simple. Elle requiert que notre algorithme d'apprentissage converge vers une politique stationnaire. En général, on restreint cette nécessité de convergence à une classe d'algorithmes d'apprentissage. Notre algorithme doit converger lorsqu'il apprend en jouant contre ces algorithmes. Il y a plusieurs type de convergences: bénéfice moyen, distribution empirique des actions, bénéfice espéré et les stratégies. C'est presque toujours cette dernière convergence qui est demandée, puisqu'elle implique toutes les autres.

Ces deux caractéristique importantes, introduites par Bowling et Veloso [1] sont cependant restreintes à des choses vérifiables à la limite, et on ne peut pas juger ces caractéristiques sur un nombre fini d'étapes. Il est possible qu'on ne voit pas l'effet de ces caractéristiques quand on entraîne notre algorithme sur un nombre fini d'itérations.

La sécurité (***safety***), où encore le non-regret (***no-regret***) est une autre qualité que l'on recherche dans un algorithme d'apprentissage. Pour ceci, on demande que la règle d'apprentissage au moins le bénéfice du maximin. En fait, cette technique nous assure que notre algorithme ne fera pas, en moyenne, pire que l'algorithme statique du maximin. On peut également étendre cette caractéristique à tous les algorithmes statiques, mais ceci est difficile à assumer en pratique.

D'autres caractéristiques, de moindre importance, ont été proposées par Powers et Shoham in [7].

4. Algorithmes Connus

Il existe plusieurs algorithmes déjà utilisés dans le domaine de l'apprentissage multi-agents. Ces algorithmes ont différentes propriétés, et viennent de différents domaines. Dans la plupart des cas, on assume connaître les matrices de bénéfice de tous les joueurs, et quelque fois également les matrices de transition. Certains algorithmes assument également une certaine stratégie de la part de l'adversaire.

4.1 Bully/Minimax

La stratégie de Bully [5] est de choisir l'action qui maximise notre bénéfice en prenant pour acquis que l'opposant choisira l'action qui maximise le sien. En pratique, si nous sommes dans un jeu de somme zéro, Bully nous donnera un algorithme de minimax, puisque les valeurs qui maximisent le profit de l'adversaire sont celles qui minimisent le profit de notre joueur.

Ainsi, l'algorithme de minimax maximise les minima que l'adversaire nous laisse. On voit également le terme maximin dans la littérature, et bien que ces deux termes puissent être facilement mélangés, il s'agit de la stratégie inverse : on minimise le maximum de l'adversaire. Dans certains cas, ces deux stratégies reviennent au même.

Bully et minimax sont deux stratégies statiques, donc elles n'apprennent pas. Un autre problème est que ces stratégies fonctionnent mieux s'il y a un ordre dans le jeu, et que les actions des joueurs ne sont pas simultanées mais ordonnées, comme dans plusieurs jeux plus complexes tels les échecs ou les jeux de cartes. Il s'agit cependant de stratégies souvent utilisées comme baseline, et peuvent même se révéler meilleures que certains algorithmes d'apprentissage selon

les situations. Une variation de Bully existe, appelée BullyMixed, où on permet (encourage) une stratégie mixte plutôt qu'une stratégie pure.

4.2 Jeu fictif (*fictious play*)

Par le jeu fictif [3], notre agent assume que la stratégie de l'adversaire est de tirer ses actions d'une distribution fixe, et estime cette distribution en comptabilisant les fréquences de chaque action que l'adversaire a joué dans le passé.

4.3 Q-learning

Q-Learning est un algorithme directement emprunté à l'apprentissage par renforcement. Cet algorithme a été fondé autour de MDP, et ne s'applique pas directement à des jeux stochastiques, mais a été considéré par plusieurs comme une base solide pour des développements futurs.

Q-Learning se base sur des estimations de paires état-action. La valeur Q(s,a) est définie comme étant l'estimation des bénéfices futurs (réduits). Une fois que ces valeurs sont apprises, la stratégie optimale est de choisir l'action qui a la Q-valeur maximale. Voici les équations de mise à jour des Q-valeur et de stratégie V(s)

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[R(s,a) + \gamma V(s')]$$

$$V(s) \leftarrow \max_{a \in A} Q(s,a)$$

où α est le taux d'apprentissage et γ le taux de réduction (*discount*).

