Apprendre à tomber par la méthode d'apprentissage par renforcement

supervisé par Tien Tuan DAO

Anh Tu NGUYEN, Yunfei ZHAO

27 octobre 2021

Table des matières

1 Introduction

1.1 Apprentissage par renforcement

L'apprentissage par renforcement est une grande catégorie dans la grande famille de l'apprentissage automatique. L'utilisation de l'apprentissage par renforcement peut faire en sorte que les machines apprennent à obtenir des scores élevés dans l'environnement et à obtenir d'excellents résultats. Mais derrière ces réalisations se trouve son travail acharné et continu essais, expérience accumulée.

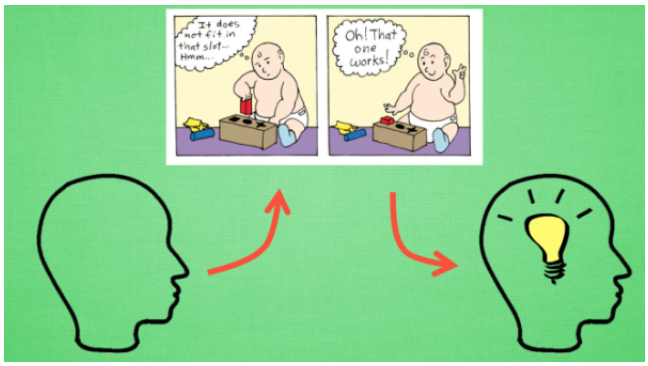


FIGURE 1 – Apprentissage par renforcement

L'apprentissage par renforcement est un type d'algorithme qui permet aux ordinateurs d'apprendre depuis zéro. Il n'y a aucune idée dans leur tête. Grâce à des essais et erreurs continus, ils apprennent enfin les règles et apprennent à atteindre leurs objectifs. Cela est le processus d'apprentissage par renforcement. Il existe de nombreux exemples d'apprentissage par renforcement dans la pratique. Par exemple, le plus célèbre Alpha GO, la machine a battu des maîtres humains en GO pour la première fois, et L'apprentissage par renforcement a permis à l'ordinateur d'apprendre à jouer seul au jeu Atari. En applicant les algorithmes, l'agent a essayé de mettre à jour leur stratégie de conduite afin d'apprendre étape par étape comment manipuler le jeu pour obtenir des scores élevés.

Comment un ordinateur(agent) apprendre ? Il s'avère que l'ordinateur a également besoin d'un professeur virtuel. Ce professeur est relativement bâclé. Il ne vous dira pas comment bouger, comment prendre des décisions. Ce qu'il fait pour vous est seulement de noter votre comportement.

Sous quelle forme devons-nous apprendre ces existants ressources, ou comment apprendre uniquement à partir du score donné par un professeur virtuel ? comment dois-je prendre une décision ? En fait, l'agent n'a besoin que de se souvenir de ces scores élevés, des scores bas et les comportements correspondants. la prochaine fois, l'agent utilise le même comportement qui a obtenu les bon scores pour obtenir des scores élevés et évider les comportements à faible score.

FIGURE 2 – Professeur virtuel

Nous savons que l'apprentissage supervisé a déjà les bonnes étiquettes pour les données. telles que celle-ci, L'apprentissage supervisé peut apprendre quelles étiquettes correspondent à ces données. Cependant, l'apprentissage par renforcement doit aller plus loin. Au début, il n'avait ni données ni d'étiquettes. Il souhaite obtenir ces données et étiquettes par des tentatives répétées dans l'environnement, puis apprendre quelles données peuvent correspondre à quelles étiquettes. Grâce à ces règles apprises, il peut choisir le comportement qui apporte des scores élevés.

L'apprentissage par renforcement est une grande famille, il contient de nombreux algorithmes et nous mentionnerons également certains des algorithmes les plus célèbres, tels que les méthodes de sélection de comportements spécifiques en fonction de la valeur des comportements, y compris **Q l'apprentissage** qui apprendre à l'aide de l'apprentissage par table, **sarsa**, **réseau Q profond** utilisant l'apprentissage des réseaux neuronaux. Et les méthodes qui produisent directement le comportement comme **gradients de politique** , ou les méthodes comprennent l'environnement, imaginent un environnement virtuel et apprennent de l'environnement virtuel, etc. L'apprentissage par renforcement associé à des réseaux de neurones a récemment conduit à un large éventail de succès dans les politiques d'apprentissage pour les problèmes de prise de décision séquentiels et Nous avons utilisé ce type de méthode pour réaliser ce projet

1.2 Explication de problème et Motivation de projet

L'objet de notre projet est d'étudier le processus de tomber. Nous voulons simuler le le processus de tomber d'une personne dans environnement numérique. Plus précisément, notre but est de faire un modèle squelette composé des muscles et des os tombe vers un sens prédéfinit(avant, arrière, gauche, droite). Nous réalisons la mission dans un environnement numérique en utilisant l'apprentissage par renforcement. Ensuite, nous allons observer les information nous pouvons obtenir pendant ce processus, par exemple les mouvements de squelette, les niveaux d'activation des muscles, les angles des articules pendant le mouvement, etc.

Ce projet est une recherche utiliser l'apprentissage par renforcement (RL) pour résoudre des problèmes dans les soins de santé. Il applique RL dans des environnements complexes de calcul, avec une stochasticité et des espaces d'action hautement dimensionnels, pertinents pour des applications réelles. il relie les communautés de la biomécanique, des neurosciences et de l'informatique. Il est une petite recherche déroule comme un TX à l'UTC supervisé par **M.Tien Tuan DAO**

Le projet est une dérivé de concours «NIPS “Learning to Run”» qui est co-organisé par des chercheurs de UC Berkeley, EPFL et du Mobilize Center. Nous utilisons la même environment **osim-rl**(OpenSim RL).

Le package **osim-rl** [6] nous permet de synthétiser des mouvements physiologiquement précis en combinant une expertise biomécanique intégrée dans le logiciel de simulation OpenSim avec des stratégies de contrôle de pointe utilisant l'apprentissage en renforcement profond.

L'environnement que nous avons utilisé est **L2M2019Env**.

2 Environnement

L'environnement **L2M2019Env** de Opensim est un squelette qui possède 22 muscles sur les jambes et 4 dégrées de liberté sur chaque jambe et nous pouvons passer les variables d'excitation dans intervalle [0,1] sur les muscles. La fréquence de simulation est 100HZ et l'observation contient totalement 41 values qui construit notre S, l'environnement. Les actions A seront les signes de simulation qu'on va passer aux muscles.

Nous utilisons certaines des attributs pour construire notre fonction de Reward. Comme cela, notre agent peut faire l'interaction avec environnement. Et nous allons détailler notre algorithme dans les chapitre suivants.

— Sur chaque jambes, nous possédons 11 muscles : abd
add
hamstrings
biceps femoris
gluteus maximus
iliopsoas
rectus femoris
vastus
gastrocnemius
soleus
tibialis anterior

— L'observation a 41 values :
position of the pelvis (rotation, x, y)
velocity of the pelvis (rotation, x, y)
rotation of each ankle, knee and hip (6 values)
angular velocity of each ankle, knee and hip (6 values)
position of the center of mass (2 values)
velocity of the center of mass (2 values)
positions (x, y) of head, pelvis, torso, left and right toes, left and right talus (14 values)
etc

3 Algorithme

3.1 Deep Deterministic Policy Gradient

En apprentissage par renforcement, il y 2 environnements : l'environnement discrète et l'environnement continue. L'environnement discrète est un environnement où leur actions sont discrètes et finies alors que les actions de l'environnement continue sont continues et infinies. L'approche pour chaque type d'environnement donc est différente. Pour l'environnement discrète, les méthodes stochastiques sont bien adaptées : la sortie du modèle est probabilité de leur actions. La méthode $\epsilon - greedy$ est appliqué pour choisi une action selon les probabilités produites par le modèle et pour s'assurer l'exploitation du modèle. Dans un environnement continue, comme l'ensemble des actions est continue, $\epsilon - greedy$ est donc très difficile à implémenter. Par conséquent, une autre approche est proposée : Deterministic Policy Gradient. La sortie du modèle est une action et pour permettre au modèle à explorer des nouvelles connaissances, des bruits sont donc ajoutés.

Notre environnement est continue donc on appliquera Deep Deterministic Policy Gradient. Pour s'assurer la convergence au point optimal et pour s'assurer l'exploration du modèle, nous appliquerons **OUnoise** ou **Gaussiannoise** avec la stratégie **"experience replay"** qui seront discuté en parties prochaines.

