RAPPORT DU PROJET

Développement d'environnements Pybullet Gym pour l'évaluation de politiques de perception active

DANS LE CADRE DU

MASTER INFORMATIQUE

PARCOURS

ANDROIDE

RÉALISÉ PAR

Nicolas DESCHAR
Alix ZHENG

ENCADRÉ PAR

Stéphane DONCIEUX
Achkan SALEHI

(Février 2021 - mai 2021)

Table des matières

1 Introduction **3**

2 Classification de la base de données MNIST **4**
2.1 Maximum de vraisemblance (MV) 6
2.2 Réseau de neurones 7
2.2.1 Perceptron multicouche à rétropropagation (RNPR) 7
2.2.2 Réseau de neurones convolutifs (CNN) 8
2.2.3 Tests des classifieurs 10

3 Méthode **13**

4 Environnement **13**

5 Fonctions de récompenses **14**
5.1 Simple 14
5.2 Un peu plus complexe 14
5.3 Observation à un pas de temps donnée 14

6 Algorithme de renforcement pour la perception active **15**
6.1 Détails de l'intérêt de l'algorithme deep-Q 15
6.2 Algorithme pas à pas 16
6.2.1 Initialisation 16
6.2.2 Boucle 16

7 Résultats **17**

8 Discussions **18**

9 Conclusion **19**

10 Annexe **20**
10.1 Cahier des charges 20

Références **22**

REMERCIEMENT

Nous tenons à remercier infiniment nos deux encadrants : Stéphane Doncieux et Achkan SALEHI pour nous guider, répondre à nos questions tout au long du projet et nous aider à aboutir nos objectifs.

1 Introduction

L'apprentissage par renforcement consiste pour un agent à apprendre de façon autonome la meilleure action à prendre à partir d'expériences, de façon à optimiser une récompense au cours du temps dans un environnement inconnu [1] .

Le problème de l'apprentissage par renforcement est un problème de l'apprentissage basé sur des interactions successives avec un environnement dans le but d'atteindre un objectif. Ce type de problème est décrit par un modèle mathématique : le Processus de décision markovien.

L'agent interagit avec l'environnement selon le schéma de la Figure 1 ci-dessous :

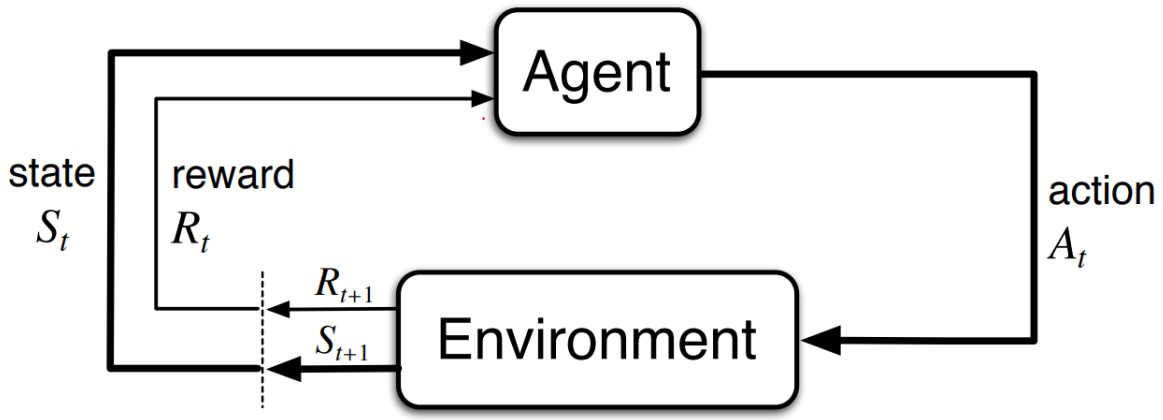


FIGURE 1 – L'interaction agent-environnement dans l'apprentissage par renforcement

Le processus de décision markovien est décrit par 4 éléments :
— $\mathcal{S}$: espace d'état
— $\mathcal{A}$: espace d'action
— $T : \mathcal{S} \times \mathcal{A} \to \mathcal{A}$: fonction de transition
— $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: fonction de récompense

A l'instant t, l'agent se situe à l'état S_t effectuera une action $A_t \in \mathcal{A}(S_t)$, où $\mathcal{A}(S_t)$ est l'ensemble des actions possible à l'état S_t. A_t est dictée par une politique :

$$\begin{array}{rccc} \pi & : & \mathcal{S} & \to & \mathcal{A} \\ & & S_t & \mapsto & \pi(S_t) = A_t \end{array}$$

Une fois l'action A_t effectuée, l'agent reçoit une récompense R_t de son environnement et effectue une transition vers l'état S_{t+1}. L'objectif d'un tel problème est de trouver la politique qui maximise la récompense totale.

En robotique, la perception active désigne de manière générale tout processus visant à améliorer la qualité du flux de données entrant dont la tâche en cours dépend. L'interprétation de données sensorielles est indispensable pour sélectionner les comportements pertinents pour améliorer l'information obtenue à partir du flux de données que ces comportement produisent dans un environnement. Un agent change de position pour améliorer la vue d'un objet spécifique ou utilise le mouvement pour mieux percevoir l'environnement. Il s'agit de plus de la capacité d'un agent à effectuer une action afin de réduire l'erreur d'une estimation d'un sous-ensemble des états de l'environnement.

Par exemple, il peut servir en vision par ordinateur à guider la recherche de correspondances entre points d'intérêt dans des images afin d'estimer l'état de l'agent et de l'environnement [2] ou à éviter les occlusions afin de conserver un objet d'intérêt dans le champ de vision [3].

Dans le cadre du projet, notre but sera de développer un environnement OpenAI gym [4] qui servira à l'apprentissage et à l'évaluation de politiques, en se basant sur le moteur physique bullet [7] . L'environnement sera composé :

— Un ensemble de N cubes (de même taille) texturés et contenant un chiffre de la base de MNIST sur une de leurs faces.
— Une caméra pouvant se déplacer autour des cubes.

La base de données MNIST -Modified National Institute of Standards and Technology- est une base de données de chiffres écrits à la main, elle contient un très grand nombre d'images pour permettre d'apprendre et de faire les tests de reconnaissance des chiffres.

La position des cubes sera fixée à une élévation h positive mais leurs orientations ainsi que la position de la caméra seront initialisées aléatoirement. La perception de l'agent sera le flux d'images recueilli par la caméra, qui sera donné en entrée à un classifieur chargé de classifier les chiffres. Nous définirons des fonctions de récompenses à l'aide du classifieur. L'objectif sera donc de classifier correctement les N chiffres présents dans l'environnement.

Nous avons suivi les tutoriels [5] [6] pour la phase classification.

2 Classification de la base de données MNIST

Les images de cette base sont de taille $28 \times 28 = 784$ et sont codés en niveau de gris. La base MNIST contient 6000 images d'entrainement et 1000 images de test.