De cet algorithme, on peut le généraliser pour plusieurs agents tel que montré en [8]. On a proposé ensuite pour améliorer cette stratégie le minimax-Q, puis ensuite Nash-Q, aussi appelé *Friend-or-Foe* (*FoF*) et finalement CE-Q. la stratégie *FoF* a ceci d'intéressant qu'elle a deux façon de mettre à jour les stratégies V(s), selon qu'elle a classifié le problème comme *Friend* (une action optimale globale existe) ou comme *Foe* (on trouve plutôt un point de selle). Cette classification s'effectue en regardant les Q-valeurs à travers l'exécution de l'algorithme. Q-Learning est rationnel et convergent.

4.4 Filtres de Kalman

L'utilisation de filtres de Kalman [4] est une manière de simplifier un problème à plusieurs états et plusieurs agents proposé par Chang, Ho et Kaebling. L'idée de cet algorithme est, dans un système collaboratif (où plusieurs agents ont des buts communs), d'être capable d'isoler le "bruit" que causent les autres agents en regard à une récompense totale. Cet algorithme utilise une mise à jour de stratégies très similaire à celle du Q-Learning, mais comme on se trouve dans une situation où on ne connaît pas *R* directement, on doit l'estimer, et c'est là que les filtres de Kalman entrent en jeu.

4.5 IGA

IGA vient de *Infinitesimal Gradient Ascent*. De fait, on applique ici une montée de gradient très légère, avec un taux d'apprentissage variable qui tend vers zéro (d'où la notion d'infinitésimal). Cette façon de faire, introduite par Singh, Kearns et Mansour, vient avec une preuve de convergence vers les bénéfices espérés d'un équilibre de Nash. De fait, IGA s'arrêtera éventuellement à un endroit où le gradient est zéro. Le problème de cette montée de gradient est que sa convergence n'est assurée qu'à l'infini, et qu'entre temps, on peut passer par plusieurs situations sous optimales, et éventuellement, en pratique, ne pas obtenir une solution

satisfaisante en un nombre fini d'itérations. Cet algorithme est toutefois important puisqu'il s'agit du premier qui s'accompagne d'une preuve de convergence. IGA a été bâti pour des jeux de deux joueurs et deux actions, mais il a été étendu pour donner GIGA (*Gineralized IGA*) [10].

4.6 PHC

PHC, pour *Policy Hill-Climbing* est une amélioration du Q-Learning. Plutôt que de baser le comportement d'apprentissage sur une stratégie pure (une seule action peut être choisi selon l'état), on utilise les Q-valeurs pour se créer une stratégie mixte (des probabilités pour chaque action). On commence par initialiser toutes les actions uniformément ($1/|A_i|$), puis on change ces probabilités selon que l'action *a* est l'action qui "gagne" l'étape de Q-Learning (celle avec la Q-valeur maximale). La probabilité de l'action *a* augmente légèrement et la probabilité de chacune des autres actions baisse légèrement.

4.7 WoLF

WoLF [1] n'est pas vraiment par lui-même un algorithme d'apprentissage, mais plutôt une philosophie qu'on peut marier avec les deux algorithmes précédents pour créer des algorithmes plus performants. L'idée de WoLF (*Win of Learn Fast*) est d'avoir deux taux d'apprentissage, un lorsqu'on gagne et un lorsqu'on perd. Ceci vient de la stratégie que lorsqu'on perd, on doit apprendre beaucoup, mais lorsqu'on gagne, on ne doit pas beaucoup modifier sa stratégie parce que nos adversaires vont alors apprendre plus, et on ne doit pas leur prêter flanc. Les variations de la technique WoLF de Bowling sont parmi les algorithmes les meilleurs à ce jour, et s'accompagne de garanties de rationalité et de convergence, et même pour le dernier, de non-regret. Il s'agit de WoLF-IGA, WoLF-PHC et GIGA-WoLF [2].