3.2 Algorithme : Deterministic Actor-Critic

Actor-Critic est une architecture très populaire basée sur le théorème de Policy Gradient. Elle est composée de 2 composants. Actor modifie le paramètre θ de policy $\pi_\theta(a|s)$ par l'algorithme de gradient ascent. Comme on ne sais pas la fonction exacte d'action-value $Q^\pi(s,a)$, on utilisera la fonction $Q^\omega(s,a)$ avec le paramètre ω. C'est Critic qui estimera cette fonction pour obtenir $Q^\omega(s,a) \approx Q^\pi(s,a)$.

On utilisera Deterministic Actor-Critic pour cet environnement qui est continue. L'idée fondamentale de cet algorithme est ce que nous avons présenté dans la paragraphe ci-dessus. Nous avons 2 approches : On-policy actor-critic et Off-policy actor-critic et elles ont quelques différences en stratégie d'apprentissage pour obtenir le point optimal.

Nous discuterons d'abord l'apprentissage par l'algorithme on-policy. Avec l'apprentissage par on-policy, nous utiliserons Actor pour modifier le paramètre θ de policy $\pi_\theta(a|s)$ et nous utiliserons également ce policy $\pi_\theta(a|s)$ pour créer des trajectoires (ensembles des états, actions, rewards,... qui servent à l'apprentissage). C'est-à-dire nous effectuerons des changements directement sur le policy qui crée des samples pour l'apprentissage. Critic utilise S̈arsa updateẗpour estimer $Q^\omega(s,a)$. L'apprentissage par on-policy est représentée dans la figure ci-dessous. L'apprentissage par on-policy ne s'assure pas la convergence au point optimal global. Par ailleurs, la convergence de cette méthode est assez lente.

$$\begin{aligned}
\delta_t &= r_t + \gamma Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t) \\
w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\
\theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t)\ \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}
\end{aligned}$$

FIGURE 3 – Apprentissage par on-policy

Pour que l'apprentissage soit plus stable, l'algorithme off-policy est appliqué. En apprentissage par off-policy, nous utiliserons une autre policy $\beta(a|s)$ pour créer le trajectoire et nous modifierons le paramètre θ du policy $\pi_\theta(a|s)$. Par ailleurs, Critic utilise Q̈-learningp̈our estimer $Q^\omega(s,a)$ à $Q^\pi(s,a)$. Ce changement permet au modèle converge plus rapidement que l'apprentissage on policy. L'apprentissage par on-policy est présentée dans la figure suivante.

$$\begin{aligned}\delta_t &= r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t)\\ w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)\\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t)\ \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}\end{aligned}$$

FIGURE 4 – Apprentissage par off-policy

Comme notre environnement est très complexe et donc l'apprentissage sera évidemment assez longue, nous décidons d'appliquer l'algorithme off-policy pour notre projet. L'algorithme complet de Actor-Critic par off-policy est :

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

FIGURE 5 – Algorithme de Actor-Critic par off-policy

3.3 Approches potentielles

Pendant l'étude du projet, nous avons consulté les solutions de la compétition **NIPS2017** : **Learningtorun**. Le but de cette compétition était d'apprendre un modèle à courir qui est différent que notre but. Par contre, l'environnement de la compétition **NIPS** : **2017** est assez identique que notre environnement donc les solution de cette compétition est un bon point de départ. Nous avons donc choisi 3 algorithmes : Actor-Critic classique, Actor-Critic Ensemble et Actor-Critic Asynchrone avec quelques réseaux neuronaux plus complexes comme ResNet ou Inception pour obtenir un modèle plus flexible.

La première approche que nous considérons est Actor-Critic classique avec des améliorations. C'est la méthode la plus simple à implémenter ainsi que la méthode assez efficace. Les améliorations que nous réaliserons seront la fonction d'activation (RELU, Sigmoid, SELU, tanh,...), les hyper-paramètres, la régularisation des layers en réseaux neuronaux, les bruits et notamment reward shaping. Dans la compétition en 2017, cet algorithme a obtenu un résultat assez remarquable.

La deuxième approche est Actor-Critic Ensemble. Elle est une approche qui avait un résultat très intéressant dans la compétition **NIPS** : **2017**. L'auteur a utilisé cet algorithme pour identifier les actions qui produit les états instables à chaque itération en utilisant un ensemble de Critic de choisir la meilleure action à partir des actions proposés par un ensemble de Actor. A partir de cette idée, nous pourrons utiliser Actor-Critic Ensemble pour identifier une action qui permetra au modèle de tomber dans la direction que nous choisirons. L'architecture de Actor-Critic Ensemble est représentée par la figure suivante.

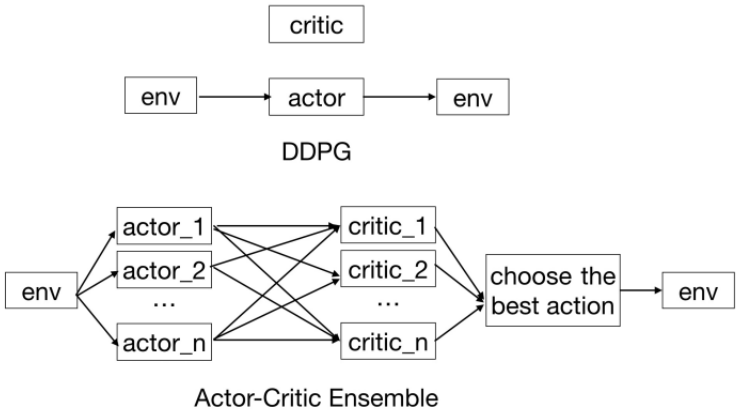


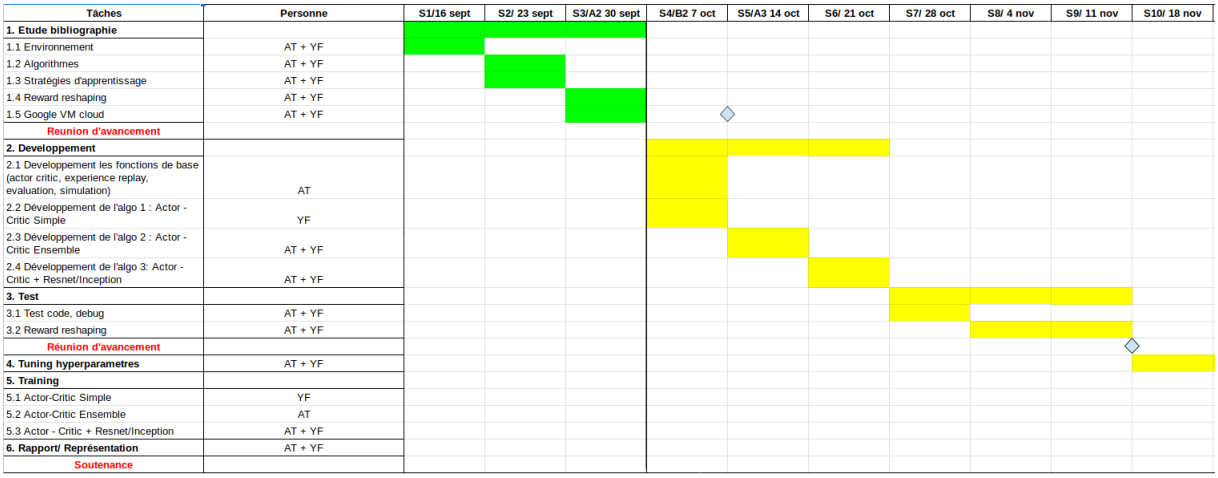
FIGURE 6 – Algorithme de Actor-Critic Ensemble

En fin, nous avons choisi la méthode Actor-Critic Asynchrone avec quelques réseaux neuronaux plus complexes. Les expériences (états, rewards, ...) seront stockées dans un buffer jusqu'à la fin de chaque épisode. Nous déciderons puis quels expériences seront choisies pour stocker dans la mémoire buffer pour l'apprentissage par "*experiencereplay*" grâce à leurs rewards. Cette approche nous permettra de diminuer la durée d'apprentissage notamment avec un tel dimensionnement grand environnement. Par ailleurs, dans la solution de la compétition **NIPS** : **2017**, les auteurs ont proposé quelques techniques pour augmenter l'efficacité de l'apprentissage.