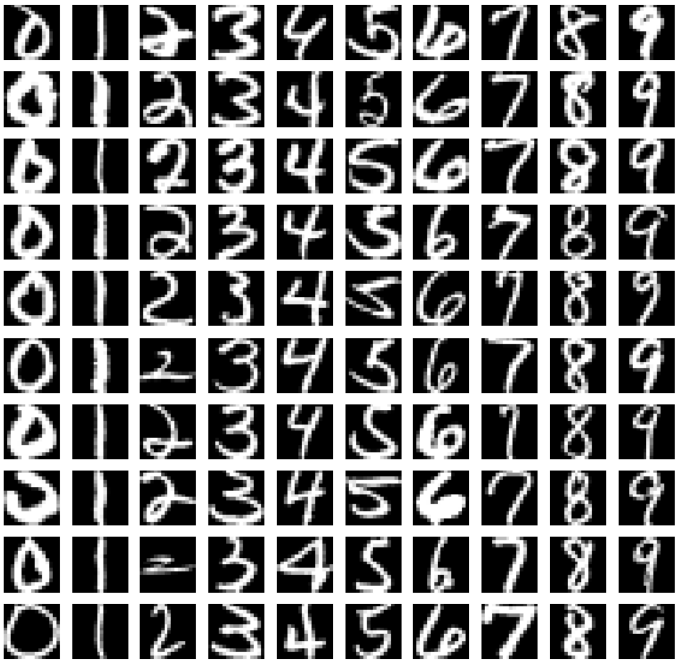


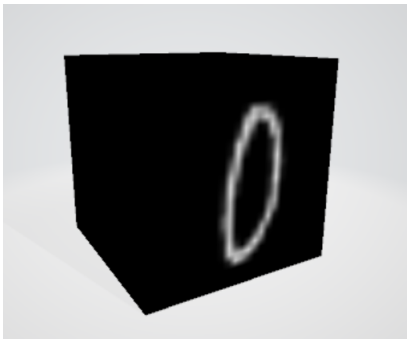
FIGURE 2 – Quelques images de la base de données MNIST

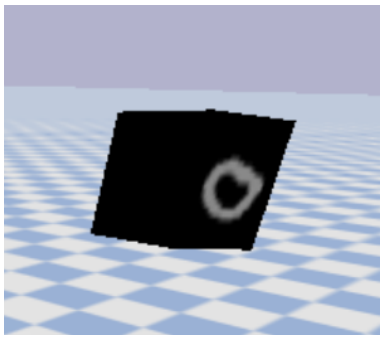
Nous avons ajouté 1500 images pour la 11e classe, dont 1000 pour l'entrainement et 500 pour le test.

Pour la classification de la base de données MNIST, nous avons cherché plusieurs méthodes :

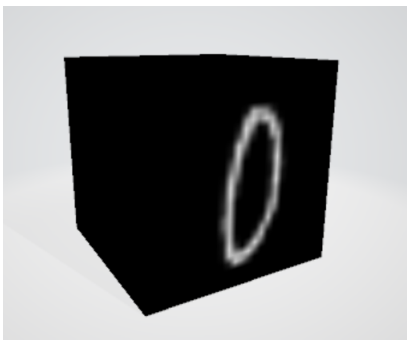
1. Maximum de Vraissemblance
2. Reseaux de neurones :
   — Perceptron multicouche à rétropropagation
   — Convolutionnel avec quelques couches entièrement connectées.

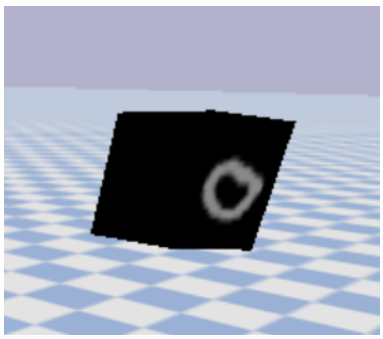
Nous avons 11 classes, les 10 premières classes correspondent aux chiffres de $[\![0, 10]\!]$, la 11e classe regroupe les images qui ne contiennent aucun digit. Pour la 11e classe, les images que nous avons pris pour l'entraînement et les tests sont issus de la base de données CIFAR [9], nous les avons choisi au hasard. Cependant, il a été plus pertinent de prendre les images de notre environnement pour l'apprentissage, pour avoir de meilleurs résultats lors de l'apprentissage pour l'agent. Ainsi, nous avons rajouté des images de la caméra contenant des cubes où un chiffre est lisible, bien que loin de la caméra.





Pour la classe 11, les images sont de cette forme :





2.1 Maximum de vraisemblance (MV)

La méthode du maximum de vraisemblance a été d'abord utilisé car nous avons des connaissances sur cette méthode. L'idée est de modéliser un pixel de l'image à l'aide de la loi de Bernouilli. En effet, les pixels sont codés en niveau de gris et la surface du cube est noire exceptée la texture du digit.

Soit les indices i donnant les images et les indices j référant aux pixels dans l'image, nous cherchons à déterminer p_j la probabilité d'illumination d'un pixel j pour une collection d'image d'une seule classe

Collection d'une classe :

$$X = \{\mathbf{x_i}\}_{i=1,\dots,N}, \qquad \mathbf{x_i} \in \{0,1\}^{784}$$

Modélisation de la variable de Bernoulli X_j, valeur du pixel j :

$$p(X_j = x_{ij}) = p_j^{x_{ij}}(1-p_j)^{(1-x_{ij})} = \begin{cases} p_j & \text{si } x_{ij} = 1 \\ 1-p_j & \text{si } x_{ij} = 0 \end{cases}$$

La vraisemblance : $\mathcal{L}(X,\theta) = p(X_0,...,X_{784}|\theta) = \prod_{i=0}^{784} p(X_i)$

Le maximum de vraisemblance d'une loi de Bernouilli pour p_j est $p_j^\star$:

$$p_j^\star = \frac{\sum_i x_{ij}}{N}$$

Nous obtenons les paramètres optimaux de chaque classe au sens du maximum de vraisemblance :

$$[p_0^\star, \dots, p_{783}^\star]$$

Calcul du maximum de vraisemblance pour estimer θ :

$$arg\max_\theta \mathcal{L}(X,\theta) = arg\max_\theta log\mathcal{L}(X,\theta)$$

Ce qui revient à calculer

$$\nabla_\theta \mathcal{L}(X,\theta) = 0$$

Nous appliquons le log-maximum de vraisemblance, ce qui nous donne :

$$\log p(x_i|\theta^{(c)}) = \log \prod_j p(X_j = x_{ij}) = \log \prod_j p_j^{x_{ij}}(1-p_j)^{x_{ij}}$$

$$\log p(x_i|\theta^{(c)}) = \sum_j \log p(X_j = x_{ij})$$

$$\log p(x_i|\theta^{(c)}) = \sum_j x_{ij} \log p_j + (1-x_{ij}) \log(1-p_j)$$

Il faut ensuite calculer pour tout $c \in [\![0, 11]\!]$ et ainsi obtenir pour une image la liste de probabilité d'appartenance à chaque classe c :

$$\log(P(x_i|\theta) = \left[\begin{array}{cccc} \log p(x_i|\theta^{(0)}) & \log p(x_i|\theta^{(1)}) & \dots & \log p(x_i|\theta^{(10)}) \end{array} \right]$$

L'estimation de la classe de l'image x_i est donnée par cette formule :

$$c^\star = arg\max_c \log(P(x_i|\theta)$$

2.2 Réseau de neurones

Il est possible de classifier avec un réseau de neurones. Ainsi, nous avons cherché à implémenter deux sortes de réseaux.