4.8 Méthode hybride

Une méthode hybride a récemment été proposée récemment par Powers et Shoham [7]. Cette nouvelle méthode propose de conjuguer 3 algorithmes faibles (jeu fictif, BullyMixed et Maximin) pour parer à plusieurs types d'opposants. De fait, on test généralement un algorithme d'apprentissage contre lui-même, mais il ne faut pas croire qu'il existe un algorithme absolu meilleur contre tout le monde tout le temps. On assume en général des adversaires qui apprennent, mais ceci n'est pas toujours le cas. On peut voir des adversaires aux politiques stationnaires, et dans ce cas, le jeu fictif est une meilleur stratégie. Sinon, il existe quand même des classes d'adversaires contre lesquels une méthode est meilleure qu'une autre. La méthode hybride (appelée MetaStrategy) commence par utiliser BullyMixed, puis après un certain temps, peu changer pour le jeu fictif, et en un dernier temps, revoit ceci et permet aussi de se tourner vers une stratégie maximin. MetaStrategy possède la qualité de non-regret.

5. Liens avec l'apprentissage classique

À première vue, l'apprentissage multi-agent et l'apprentissage classique n'ont que peu d'éléments en commun, à part le terme apprentissage lui-même. L'apprentissage multi-agents apprend des paramètres très directs (les probabilités de choisir une action à un certain état) alors que l'apprentissage classique apprend des paramètres plus complexes. Aussi, même si l'on veut voir un classifieur comme un agent qui "classifie" dans une action (fait un choix), ou encore, via des softmax, nous produit une distribution de probabilités sur les actions, il nous manque toujours le quoi classifier. De fait, les algorithmes multi-agents ne classifient pas vraiment, puisqu'ils prennent leur décision sans vrai savoir. On pourrait toujours considérer l'état dans

lequel on se trouve comme un élément connu, ou encore prendre le comportement passé de l'adversaire comme données.

Il reste tout de même que si on considère chaque agent comme une boîte noire, on pourrait lui substituer un classifieur d'actions. Le problème ici est qu'on mélange deux philosophies relativement séparées, et que de deux méthodes potentiellement lourdes en calcul et en temps, on risque de se perdre. En plus, on fait disparaître un élément clé de l'apprentissage supervisé standard : des sorties connues. Tout ceci n'est pas complètement distinct, et on pourrait voir dans l'avenir une plus grande proximité entre ces deux modèles d'apprentissage, mais dans l'état actuel des choses, ils sont encore trop éloignés pour 'travailler ensemble'.

Références

[1] Bowling, M., Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[2] Bowling, M. (2004). Converge and no-regret in multiagent learning. In *Proceedings of the 17th Neural Information Processing Systems (NIPS)*

[3] Brown, G. (1951). Iterative Solution of Games by Fictitious Play. In *Activity Analysis of Production and Allocation*. New York: John Wiley and Sons.

[4] Chang, Y., Ho, T., Kaelbling L. (2003). All learning is local: Multi-agent learning in global reward games. In *Proceedings of the 16th Neural Information Processing Systems (NIPS)*

[5] Littman, M. & Stone, P. (2001). Implicit Negotiation in Repeated Games. In *Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages*, pp. 393-404.

[6] Nudelman, E., Wortman, J., Leyton-Brown, K., & Shoham, Y. (2004). Run the GAMUT: A Comprehensive A pproach to Evaluating Game-Theoretic Algorithms. *AAMAS-2004*. To Appear.

[7] Powers, R., Shoham, Y. (2004). New criteria and a new algorithm for learning in multi-agent systems. In *Proceedings of the 17th Neural Information Processing Systems (NIPS)*

[8] Shoham, Y., Powers, R., & Grenager, T. (2003). Multi-Agent Reinforcement Learning: a critical survey. Technical Report.

[9] Singh, S., Kearns, M., & Mansour, Y. (2000). Nash convergence of gradient dynamics in generalsum games. In *Proceedings of UAI-2000*, pp. 541-548, Morgan Kaufman.

[10] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 928–925, 2003.