3.4 Stratégie d'apprentissage : Experience Replay

Nous savons que l'apprentissage renforcement ne possède pas le dataset pré-généré depuis lequel qu'on puisse apprendre. Dans notre cas, nous pouvons pas savoir quels sont les combinatoires des muscles activés pendant une épisode peuvent causer le squelette tombe vers le sens qu'on veut. Donc l'agent doit enregistrer tous les transitions des états pour qu'on les utilise dans les apprentissages suivants. le memory-buffer utilisé pour enregistrer ces transition est Experience Reply. Les deux architectures des memory-buffer plus utilisés sont *cyclic memory buffers* et *resevoir-sampling-based memory buffer*. D'ici, nous allons rencontrer un problème éternel d'apprentissage renforcement : le trade-off d'**Exploitation** et **Exploration**. Agent doit essayer d'assez explorer l'environnement et en même temps il doit exploiter ce qu'il a apprise de l'expérience.

4 Planning



Tâches	Personne	S1/16 sept	S2/ 23 sept	S3/A2 30 sept	S4/B2 7 oct	S5/A3 14 oct	S6/ 21 oct	S7/ 28 oct	S8/ 4 nov	S9/ 11 nov	S10/ 18 nov
1. Etude bibliographie											
1.1 Environnement	AT + YF										
1.2 Algorithmes	AT + YF										
1.3 Stratégies d'apprentissage	AT + YF										
1.4 Reward reshaping	AT + YF										
1.5 Google VM cloud	AT + YF										
Reunion d'avancement											
2. Developpement											
2.1 Developpement les fonctions de base (actor critic, experience replay, evaluation, simulation)	AT										
2.2 Développement de l'algo 1 : Actor - Critic Simple	YF										
2.3 Développement de l'algo 2 : Actor - Critic Ensemble	AT + YF										
2.4 Développement de l'algo 3: Actor - Critic + Resnet/Inception	AT + YF										
3. Test											
3.1 Test code, debug	AT + YF										
3.2 Reward reshaping	AT + YF										
Réunion d'avancement											
4. Tuning hyperparametres	AT + YF										
5. Training											
5.1 Actor-Critic Simple	YF										
5.2 Actor-Critic Ensemble	AT										
5.3 Actor - Critic + Resnet/Inception	AT + YF										
6. Rapport/ Représentation	AT + YF										
Soutenance											

FIGURE 7 – GANTT du projet

Nous avons porté une attention particulière sur la réalisation du GANTT afin d'y intégrer un maximum d'informations utiles pour nous permettre de nous projeter assez loin dans la réalisation du projet et pouvoir livrer nos livrables en temps et en heure.

Ce projet durera pendant 17 semaines à partir de la première réunion jusqu'à la soutenance du projet. Le projet est divisé en 6 phases principales : l'étude de bibliographies, l'implémentation des algorithmes, le test de codes, l'optimisation des algorithmes, l'apprentissage et la documentation. Il y a également 2 réunions d'avancement et une soutenance pour revoir les résultats pendant la réalisation du projet.

Nous avons 3 semaines pour l'implémenter les algorithmes que nous avons choisis, 3 semaines de revoir les algorithmes et le "reward reshaping" avant 6 semaines de l'optimisation et l'apprentissage. Nous espérons d'obtenir 3 modèles qui pourront répondre la demande du porteur du projet.

5 Déroulement du projet

5.1 Implémentation

Après avoir défini les composants nécessaires pour ce projet, nous avons commencé à les implémenter. Pendant ce projet, nous avons implémenté les réseaux neuronaux de Actor et Critic, l'algorithme Actor-Critic, la stratégie Experience Replay et Prioritized Experience Replay, la détecteur de direction, OU-Noise et la suivie d'apprentissage par Tensorboard. Ces composants ont été développé en Python 3.6.1 et en Tensorflow 1.14.

5.1.1 Actor-Critic

Tout d'abord, nous avons implémenté les réseaux de Actor et Critic. Nous avons eu 2 approches pour les réseaux neuronaux :

- **Approche 1 :** Actor prend l'état du corps (les muscles, les articulations, les forces, etc ...) en entrée lors que Critic prend une action et un état du corps en entrée.
- **Approche 2 :** Actor prend l'angle entre le corps et la direction prédéfinie (Nous représenterons dans la partie prochaine) en entrée lors que nous entrons une action et un angle entre le corps et la direction prédéfinie au réseau.

Pour toutes les 2 approches, Actor propose une action à l'environnement lors que Critic sort une Q-value de l'action et l'état en entrée. Par ailleurs, nous avons des couches cachées différentes pour les réseaux de Actor et ceux de Critic. Pour Actor, nous avons utilisé 3 couches cachées : la première couche cachée a 400 unités cachés, la deuxième couche a 300 unités et le nombre des unités dans la dernière couche est égal au nombre d'éléments dans l'espace d'actions. Nous avons ajouté les couches de type Batch Normalization pour normaliser des données. Cela nous permet d'éviter over-fitting pendant l'apprentissage. Nous avons eu RELu comme la fonction d'activation dans les 2 premières couches et tanh comme la fonction d'activation de la dernière couche. La valeur de la fonction tanh est comprise dans [-1,1]. Comme les actions danc cet environnement sont comprises dans [0,1] dans nous avons converti les valeurs en [-1,1] aux valeurs en [0,1] par la formulaire : $y = \frac{1}{2}x + \frac{1}{2}$ avec $x \in [-1, -1]$ et $y \in [0, 1]$. Il y a eu 2 entrées dans Critic : les actions et les états. Les états entrées dans réseau ont été passés une couche cachée de 400 unités. Les états ensuite ont été passés une couche cachée de 300 unités. Les actions entrées ont été aussi passées une couche cachée de 300 unités. Les sorties de 2 couches cachées de 300 unités ont été immergé et puis passées une dernière couche qui a eu un seul unité. C'est notre Q-value. Toutes les couches ont eu RELu comme la fonction d'activation.

Nous avons ensuite implémenté l'algorithme Actor-Critic comme pseudo-code présenté dans la partie précédente. Au début, nous avons utilisé Keras pour implémenter les réseaux neuronaux et l'algorithme mais la vitesse d'apprentissage était très lente (environ 20s par épisode) et il était difficile à contrôler les calculs pendant d'apprentissage. Par conséquent nous avons re-implémenté en Tensorflow 1.14. La vitesse d'apprentissage a été amélioré significativement (environ 12s par épisode). Par ailleurs, nous avons développé la suivie d'apprentissage par des graphes par Tensorboard. Nous pourrons savoir la tendance de l'apprentissage.

Nous avons implémenté également la visualisation qui prendra le modèle déjà entraîné et visualisera le mouvement du modèle ainsi que les graphes importants.

5.1.2 Détection de direction

La détection de direction du modèle est la partie la plus importante dans ce projet car c'est un critère fondamental pour évaluer la performance des actions et puis pour calculer le reward. Comme il n'y a pas de fonctions prédéfinies existantes dans l'environnement qui nous permet de déterminer ces informations, nous avons cherché à implémenter une fonction qui retournerait le vecteur de direction du corps à chaque time-step. Pendant la recherche, nous avons trouvé des éléments importants dans l'espace d'observations de l'environnement : la position de la masse centrale, la position du talus gauche et la position du talus droit. Nous avons calculé le point milieu entre 2 talus et puis le vecteur à partir de ce point à la masse centrale. Nous l'appelons le vecteur de direction.

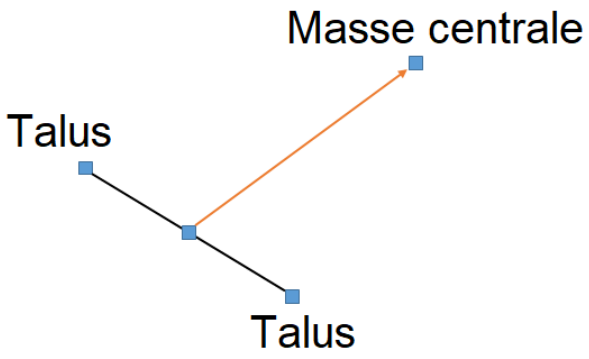


FIGURE 8 – Direction du modèle

Cette méthode n'était pas très précise dans tous les cas. Cependant, en général ce calcul peut retourner des résultats acceptables.

5.1.3 Experience Replay - Prioritized Experience Replay

Comme ce que nous avons discuté dans la partie précédente, Experience Replay est une stratégie qui nous permettra de converger plus rapidement au point optimal en choisissant des expériences non corrélées dans la mémoire. A chaque time-step, nous avons enregistré l'état, reward, l'action, l'état suivant et l'indicateur pour savoir si c'est un état terminal. Nous avons ensuite choisir aléatoirement un ensemble d'expériences pour que le modèle puisse apprendre. Le nombre d'expériences choisies à la fois est à notre choix. C'est la stratégie Experience Replay simple. Nous avons implémenté et testé cette stratégie sur notre environnement et nous avons obtenue des résultats pas très bons et quelquefois le modèle n'a pas convergé. Car cet environnement est très compliqué, il était difficile à faire un bon action. Par conséquent, dans la mémoire, le nombre de bonnes actions est très faible relativement au nombre de mauvaises actions. Nous avons trouvé une solution pour ce problème : Prioritized Experience Replay.