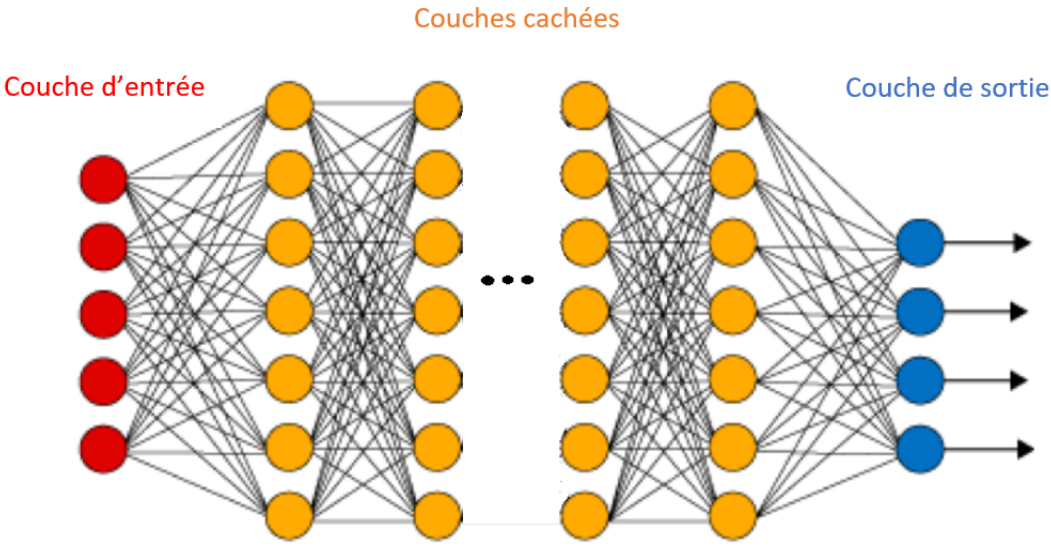


FIGURE 3 – Forme d'un réseau de neurone

2.2.1 Perceptron multicouche à rétropropagation (RNPR)

Le perceptron multicouche est un réseau à propagation directe constitué de plusieurs couches entièrement connectées au sein desquelles une information circule de la couche d'entrée vers des couches cachées. La dernière couche de neurones est appelée couche de sortie et correspond au retour du réseau. Les informations sont propagées par une fonction d'activation. Une fonction d'activation est une fonction appliquée à la sortie de chaque neurone. Durant la phase d'entraînement, après avoir calculé les erreurs du réseau de neurones, il est nécessaire de les corriger afin d'améliorer ses performances : c'est la rétropropagation.

Pour la propagation directe, nous utilisons une transformation linéaire. Le principe est décrit par la Figure 3. Sur l'exemple, chaque neurone est multiplié par une matrice de poids w_i, puis additionné par une valeur de biais pour la régularisation.

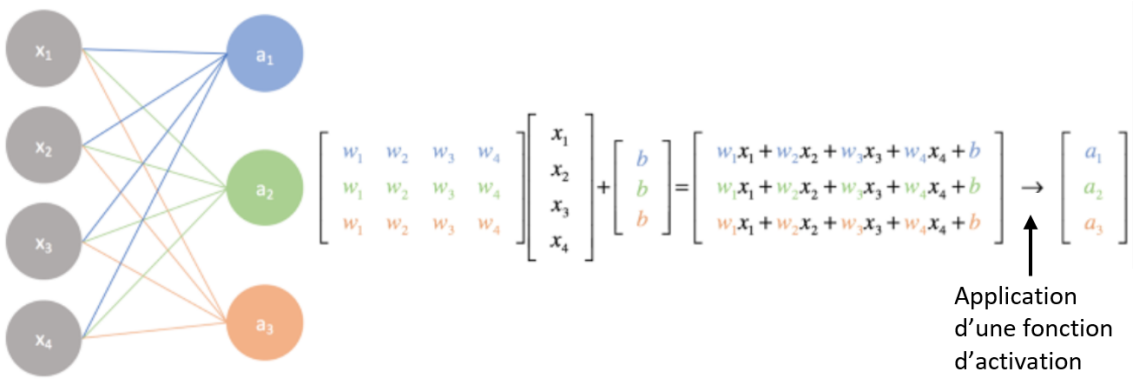


FIGURE 4 – Exemple sur un réseau simple pour le CNN

Dans notre cas, notre couche d'entrée est composée de 784 neurones car il y a 784 pixels dans une image de la base MNIST, la couche de sortie est de 11 neurones pour chacun des chiffres de 0 à 10 et la classe 11. Nous appliquons à les couches cachées, la fonction d'activation reLu :

$$reLu(x) = \max(0, x)$$

Nous appliquons à la dernière couche une fonction softmax :

$$softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j}$$

Ainsi, la sortie du réseau est un vecteur dont la somme des éléments est égale à 1.

Pour la rétropropagation, nous utilisons la méthode de la descente de gradient pour minimiser l'erreur de sortie du réseau de neurones avec le résultat attendu.

L'erreur entre la sortie donnée par le réseau et la sortie attendue est calculée en utilisant la fonction entropie croisée, une fonction beaucoup utilisée pour les problèmes de classification :

$$\mathcal{L}(y) = \sum_i -log(y_i)$$

où y désigne la sortie du réseau de neurones et i une classe parmi les 11 possibles.

Ainsi, pour la descente de gradient, il faut calculer

$$\frac{\partial L(y)}{\partial y_i} = 0$$

2.2.2 Réseau de neurones convolutifs (CNN)

Nous pouvons aussi utiliser un réseau de neurones convolutifs ou réseau de neurones à convolution. Le réseau est défini par :

- deux couches de convolutions définies par un noyau de convolution de taille 5×5 alternés par une couche de max-pooling
- le résultat est amélioré par 2 couches entièrement connectées pour améliorer la pertinence du résultat.

Le pas (stride en anglais) est de 1. Il représente le décalage du noyau de convolution entre chaque calcul (un pas de 1 déplace le noyau d'un pixel par incrément). Le padding permet de redimensionner l'image sortie et éviter les dépassements lors de l' application de la convolution. L'image d'entrée étant carrée, il n'est pas nécessaire et est donc de 0 dans notre cas.

Le max-pooling est calculé en sélectionnant un carré de pixels de taille 2×2 (pour un pooling de 2×2) puis en prenant la valeur maximale de cette portion. Ensuite, le carré est décalé vers la droite du nombre de pas de cases, puis l'opération est répétée.

La couche de pooling permet de réduire la taille d'une image intermédiaire de chaque couche de convolution, réduisant ainsi la quantité de paramètres et de calculs dans le réseau pour ainsi réduire le sur-apprentissage. C'est pourquoi nous avons inséré une couche pooling entre deux couches de convolutions successives, la taille du noyau de pooling est de 2×2 et le pas est de 2.