Dans Prioritized Experience Replay (PER), nous avons choisi les transitions selon leurs Temporal Difference ($y_i = r_i + \gamma Q' \left(s_{i+1}, \mu'\left(s_{i+1}|\theta^{\mu}\right)|\theta^{Q'}\right)$ dans ce cas) : Les transitions qui ont une faible erreur de Temporal Difference lors de leur premier passage ont une faible probabilité d'être échantillonnées à nouveau

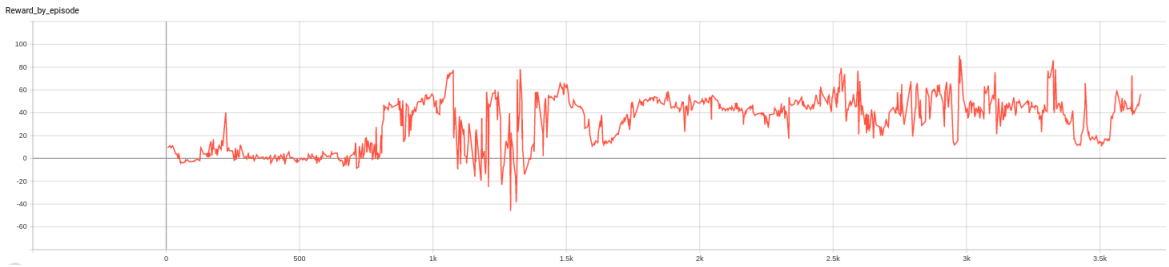


FIGURE 9 – Apprentissage par Experience Replay

et l'inverse, les transitions qui ont une grand erreur de Temporal Difference ont une grande probabilité d'être échantillonnées. Les transitions sont triées par le rang. Le rang est l'emplacement dans la mémoire trié de haut en bas par erreur TD et alpha détermine la quantité de priorité utilisée (alpha = 0 est identique à loi uniforme).

Nous avons testé PER avec notre environnement et nous avons obtenu les résultats remarquables par rapport à Experience Replay simple. Le modèle a convergé plus rapidement et beaucoup plus stable.

5.1.4 OU-Noise

Pour assurer l'exploitation de notre modèle, nous avons implémenté les fonctions qui permettraient d'ajouter des bruits pendant l'apprentissage. Il y a 2 façon d'ajouter des bruits : Ajouter directement aux actions proposées par Actor et Ajouter aux paramètres. L'ajout direct aux paramètres sera plus stable mais il est difficile à implémenter. Par conséquent nous avons implémenté des bruits d'actions. Nous avons implémenté 2 types de bruits : les bruit introduits par le processus d'Ornstein–Uhlenbeck et les bruits Gaussiens. En faisant les tests avec des environnements différents, nous avons observé que les bruits d'Ornstein-Uhlenbeck était plus performant que les bruits Gaussiens.

Nous avons implémenté également le décalage de bruits qui diminuerait les bruits ajoutés au modèle pendant l'apprentissage de temps en temps. Cela permettra le modèle d'exploiter beaucoup au début de l'apprentissage pour comprendre l'environnement.

5.2 Test

Nous avons testé tous les composants que nous avons implémenté avec l'environnement Pendulum-v0 de OpenAI Gym. L'algorithme Actor-Critic avec PER a convergé très rapidement après environ 30 épisodes. C'était raisonnable car cet environnement a une petite espace d'état.

Nous avons puis testé les composants implémentés avec notre environnement sur Google Cloud Platform. Nous avons utilisé une machine de 8 CPUs et un RAM de 30Go pour pourvoir stocker tous les mémoire de transitions. Avec la taille de batch de 64 transitions, une épisode a pris environ 12 à 15 secondes. Tous les composants ont bien marché et étaient prêt pour l'apprentissage.

5.3 Reward shaping

Notre environnement n'a été créé que pour apprendre à marcher et la fonction de reward par défault de l'environnement était basée sur la distance et l'effort à marcher. Par conséquent, nous avons cherché à créer des nouvelles fonctions de reward pour obtenir notre but : Apprendre à tomber dans un sens prédéfini. Dans la partie précédente, nous avons introduit la manière de déterminer le vecteur de direction actuelle du corps. Dans cette partie nous introduirons la façon de définir la direction à tomber pour le modèle et puis des idées de reward shaping basées sur la définition de la direction et le vecteur de direction du corps.

5.3.1 Définition de la direction à tomber

Nous avons observé dans la figure 10 que le corps est posé initialement à l'origine du repère Oxyz selon l'axe Oy. La tête est vers le sens positif de l'axe Ox, Oz est vers le côte droit du corps. Par conséquent, le côté droit du corps est les vecteurs parallèles avec l'axe Oz ayant la même direction avec le vecteur $\vec{Oz}$, le côté gauche du corps est les vecteurs parallèles avec l'axe Oz ayant la même direction avec le vecteur $-\vec{Oz}$. Le devant du corps est les vecteurs parallèles avec le vecteur $\vec{Ox}$.

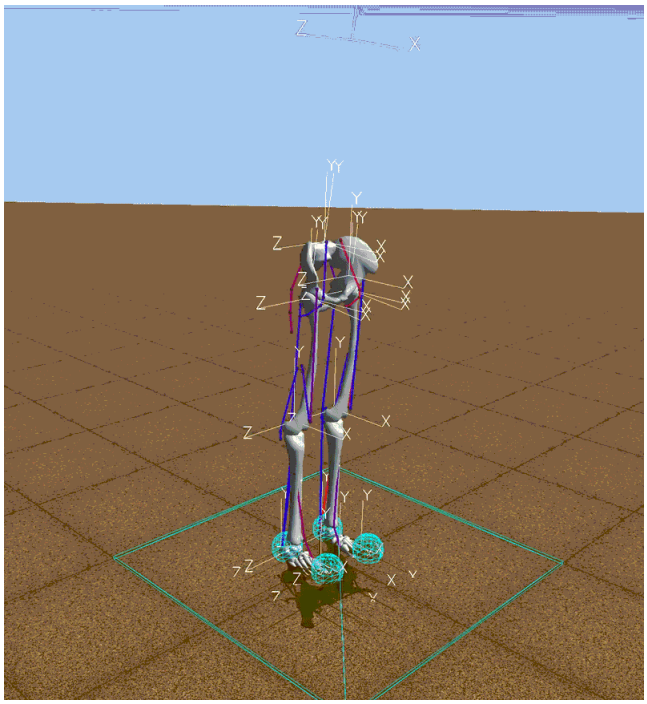


FIGURE 10 – Le corps à l'état initial

Pour faciliter les calculs de reward après, à chaque time-step, nous avons calculé également le point au milieu 2 talus et puis construit un vecteur, en partant de ce point, parallèle au $\vec{Oz}$ si nous souhaitons au modèle de tomber à droit, parallèle au $-\vec{Oz}$ si nous souhaitons au modèle de tomber à gauche et parallèle au $\vec{Ox}$ si nous souhaitons au modèle de tomber devant.

5.3.2 Reward shaping

Nous avons implémenté et testé 3 fonctions de reward différentes et choisi les meilleurs fonctions pour le fine-tuning après.

Pénalisation par l'angle entre la direction à tomber et la direction du corps A chaque time-step, nous avons calculé le vecteur de la direction du corps et le vecteur de la direction à tomber selon ce que nous avons présentés précédemment, et puis nous avons calculé l'angle entre ces 2 vecteurs. Cet angle était compris en $[0, \pi]$ et nous avons retourné $-angle$ comme le reward pour réaliser l'apprentissage. La fonction de reward a retourné donc des valeurs entre $[-\pi, 0]$. Nous espérions que le modèle puisse apprendre la façon pour diminuer la pénalisation qu'il reçoit chaque time-step en diminuant l'angle entre la direction à tomber et la direction du corps. C'est-à-dire le corps aurait une tendance d'approcher la direction à tomber prédéfini de temps en temps.

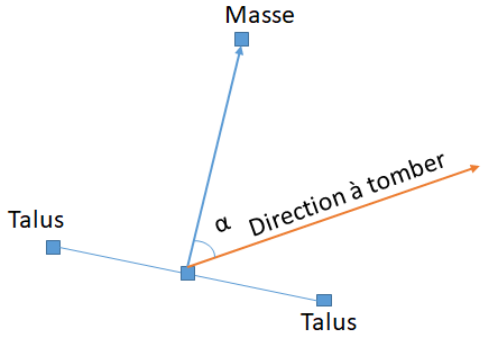


FIGURE 11 – Angle entre la direction à tomber et la direction du corps

Nous avons testé cette fonction de reward avec notre environnement et obtenu le résultat dans la figure 12 suivante.

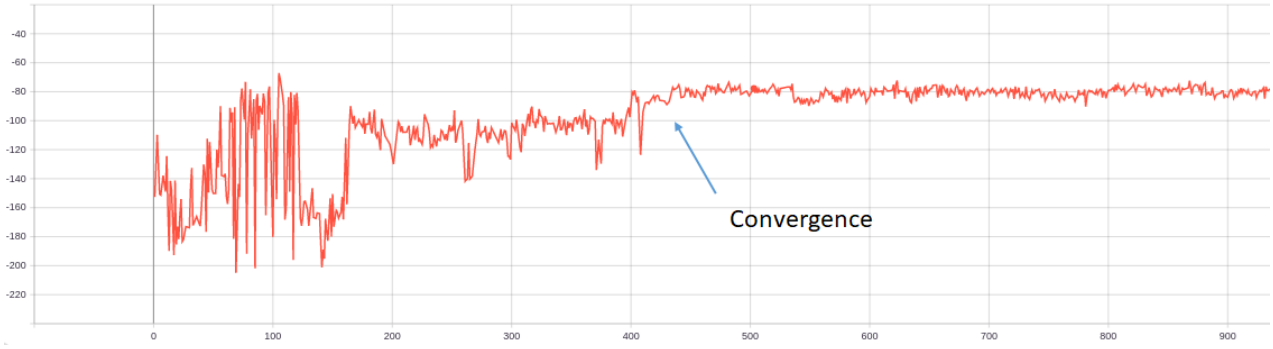


FIGURE 12 – Graphe de reward par épisode pendant l'apprentissage avec la première fonction de reward

Le modèle a convergé très rapidement à la valeur de reward environ -80 par épisode. Nous avons vérifié le fichier log qui a tous les informations pendant l'apprentissage ainsi que vérifié par la visualisation du modèle

pour savoir le comportement du modèle. Le modèle a bien trouvé la façon pour diminuer la pénalisation reçue à chaque épisode mais malheureusement ce n'était pas de diminuer l'angle entre le corps et la direction prédéfinie. Le modèle a essayé de tomber plus rapidement que possible à n'importe quelle direction pour diminuer le nombre de time-step et il pouvait donc diminuer le pénalisation. Par conséquent, nous n'avons pas retenu cette fonction de reward.

Cosinus de l'angle entre la direction à tomber et la direction du corps A chaque time-step, nous avons calculé le vecteur de la direction du corps et le vecteur de la direction à tomber selon ce que nous avons présentés précédemment, et puis nous avons calculé le cosinus de l'angle entre ces 2 vecteurs par la formule : $\cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$. Nous avons choisi la fonction cosinus parce que la valeur retournée par cette fonction était comprise en $[-1, 1]$ et le cosinus de 2 vecteurs était proche à 1 si ces 2 vecteurs étaient proches. C'est-à-dire si 2 vecteurs sont plus proches, nous recevrons plus de reward. Par conséquent, le modèle essayerait d'approcher le corps vers le vecteur de direction à tomber.

FIGURE 13 – Graphe de reward par épisode pendant l'apprentissage avec la 2ème fonction de reward

Le modèle a bien convergé selon le graphe de reward par épisode. Nous avons vérifié et visualisé le comportement du modèle et nous avons obtenu un résultat assez intéressant : le modèle a trouvé la manière de maximiser le reward reçu en tentant le corps vers la direction prédéfinie. Après 5000 épisodes, le corps pouvait tomber avec un angle de 45^o par rapport de la direction prédéfinie. Nous pourrions améliorer ce résultat en faisant le fine-tuning pour trouver des bons hyper-paramètres. Par conséquent, nous avons retenu cette fonction de reward pour le fine-tuning après.

Reward par l'écart entre l'angle de l'état actuel par rapport à la direction prédéfinie et l'angle de l'état précédent par rapport à la direction prédéfinie Comme la première approche, à chaque time-step, nous avons calculé l'angle entre le corps et la direction prédéfinie. Par ailleurs, nous avons calculé également l'angle entre le corps de l'état précédent et la direction prédéfinie. Enfin, nous avons calculé l'écart entre les 2 angles comme le reward. Cette fonction de reward permettrait le modèle à avancer vers la direction prédéfinie à chaque time-step.

Le modèle a convergé comme la dernière approche. Nous avons vérifié le log et nous avons vu que le modèle essairait avancer vers la direction prédéfinie après chaque time-step. La visualisation a bien confirmé notre observation. Comme la dernière fois, le résultat pourrait être amélioré par fine-tuning donc nous avons retenu cette fonction pour la fine-tuning.

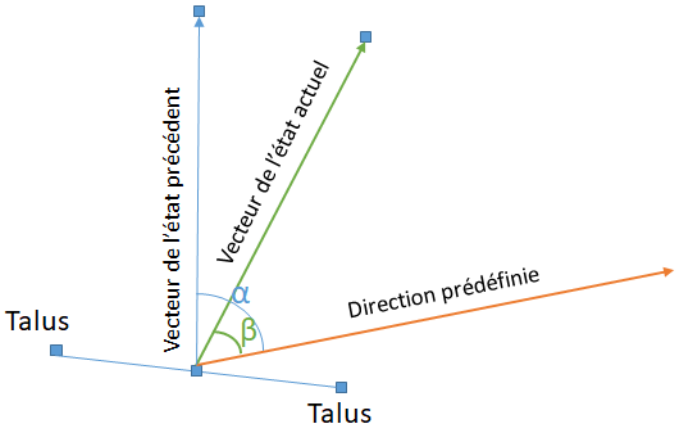


FIGURE 14 – l'écart entre l'angle de l'état actuel par rapport à la direction prédéfinie et l'angle de l'état précédent par rapport à la direction prédéfinie. Dans ce graphe, le reward de ce time-step est égal à $\alpha - \beta$.

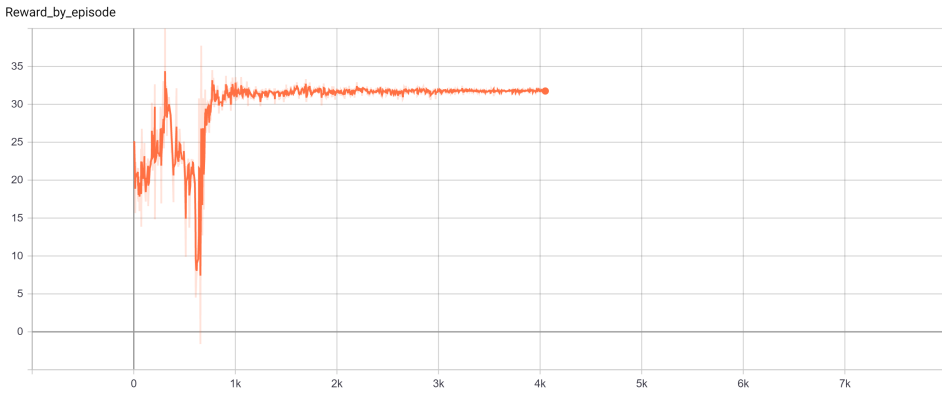


FIGURE 15 – Graphe de reward par épisode pendant l'apprentissage avec la 3ème fonction de reward

5.4 Fine-tuning

Nous avons retenu 2 fonctions de reward pour chercher des bons hyper-paramètres. Il y avait plusieurs hyper-paramètres : la taille du batch, les hyper-paramètres d'OU-Noise, les hyper-paramètres des réseaux neuronaux comme le taux d'appretissage, l'architecture des réseaux, les paramètres de décalage de bruit, les paramètres de l'algorithme comme le taux de la mise à jour τ, le "discount factor" γ, le nombre d'épisodes à parcourir, etc ...

Après quelques essaies, nous avons fixé quelques hyper-paramètres car ils n'étaient pas très effectifs dans l'amélioration de la performance :

— Nombre d'épisodes : 10000 épisodes
— La taille du batch : 32 transitions
— Les hyper-paramètres d'OU-Noise : $\sigma = 0.3$, $\theta = 0.15$. Nous avons fixé ces hyper-paramètres car nous avons utilisé le décalage de bruit pour changer le bruit pendant l'apprentissage.