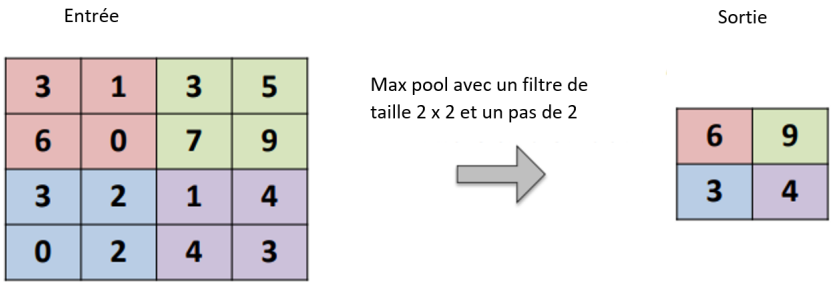


FIGURE 5 – Exemple sur un réseau simple pour le CNN

Le calcul du noyau de convolution appliqué à l'image est :

$$J(x,y) = (I \star h)(x,y) = \sum_{m=-\frac{d-1}{2}}^{\frac{d-1}{2}} \sum_{n=-\frac{d-1}{2}}^{\frac{d-1}{2}} I(x-m, y-n)h(m,n)$$

I est l'image d'entrée de taille $m \times n$, $J(x,y)$ représente la nouvelle valeur du pixel (x,y) de l'image I et h est le noyau de convolution de taille $d \times d$.

La sortie d'une couche de convolution est de taille :

$$(\left\lfloor \frac{N+2.P-F}{S} \right\rfloor + 1) \times (\left\lfloor \frac{N+2.P-F}{S} \right\rfloor + 1)$$

pour une entrée de taille $N \times N$ dans la couche de convolution, une taille de pooling $P \times P$, une taille de noyau de convolution $F \times F$, et une taille du stride $S \times S$.

Pour la rétropropagation, nous utilisons aussi la méthode de la descente de gradient pour minimiser l'erreur entre la sortie du réseau de neurones et le résultat attendu.

L'erreur entre la sortie donnée par le réseau et la sortie attendue est calculée en utilisant la fonction entropie croisée, une fonction beaucoup utilisée pour les prolèmes de classification :

$$p_i = \frac{exp(y_i)}{\sum_j exp(y_j)}$$

$$\mathcal{L}(y) = \sum_i -log(p_i)$$

y la sortie du réseau de neurones, i représente une classe parmi tous les j classes.

Ainsi, pour la descente de gradient, il faut calculer

$$\frac{\partial L(y_i)}{\partial y_i} = 0$$

La sortie du réseau de neurones étant un vecteur de taille 11 dont la somme des éléments est égale à 1.

2.2.3 Tests des classifieurs

Pour trouver le nombre de couches cachées et le nombre de neurones par couche pour le perceptron multicouche, nous avons :

1. testé pour un très grande nombre de couches (200) et de neurones (200). Puis, nous avons réduit le nombre de neurones et de couches jusqu'à avoir un réseau qui prédit le bon chiffre,
2. testé en changeant le nombre de neurones par couches et le nombre d'entraînements.

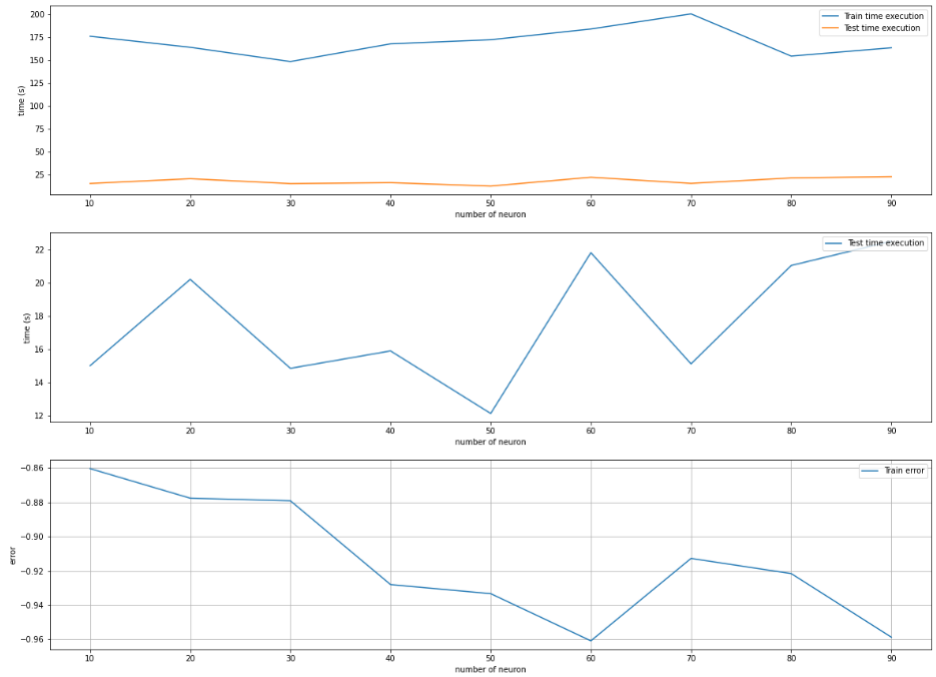


FIGURE 6 – Tests du nombre de couches cachées pour le RNPR

Nous observons que 40 neurones par couches cachées suffisent pour avoir une erreur faible.

L'erreur représentée sur la courbe est la moyenne des erreurs parmi les images mises à l'entraînement.

De même pour trouver le nombre de couches entièrement connectées pour le réseau de neurones convolutifs, nous avons fait une série de tests en changeant le nombre de neurones par couches et le nombre d'entraînement.

Nous observons sur la 3e courbe de la Figure 5 ci-dessous que deux entraînements suffisent pour avoir une erreur faible sans sur-apprentissage, car après deux entraînement, nous pouvons voir une augmentation de l'erreur.

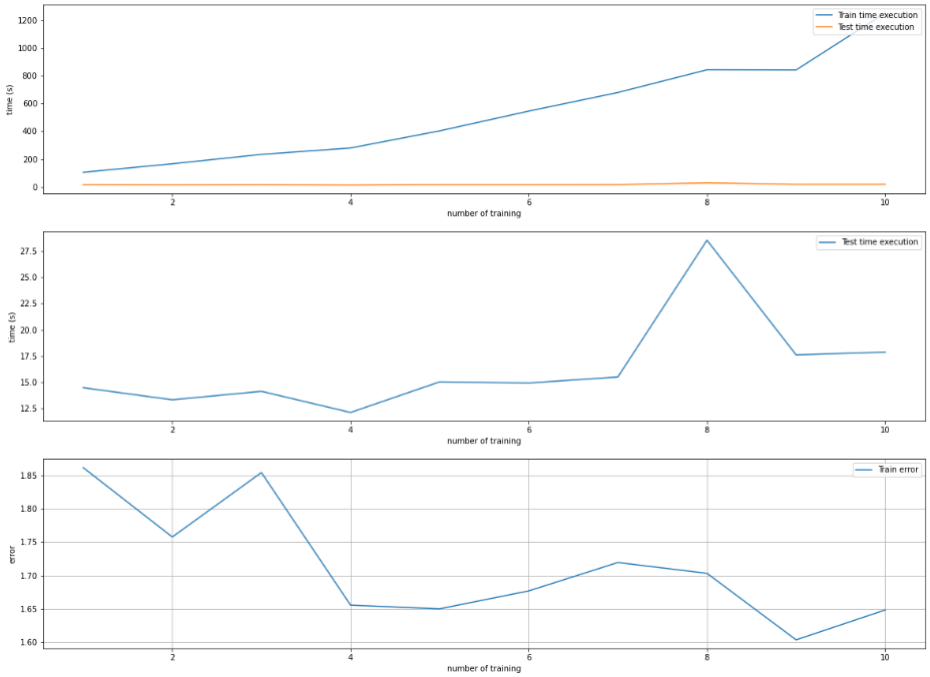


FIGURE 7 – Tests du nombre d'entraînement pour le CNN

Nous observons sur la 3e courbe de la Figure 4 que 40 neurones pour la deuxième couche donne une erreur faible sans sur-apprentissage, car après 40 neurones, nous pouvons voir une augmentation de l'erreur.