— Nous avons fixé $\tau = 0.001$ et $\gamma = 0.99$. Nous avons choisi $\tau = 0.001$ car nous voulons ralentir la vitesse d'apprentissage car éviter de tomber au point optimum local. $\gamma = 0.99$ permettrait d'apprendre presque tout le Temporal Difference.
— Nous n'avons utilisé qu'une seule architecture des couches cachées de Actor et de Critic dans la cadre de ce projet car nous n'avons pas beaucoup de temps pour essayer des autres architectures potentielles.

Les autres hyper-paramètres étaient optimisé pour améliorer la performance :

— 2 approches des réseaux neuronaux de Actor et de Critic : Comme ce que nous avons représenté dans la première partie, nous avons essayé 2 approches différentes : une avec l'état du corps (les muscles, les acticulations, les joints, etc ...) comme l'entrée du réseau et une avec l'angle entre le corps et la direction prédéfinie en entrée.
— Noise decay ou le décalage de bruit : Nous avons utilisé un facteur pour standardiser les valeurs générées par OU-Noise aux valeurs raisonnables pour cet environnement. Par ailleurs, nous souhaitons diminuer l'effet du bruit de temps en temps pendant l'apprentissage. C'est-à-dire au début de l'apprentissage, le modèle exploiterait en utilisant un grand bruit. A la fin de l'apprentissage, le modèle n'exploiterait plus et il utiliserait ses connaissances de l'environnement pour obtenir plus de reward que possible. En général, nous avons utilisé 2 hyper-paramètres pour régler l'exploitation du modèle.
— Les taux d'apprentissage de Actor et Critic étaient choisis pour optimiser car nous souhaitons contrôler la vitesse d'apprentissage. Si le modèle apprend trop rapidement, il tombera au point optimal local.

Pour le fine-tuning, nous avons utilisé une méthode très utilisée : Grid Search. Nous avons combiné plusieurs valeurs de ces hyper-paramètres à optimiser pour trouver un meilleur modèle.

Pour évaluer les modèles obtenus, nous avons développé une fonction qui nous permettrait de visualiser leurs comportements. Par ailleurs nous avons considéré le reward total obtenu chaque épisode pour évaluer le modèle.

Après le fine-tuning de 2 fonctions de reward présentées dans la partie précédente avec 2 types de hyper-parramètres pendant 2 semaines, nous avons obtenu les résultats :

— Les comportements des 2 approches étaient assez identiques. Cependant, la performance de la 2ème approche (l'angle entre le corps et la direction prédéfinie comme entrée des réseaux) était un peu mieux. Pendant l'apprentissage, le modèle par cette approche devenait instable de temps en temps lors que le modèle entraîné par la première approche était stable.
— Le décalage de bruit contribuait un grand effet à l'apprentissage du modèle et nous avons obtenu les résultats identiques pour toutes les 2 fonctions de reward. Nous avons essayé 3 valeurs pour le facteur de standardisation de bruit : 0.8 (diminuer le bruit), 1 (ne pas normaliser) et 1.5(augmenter l'effet du bruit). Pour la valeur de 1.5, nous avons observé que l'apprentissage était très lent et la performance du modèle n'était pas bonne. Pour la valeur du facteur de standardisation de bruit de 0.8 ou 1, l'apprentissage était plus stables et nous avons un résultat assez intéressant. Nous avons également essayé 2 valeurs pour le taux de décalage de bruit : 0.95 et 0.5. Pour la valeur de 0.5, le modèle a convergé rapidement mais il a tombé au optimum local. Sa manière pour optimiser le reward était de tomber plus vite que possible. Par conséquent, nous avons retenu 0.8 pour le facteur de standardisation de bruit et 0.95 pour le taux de décalage de bruit.
— Nous avons essayé plusieurs valeurs pour les taux d'apprentissage de Actor et de Critic. Pour le taux d'apprentissage de Actor, nous avons utilisé les valeurs : 0.01, 0.001, 0.0001, 0.00001, 0.000001 et nous

avons essayé 0.1, 0.01, 0.0001, 0.00001 pour Critic. Nous avons trouvé que avec le taux d'apprentissage de Actor égal à 0.00001 et le taux d'apprentissage de Critic égal à 0.001 nous avons obtenu le meilleur modèle pour la 2ème fonction de reward (reward par le cosinus de l'angle). Pour la 3ème fonction de reward (le décalage de l'angle), nous avons obtenu le meilleur modèle quand le taux d'apprentissage de Actor était égal à 0.00001 et le taux d'apprentissage de Critic était égal à 0.0001.

6 Résultat - Discussion

Pendant le fine-tuning, nous avons optimisé 2 fonctions de reward et 2 approches pour les réseaux neuronaux, chacune a obtenu 2 modèles de tomber à droit. Nous avons trouvé que le comportement des modèles entraînés par 2 fonctions de reward étaient identiques. Notre stratégie pendant la visualisation de modèle était de ne réaliser des actions que chaque 3 time-steps. Nous avons pris cette idée du journal de la compétition en 2017.

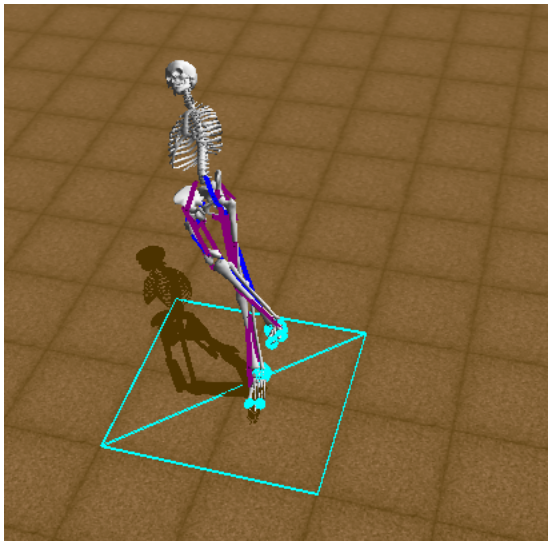


FIGURE 16 – Le modèle tombe à droit vers un angle de 45^o par la première approche

Nous pouvons trouver pendant la visualisation que le comportement des modèles entraînés par 2 approches différentes étaient aussi identiques. Pour tomber à droit, le modèle a essayé de lever la jambe droit plus haute que possible. L'angle entre le corps et la direction prédéfinie était de 39^o à 45^o. Nous avons vérifié le log qui a eu tous les états du corps pendant l'apprentissage et nous avons trouvé que le modèle a rarement tombé avec un angle entre le corps et la direction prédéfinie inférieur à 37^o et il n'a tombé jamais avec un angle inférieur à 30^o même si nous avons ajouté beaucoup de bruits pendant l'apprentissage.

Pour la première approche de réseaux neuronaux, nous avons obtenu des modèles qui peuvent tomber à un angle de 45^o par rapport la direction prédéfinie. C'était le meilleur résultat de cette approche si nous utilisons ces fonctions de reward car Actor n'a pas connu la direction actuelle du corps qui était utilisée pour calculer le reward. Ce résultat pourra être amélioré. En effet, si nous avons plus des informations sur l'effet des muscles, les joints et les forces sur le mouvement du modèle, nous pourrons faire une fonction de reward plus directe au lieu d'une fonction de reward par l'angle entre le corps et la direction prédéfinie. Nous pouvons observer dans la figure 18 et la figure 19, le modèle entraîné par la fonction de reward avec le décalage de l'angle était plus stable et il a convergé plus rapidement. Par contre, ces 2 modèles ont obtenu le même résultat pendant la visualisation.

Nous avons obtenu des mieux modèles avec la 2ème approche. C'était raisonnable car Actor a connu la direction actuelle du corps donc il a eu des connaissances plus directes pour améliorer le reward. Cependant, cette approche a eu plusieurs problèmes. Tout d'abord, l'apprentissage était très instable même si nous avons

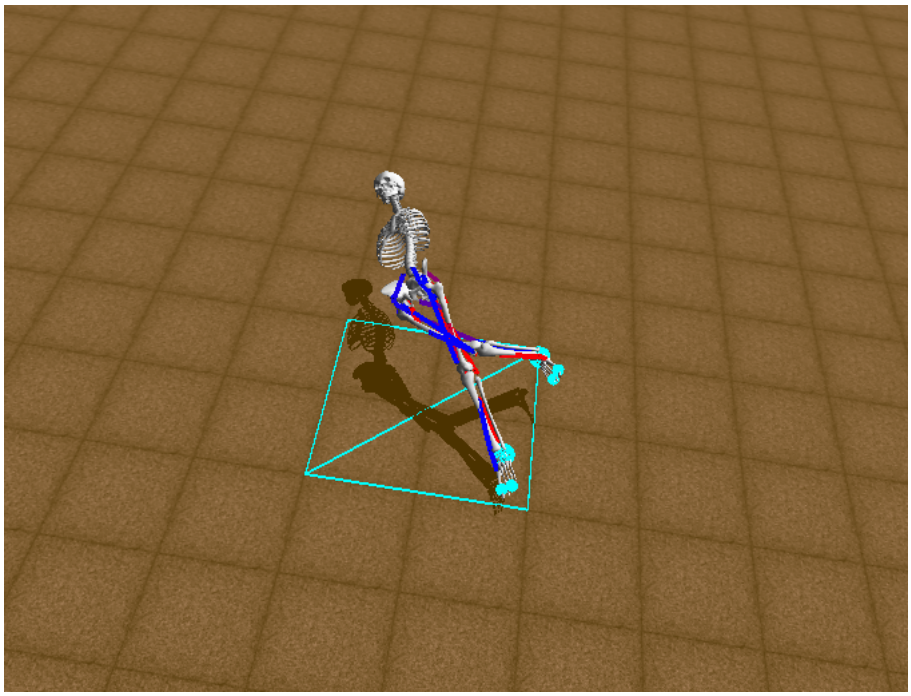


FIGURE 17 – Le modèle tombe à droit vers un angle de 39^o par la 2ème approche

FIGURE 18 – Graphe de reward par épisode pendant l'apprentissage avec la fonction de reward par cosinus entraîné par la première approche

utilisé Prioritized Experience Replay. En effet, l'état qui était l'angle entre le corps et la direction prédéfinie était trop simple à interpréter le "vrai" état du corps pour choisir des bonnes actions. Le modèle dépendait donc de l'initialisation aléatoire de paramètres dans les réseaux Actor et Critic. Par conséquent, c'était très difficile pour le fine-tuning. Le 2ème problème était qu'il y avait plusieurs valeurs possibles des muscles, des joints, etc ... pour une valeur de l'angle entre le corps et la direction prédéfinie. Par conséquent, avec un seul angle, Actor ne pouvait pas comprendre totalement l'état actuel du corps pour avoir un bonne action. Nous pouvons observer dans le graph de reward obtenu dans un épisode et le nombre de steps dans un épisode que le modèle devient de plus en plus instable pendant l'apprentissage car l'entrée de Actor ne pouvait pas comprendre tout l'état du corps et donc il a donné des actions très différentes. en analysant le résultat dans la figure 20 et la figure 21 nous pouvons observer que la performance de la fonction de reward par décalage de l'angle était un peu mieux. Après la vérification par la visualisation, nous avons retenu ce modèle de cette approche.

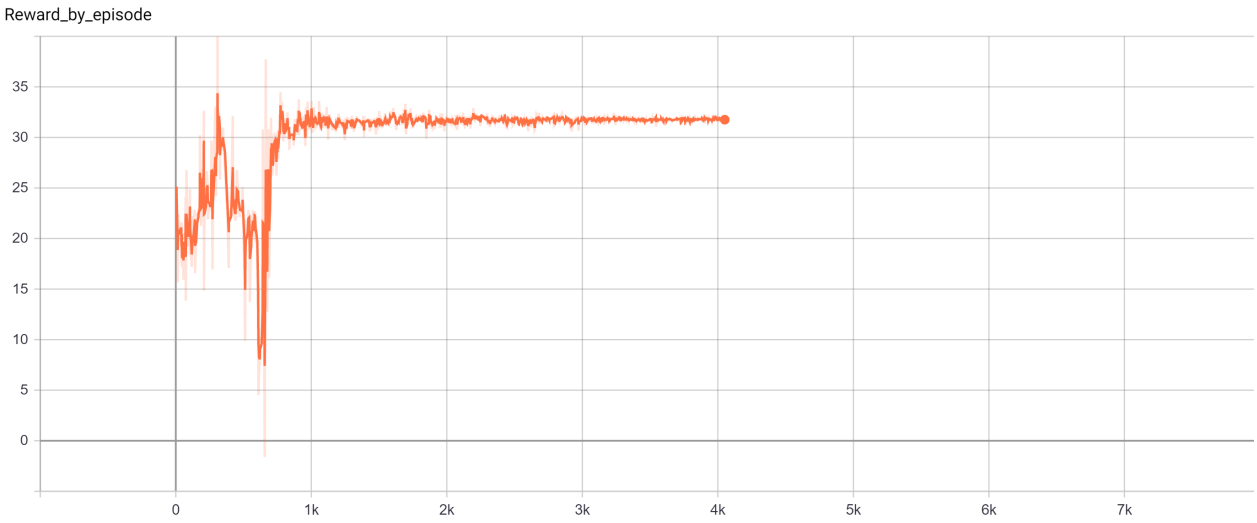


FIGURE 19 – Graphe de reward par épisode pendant l'apprentissage avec la fonction de reward par décalage de l'angle entraîné par la première approche

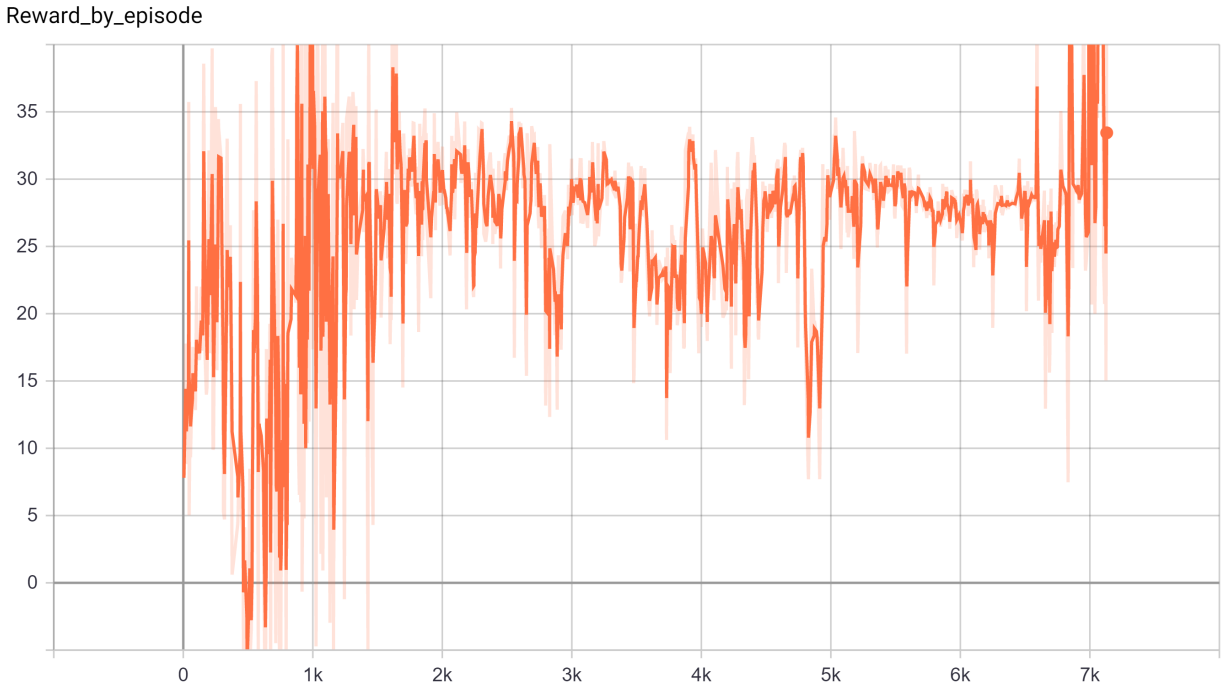


FIGURE 20 – Le reward total obtenu dans un épisode pendant l'apprentissage de la 2ème approche avec la fonction de reward par décalage de l'angle

Nous avons également affiché les graphes des états du corps (les muscles et les articulations) pendant la visualisation (la figure 22 et 23). Nous avons observé que les valeurs des muscles ont été changé par les actions générées par le modèle dans 3 premiers time-steps, et puis le modèle a essayé de maintenir les muscles aux valeurs constantes. Ces valeurs étaient plupart des valeurs extrêmes (0 et 1). Les articulations ont été activé pendant 20 premiers time-steps, quelques articulations ont continué à varier jusqu'à la fin lors que

FIGURE 21 – Le reward total obtenu dans un épisode pendant l'apprentissage de la 2ème approche avec la fonction de reward par cosinus

quelques autres ont resté à constant. Ces 2 types d'articulations peuvent avoir des informations importants sur le mouvement du modèle.