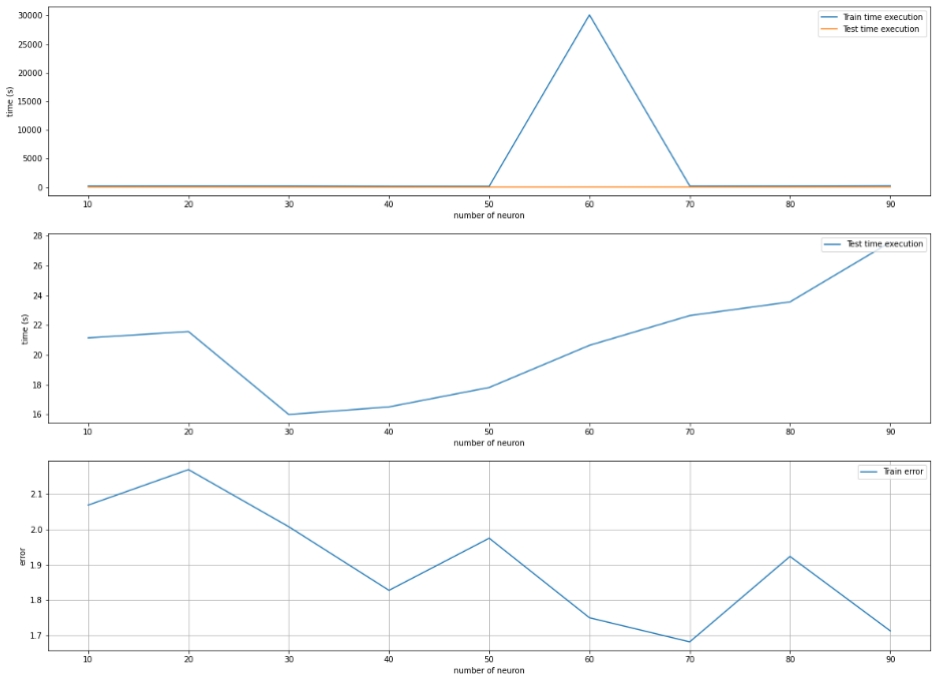


FIGURE 8 – Tests du nombre de neurones de la deuxième couche cachée pour le CNN

De même, nous observons sur la 3e courbe de la Figure 4 que 40 neurones pour la première couche donne une erreur faible sans sur-apprentissage, car après 40 neurones, nous pouvons voir une augmentation de l'erreur.

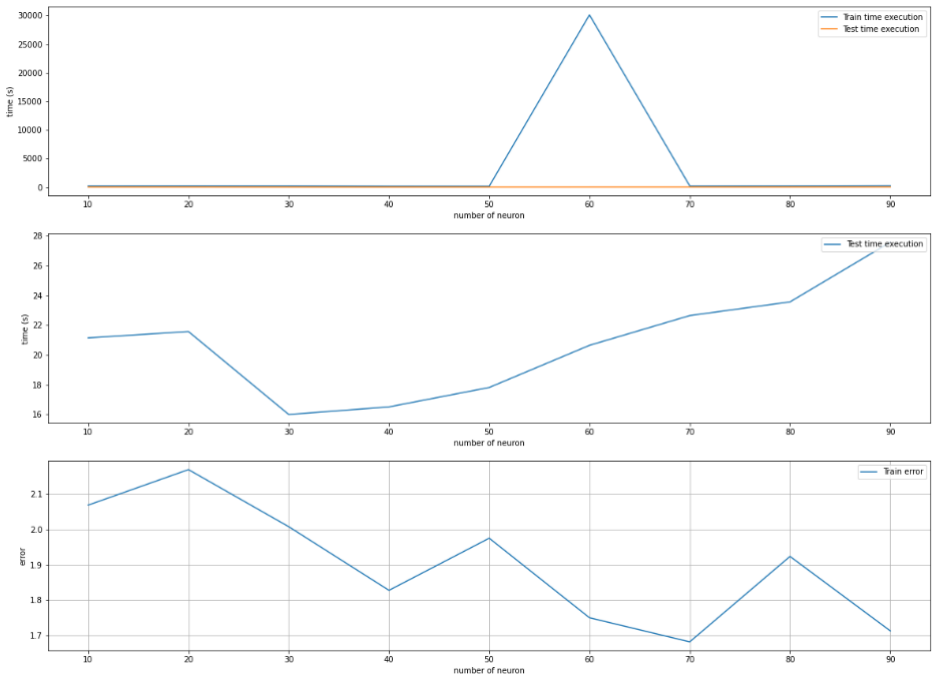


FIGURE 9 – Tests du nombre de neurones de la première couche cachée pour le CNN

Voici les résultats que nous avons obtenus :

	Temps pour trouver les paramètres (1)	Temps de test (2)	Taux de réussite (3)
MV	0.7750675678253174	76.360302686691	0.8417
RNPR	354.89	67.856763362884	0.8785
CNN	240.69	17.06	0.9196

(1) : Temps en secondes nécessaire pour trouver les paramètres optimaux pendant la phase d'entraînement. RNPR a été calculé pour 40 de neurones par couches cachées, CNN est calculé avec 40 neurones par couches entièrement connectées.

(2) : Le temps de test est calculé pour 1500 images.

(3) : La probabilité de réussir à prédire la bonne classe.

3 Méthode

Pour répondre à l'objectif, nous avons tout d'abord créé l'environnement que nous utiliserons pour faire les tests d'apprentissage. Puis nous avons cherché à classifier la base de données MNIST, nous avons créé pour cela plusieurs classifieurs, d'abord à maximum de vraisemblance, puis à réseau de neurones via la bibliothèque pytorch qui permet de faciliter la création de réseaux de neurones. Nous devons définir une fonction de récompense. Pour définir la fonction de récompense, nous avons observé le comportement du classifieur lorsque les entrées sont des images de la caméra. Enfin, nous avons implémenté l'algorithme de renforcement deep-Q pour guider le mouvement de la caméra.

4 Environnement

Pour construire notre environnement gym, nous nous sommes inspirés des exemples fournis par Bullet. Nous avons créé nos propres fichiers URDF, OBJ, MTL. URDF signifie Unified Robot Description Format, et est un format pour décrire l'état d'une structure. Dans notre cas, nous l'utilisons pour fixer l'état des cubes de l'environnement. Il faut un fichier par cube. Pour créer nos propres fichiers URDF, nous avons suivi le tutoriel [8]. ROS (Robot Operating System) est un framework de logiciels pour robots. Il s'agit d'une collection d'outils, de bibliothèques visant à simplifier la création de comportements robotiques complexes et robustes sur une grande variété de plateformes robotiques.

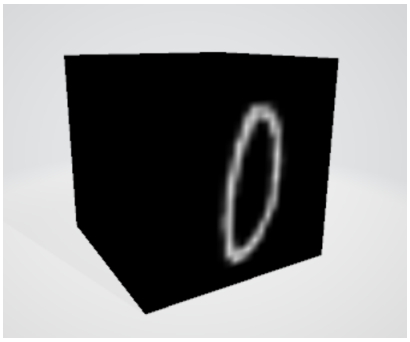


FIGURE 10

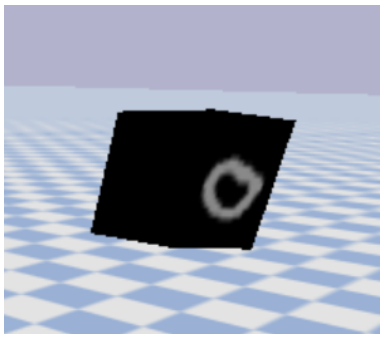


FIGURE 11

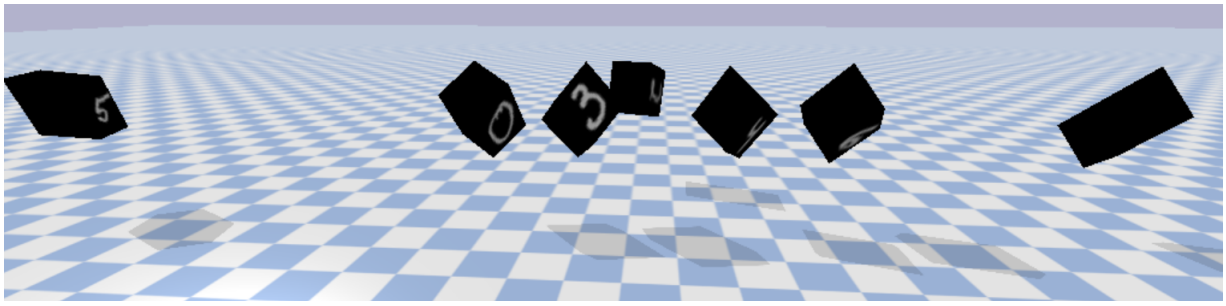


FIGURE 12

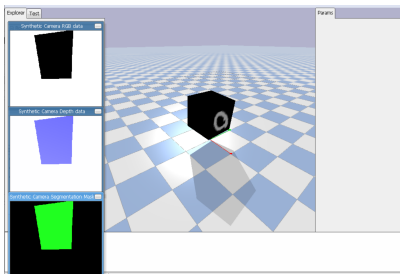


FIGURE 13 – Exemple de l'environnement avec la caméra

5 Fonctions de récompenses

Après avoir créé le classifieur, nous avons cherché à mettre en place une fonction de récompense qui permet de guider l'apprentissage.

5.1 Simple

Pour cela, notre première idée était une fonction de la forme :

$$r(S_t, A_t) = \begin{cases} 1 & \text{si il y a un chiffre détécté} \\ 0 & \text{sinon} \end{cases}$$

Cependant, la récompense étant rare, cela n'aidait pas beaucoup l'agent, et une fonction donnant des récompenses plus progressives mais plus souvent était nécessaire.
Une seconde idée à été une fonction de la forme :

$$r(S_t, A_t) = \max_{i \in [0,9]} y_i$$

Pour où la récompense est la meilleure probabilité de classification. Même si très simple, cette fonction donne des résultats satisfaisants. Nous avons cependant continué a chercher d'autres options de fonctions de récompense.

5.2 Un peu plus complexe

Une seconde idée était une fonction de la forme :

$$r(S_t, A_t) = \begin{cases} \frac{1}{2} \sum_{i=0}^{9} y[i] & \text{si } \sum_{i=0}^{9} y[i] > y[11] \\ 0 & \text{sinon} \end{cases}$$

Avec le vecteur de taille 11 y, la solution du classifieur. Cependant, cela produit aussi des erreurs de classifications.

5.3 Observation à un pas de temps donnée

Une idée a été de déterminer la récompense sur un ensemble d'images plutôt que sur une seule. Ainsi, nous observons le comportement du réseau sur les images de l'environnement en modifiant la pose de la caméra.

6 Algorithme de renforcement pour la perception active

La seconde partie du projet est dédiée à l'élaboration de l'algorithme d'apprentissage par renforcement.

Le but de l'apprentissage est de permettre à l'agent de choisir la meilleure action possible selon l'état du système. L'apprentissage par renforcement est un type d'apprentissage dans lequel l'amélioration de l'agent se fait par tentatives successives, en observant l'efficacité de ses actions. Ce type d'apprentissage à pour principal avantage de ne pas requérir de connaissance préalable sur le système avant sa mise en route. Le but de l'apprentissage par renforcement est donc pour l'agent d'associer à un état du système l'action à effectuer qui lui rapportera la meilleure récompense. Un des moyens d'y parvenir est via l'apprentissage de la fonction Q (Q-learning en anglais).

Cette méthode vise à construire une fonction Q prenant un état du système en entrée et renvoyant la meilleure action à effectuer. La construction de la fonction Q se fait par approximation, en utilisant la méthode de descente de gradient qui permet d'estimer une fonction à partir de connaissances sur ses entrées et sorties. La descente de gradient doit donc se faire après coup, à partir de l'état du système, l'action choisie et la récompense obtenue.

Nous avons défini un état du système comme étant l'ensemble des images vues au cours de l'exécution ainsi que l'ensemble des actions ayant mené à l'état actuel. Le nombre d'états possibles étant alors infini, il nous est impossible de gérer les états par des moyens classiques de stockage (liste,...). Nos professeurs nous ont alors redirigés vers l'algorithme dit de "deep Q-learning", nous fournissant un papier présentant cet algorithme. [10]

6.1 Détails de l'intérêt de l'algorithme deep-Q

Dans les cas simples, nous représentons la fonction Q par une liste où $Q[i]$ représente la meilleure action à effectuer si le système est à l'état i. Cela n'est possible que si le nombre d'états est limité, du à la limite du stockage en liste.

L'algorithme deep-Q permet de contourner cette limite et permet de créer une fonction Q pour un nombre potentiellement infini d'états possibles.

Cela est permis par l'utilisation de réseaux de neurones. Au lieu d'approximer la fonction Q "à la main", celle ci sera "contenue" dans le réseau de neurones. La différence avec le cas classique est que au lieu de renvoyer l'action maximisant la récompense, le réseau de neurones renvoie autant de valeurs que d'actions possibles, et la sortie i correspond à la récompense espérée si l'action i est choisie. Cela permet de réduire la taille des instances manipulées du nombre d'états possibles au nombres d'actions possibles, généralement très faible. Dans notre cas, si le nombre d'états possibles est infini, seulement 12 actions sont possibles, ce qui permet de ne manipuler que 12 valeurs à chaque itération.