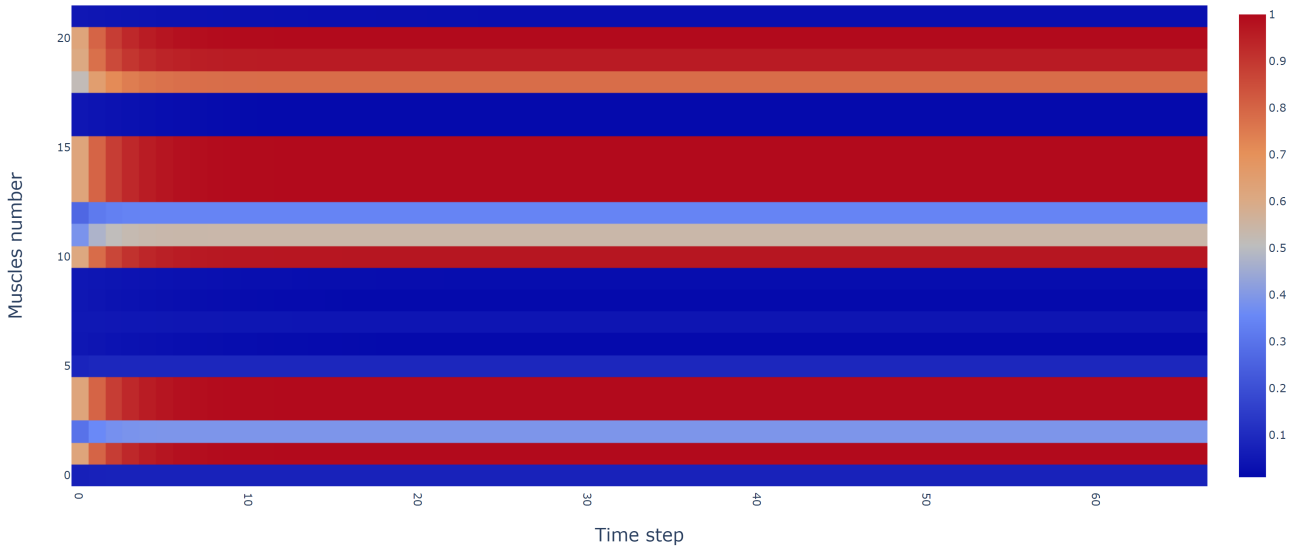


FIGURE 22 – Les muscles pendant la visualisation

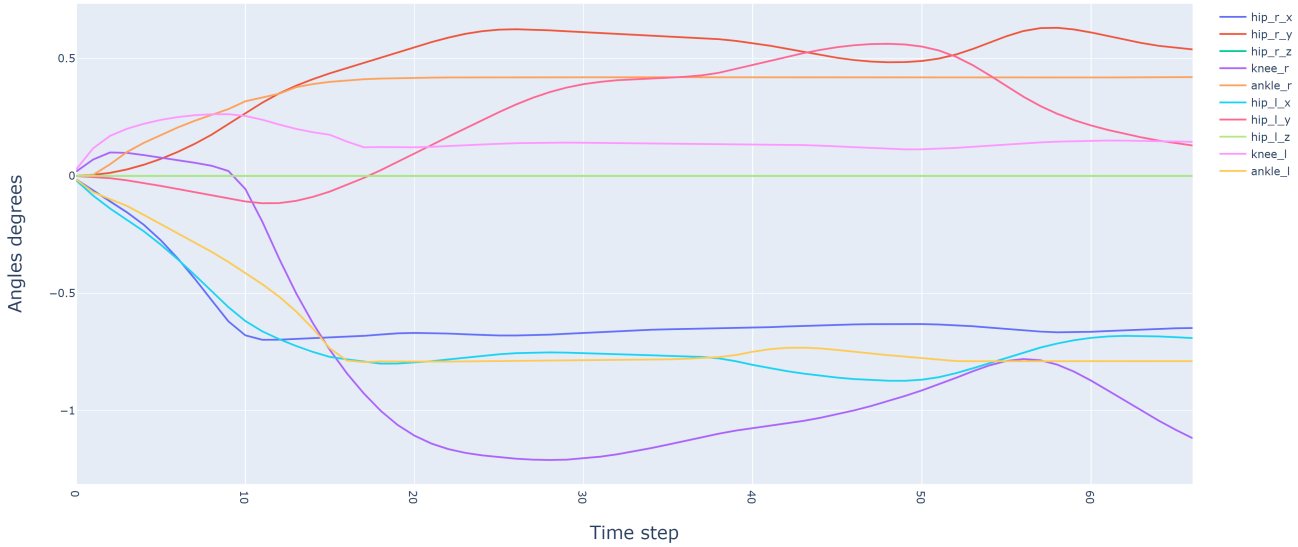


FIGURE 23 – Les articulations pendant la visualisation

Nous avons pensé à combiner ces 2 approches en ajoutant l'angle du corps et la direction prédéfinie à l'ensemble d'états de l'environnement mais il serait difficile pour le modèle comprendre la différence entre cet angle et les autres facteurs dans l'ensemble d'états. Nous pouvons ajouter des poids qui indique l'importance de ces facteurs aux réseaux de Actor et Critic juste après l'entrée des réseaux. Par conséquent, le modèle pourra comprendre de temps en temps la différence entre l'angle et les autres facteurs (Nous avons pris l'idée de la mécanisme "Attention" très utilisée en Traitement de la langue naturelle (NLP)). Par contre, il faut refaire la fonction de reward pour mieux évaluer les actions et leurs effets sur le nouvel ensemble d'états (l'angle et les facteurs par défaut de l'environnement). Nous pourrons également améliorer la performance des modèles en ajoutant des bruit aux paramètres au lieu aux actions car les modèles peuvent mieux exploiter l'environnement. Enfin, pour que les actions et les états soient mieux liées pendant l'apprentissage, nous avons besoin des plus des informations et des connaissances concernant l'effet des facteurs dans cet environnement à la direction du corps et puis les modéliser à une fonction mathématique pour améliorer la fonction de reward.

7 Conclusion

En conclusion, Reinforcement Learning représentent un champ de plus en plus exploré en informatique et notamment dans le domaine de la biologie et du médicament car il permet de simuler des processus qui seraient irréalisables dans la réalité.

Dans ce projet, nous avons implémenté l'algorithme Deep Deterministic Policy Gradient avec Actor-Critic pour réaliser un modèle qui peut tomber dans une direction choisie. La stratégie d'apprentissage Experience Replay et notamment Prioritized Experience Replay ont été implémentées et appliquées qui nous permettent de stabiliser l'apprentissage ainsi que s'assurer la convergence du modèle. Nous avons également analysé les résultats obtenus pendant l'apprentissage pour le fine-tuning. A la fin du projet, nous avons obtenu 2 modèles qui peuvent tomber à droit avec un angle de 45^o et 30^o. Pour pouvoir tomber à droit, le modèle essaie de lever la jambe droit plus haut que possible et le corps devient déséquilibre et donc tombe à droit. Ces résultats peuvent être améliorés si nous avons plus d'informations sur l'effet des états par défaut sur la direction du corps et si nous avons une manière plus précise pour déterminer la direction actuelle du corps.

Ce rapport fait la synthèse de nos recherches durant le semestre sur l'application de Reinforcement learning dans un problème biologique. Nous avons découvert un champ informatique que nous n'avons jamais rencontré auparavant. De plus, le côté expérimental du projet nous a amené à développer nos compétences dans le domaine de Data Science qui seront importants pour nos projets professionnels.

Références

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay, 2017.

[2] G. Chen and Y. Peng. Off-policy actor-critic in an ensemble : Achieving maximum general entropy and effective environment exploration in deep reinforcement learning, 2019.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2015.

[4] M. Pavlov, S. Kolesnikov, and S. M. Plis. Run, skeleton, run : skeletal model in a physics-based simulation, 2017.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2015.

[6] stanford. Opensim rf. `http://osim-rl.stanford.edu/docs/home/`.

[7] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[8] Łukasz Kidziński, S. P. Mohanty, C. Ong, Z. Huang, S. Zhou, A. Pechenko, A. Stelmaszczyk, P. Jarosik, M. Pavlov, S. Kolesnikov, S. Plis, Z. Chen, Z. Zhang, J. Chen, J. Shi, Z. Zheng, C. Yuan, Z. Lin, H. Michalewski, P. Miłoś, B. Osiński, A. Melnik, M. Schilling, H. Ritter, S. Carroll, J. Hicks, S. Levine, M. Salathé, and S. Delp. Learning to run challenge solutions : Adapting reinforcement learning methods for neuromusculoskeletal environments, 2018.

[6] [3] [5] [5] [1] [7] [4] [2] [8]