6.2 Algorithme pas à pas

6.2.1 Initialisation

L'algorithme réalise un certain nombre d'épisodes, la caméra étant déplacée à un endroit aléatoire à la fin de chaque épisode. Cela permet à l'algorithme de s'entraîner sur différents états initiaux, afin d'augmenter ses performances. Un épisode se termine si l'agent à rempli sa mission (identifier avec succès un cube) ou si le nombre d'itérations maximal est écoulé.

L'algorithme garde en mémoire 2 variables, les séquences et l'historique.

La séquence d'un épisode correspond à l'ensemble des images vues et des actions choisies depuis le début de l'épisode. Ainsi, $sequence.get(i)$ renvoie la liste des images de l'itération 0 à l'itération i et la liste des actions de l'itération 0 à l'itération $i-1$. Selon notre définition d'un état du système, $sequence.get(i)$ renvoie donc l'état du système à l'itération i. Cette modification par rapport à l'algorithme original permet de n'avoir qu'un seul stockage des images et des actions par épisode, là où l'algorithme original créait une nouvelle séquence à chaque itération, pour un gain considérable en complexité mémoire.

L'historique est une liste gardant en mémoire toutes les transitions rencontrées. Une transition selon l'algorithme original est l'ensemble : pour la classification, il faudrait utilisé Un modèle génératif permet d'estimer la confiance d'une prédiction et donc de formuler un rejet d'une prédiction et pour une exploration plus avancée nous pourrions utiliser un stratégie evolutionnaire pour générer des trajectoires. La réduction réalisée sur les séquences permet de retrouver une séquence à partir de l'itération et l'épisode, permettant une réduction de la taille mémoire d'une transition. Cela nous a permis de considérer une transition comme étant l'ensemble :

$$\begin{cases} start & : t \\ action & : \text{action choisie au temps t} \\ reward & : \text{récompense obtenue au temps t} \\ end & : t+1 \\ episode & : \text{ numéro de l'épisode} \\ done & : \text{indique si l'état t+1 est un état final} \end{cases}$$

Les listes de stockage de l'historique et des séquences est initialisée au lancement de l'algorithme, et une nouvelle séquence est créée au début de chaque épisode.

6.2.2 Boucle

La boucle des actions répétées à chaque itération débute par le choix de l'action à effectuer. Ce choix se fait via la stratégie d'exploration dite "ε-greedy" : un nombre aléatoire est tiré, si ce nombre est supérieur à une valeur ε fixée, généralement très faible, alors l'agent suit la politique. Ici, cela signifie que l'action choisie correspond au rang de la plus grande valeur retournée par le réseau de neurones, en lui donnant la séquence actuelle en argument. Si le nombre est inférieur à la valeur ε, alors l'action est choisie aléatoirement parmi les actions possibles. Cette méthode permet d'assurer l'efficacité de l'algorithme, puisque l'agent suivra dans la majorité du temps le choix lui donnant la meilleure récompense, tout en permettant d'explorer de nouvelles possibilités avec la possibilité d'agir aléatoirement.

L'action suivante est l'action réelle : l'algorithme exécute un "pas" de l'environnement, déplaçant la caméra selon l'action choisie et récupérant plusieurs informations

utiles, dont la récompense de l'action effectuée, et le booléen *done* indiquant si l'objectif de l'agent a été rempli.

L'algorithme réalise ensuite les mises à jour de la séquence et de l'historique. L'image de la caméra est récupérée, et le couple (image, action) est ajouté à la séquence actuelle. La transition comprenant l'action choisie au cours de l'itération en cours et la récompense obtenue est ajoutée à l'historique. Une transition est choisie aléatoirement dans l'historique, et sert de test de performance du réseau via la descente de gradient.

La dernière partie de la boucle sert à améliorer le réseau de neurones. La méthode décrite dans l'algorithme original, et qui est celle retenue ici, est la méthode de la descente de gradient stochastique. Une transition est choisie aléatoirement dans l'historique, et nous servira de test de performance du réseau.

La fonction objectif y_i de la fonction Q est alors calculée comme suit :

$$y_i = \begin{cases} r_i & \text{si transition mène à un état final} \\ r_i + \alpha * \max\limits_a Q(\phi_{i+1}, a, \theta) & \text{sinon} \end{cases}$$

où r_i est la récompense à l'itération i, α est le facteur de réduction des récompenses, a est une action possible de l'environnement, ϕ_{i+1} représente la séquence à l'itération $i+1$ et θ paramètres du réseaux.

Nous calculons ensuite $Q(\phi_i, a, \theta)$, et l'erreur entre ces deux valeurs est calculée via la méthode des moindres carrés. Cette erreur est enfin remontée au réseau de neurones par rétropropagation.

7 Résultats

Initialement, nous ne sommes pas parvenus a faire fonctionner l'algorithme à cause d'un problème dans la base de classification (voir Discussions). Les résultats suivants sont réalisés avec une base de classification améliorée.

Après modification de la base d'apprentissage, nous obtenons de bons résultats sur la base des chiffres sur la 11e classe.

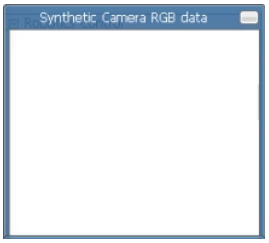


FIGURE 14 – Caméra filmant le ciel de l'environnement

Sortie du réseau de neurones :

Classe	valeur de sortie	Classe	valeur de sortie	Classe	valeur de sortie
0	0.01380341	4	0.0061454	8	0.0061454
1	0.0061454	5	0.0061454	9	0.0061454
2	0.0061454	6	0.00691775	10	0.9301156
3	0.0061454	7	0.0061454		

Nous observons une classification de 0.93 pour la classe 11, signe que l'identification d'images sans cube est réussit.

Un exemple de détection de chiffre, mais d'erreur sur l'identification, signe que la base d'apprentissage n'est pas assez grande

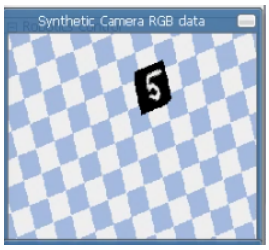


FIGURE 15 – Caméra filmant un cube avec le chiffre 5 lisible

Sortie du réseau de neurones :

Classe	valeur de sortie	Classe	valeur de sortie	Classe	valeur de sortie
0	0.005140059	4	0.005140059	8	0.05140059
1	0.007343848	5	0.005140059	9	0.05140059
2	0.44752684	6	0.006265604	autre	0.05140059
3	0.005140059	7	0.005657447		

Nous voyons sur cet exemple que si le classifieur se trompe sur le chiffre (il croit lire un 2 alors qu'un 5 est sur l'image), il est certain qu'un chiffre est sur l'image et élimine toutes les autres possibilités. Cela montre qu'une bonne performance du classifieur peut être atteinte si la base d'apprentissage était enrichie de davantage de cubes avec chiffres visibles.

Nous ne sommes cependant pas parvenus à faire fonctionner l'algorithme deep-Q. La cause a été le caractère extrêmement rare de la capture d'un cube par la caméra.

8 Discussions

Le premier problème lors de l'exécution a été le classifieur, plus précisément sa base de données d'apprentissage. En effet, l'apprentissage sur les bases MNIST ne permettent pas un apprentissage efficace. La conséquence a été que le classifieur était incapable de déterminer la classe des images lorsque le chiffre ne prenait pas toutes la largeur et la longueur de l'image capturée par la caméra. De plus, le classifieur se trompe souvent pour des images présentant un cube noir sans chiffre.

Une tentative de résolution de ce problème a été d'intégrer des images de l'environnement à la base d'apprentissage.

Le résultat a été meilleur : l'ajout de plusieurs milliers d'images de l'environnement sans cubes visibles a permis d'améliorer la classification de la 11e classe lors des captures d'images de la caméra dans l'environnement. L'ajout de 1200 images de cubes avec chiffre visible par classe a été efficace sur l'image avec chiffre lisible.

Le second problème, plus complexe que nous ne sommes pas parvenus à résoudre, est sur la taille de l'environnement et les angles de caméra. Sur les centaines de tests que nous avons effectués, les cubes n'ont quasiment jamais été visibles à la caméra, et étaient mal orientés ou trop distants pour être identifiés dans les rares cas où un cube apparaissait sur la caméra. La conséquence est que l'algorithme d'apprentissage n'a jamais rencontré de récompense élevée lorsqu'il ne trouve jamais d'images contenant un cube avec un chiffre, et n'a donc jamais pu apprendre efficacement.

9 Conclusion

En conclusion, malgré des tentatives de correction de la base d'apprentissage, nous ne sommes pas parvenu à faire fonctionner efficacement l'algorithme deep-Q. Si des mouvements non aléatoires de la caméra nous indiquent que l'algorithme tente de se diriger vers des séquences à récompense plus élevée, l'absence de détection d'image contenant un chiffre et donc de récompense élevée empêche l'algorithme de fonctionner correctement.

Le fait que la caméra ne soit pas contrainte sur ses mouvements et ses angles de prises de vues conduit le cube à n'être quasiment jamais capturé par la caméra, limitant les possibilités d'apprentissage et menant à une situation où la caméra erre sans but quasi aléatoirement puisqu'elle n'a rencontré que des récompenses faibles au cours des itérations et épisodes.

Il serait également nécessaire d'augmenter encore plus la taille de la base d'apprentissage, avec plus d'images des différentes classes. Cela n'a pas été fait par manque de temps.

Pour la classification, une possibilité d'amélioration serait d'utiliser un modèle génératif permettant d'estimer la confiance d'une prédiction et donc de formuler un rejet d'une prédiction du classifieur. [20]

Pour une exploration plus avancée nous pourrions utiliser un stratégie évolutionniste pour générer des trajectoires.

10 Annexe

10.1 Cahier des charges

Objectif

L'objectif du projet est de développer un environnement Gym basé sur le moteur physique Pybullet pour l'évaluation de politiques de perception active d'un robot capable de reconnaître le chiffre sur chaque cube présent dans l'environnement.

Ainsi, il faudra :

— Créer l'environnement, les cubes avec la texture de digit sur l'une des face de chacun des cubes.
— Initialiser l'emplacement de la caméra et les cubes dans l'environnement.
— Classifier les chiffres sur les images de la base de donnée MNIST.
— Définir les fonctions récompenses qui guidera le déplacement de la caméra.
— Implémenter un algorithme d'apprentissage par renforcement.

Modélisation

Objets d'intérêt

Les objets d'intérêt de notre environnement sont N cubes numérotés par un chiffre de 1 à 9 sur une des 6 faces du cube. Les chiffres seront issus de la base de données MNIST. MNIST signifie -Modified National Institute of Standards and Technology- contient une série de chiffres manuscrites pour les tests et les entraînement Cette base de donnée servira pour la classification des chiffres.

Environnement $3D$

L'environnement 3D simulé servira à l'apprentissage et à l'évaluation des politiques pour répondre l'objectif. Il est de largeur W, de longueur L et de hauteur H. Il est constitué d'une caméra $3D$ et de N cubes.

Ainsi, l'espace des états de notre environnement est composé de point défini par un vecteur de taille 3 : $[x, y, z]$, $x < W, y < L, z < H$.

Agent

L'agent qui est soumis à l'apprentissage est une caméra 3D pouvant explorer l'environnement et ayant pour objectif de détecter le numéro sur une des faces du cube dans son environnement.

Actions

L'ensemble des actions possible pour l'agent comprend les changements de pose de la caméra et les rotations autour de l'axe fixe des cubes :

— • 6 pour le centre de projection de la caméra : déplacement de la position selon les axes x, y, z dans l'environnement
— • 6 pour la position de la caméra : yaw, pitch, roll

État initial de l'environnement

Il s'agit de définir l'état initial de l'environnement à chaque simulation. Pour cela, nous définirons l'état initial de tous les composants de l'environnement.

— • Position initiale des cubes fixés, ils sont fixés dans leur référentiel et subissent la force gravitationnelle
— • Orientation d'un cube aléatoire
— • Position et orientation initiale de la caméra définis aléatoirement
— • Taille de l'environnement fixé à $L \times W \times H$

Apprentissage

Espace des états

L'ensemble des états sont (x,y,z) coordonnées dans l'espace de l'environnement, $x \in [\![2, \mathrm{W}]\!]$, $y \in [\![2, \mathrm{L}]\!]$, $z \in [\![2, \mathrm{H}]\!]$.

Espace des actions

A chaque état, l'agent a 12 actions possibles :

$$\{x+1, x-1, y+1, y-1, z+1, z-1, yaw+1, yaw-1, pitch+1, pitch-1, roll+1, roll-1\}$$

$\{x, y, z\}$ correspondent à la positon de projection de la caméra. Et $\{yaw, pitch, roll\}$ correspondent aux rotations possibles de la caméra.

Après avoir créé l'environnement qui servira pour l'apprentissage, nous devrons définir un classifieur sur les images de la base de donnée MNIST. Puis nous devrons définir les fonctions de récompenses qui aideront à l'apprentissage. En effet, les algorithmes d'apprentissage par renforcement permettent de trouver la politique optimale qui maximise une récompense.

Dans notre cas, la valeur de la récompense augmente lorsque l'agent détecte un chiffre. Nous utiliserons un modèle d'apprentissage profond qui réussit à apprendre des politiques de contrôle directement à partir d'une entrée sensorielle en utilisant l'apprentissage par renforcement. Le modèle est un réseau de neurones convolutifs, formé avec une variante de l'apprentissage Q, dont l'entrée est constituée d'images et dont la sortie est une fonction de valeur estimant les récompenses futures. L'algorithme de renforcement est appelé le "deep-Q" [10].

Références

[1] R. S. Sutton and A. G. Barto, Reinforcement Learning An Introduction. Cambridge, Mass : MIT Press, 1998. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[2] Chili, Margarita, and Andrew J. Davison "Active matching" European conference on computer vision. Springer,Berlin, Heidelberg, 2008.

[3] R. Cheng, A. Agarwal, et K. Fragkiadaki, « Reinforcement Learning of Active Vision for Manipulating Objects under Occlusions », ArXiv181108067 Cs, févr. 2019. [En ligne]. Disponible sur : http://arxiv.org/abs/1811.08067.

[4] OpenAI, « Gym : A toolkit for developing and comparing reinforcement learning algorithms ». https://gym.openai.com.

[5] « CS231n Convolutional Neural Networks for Visual Recognition ». https://cs231n.github.io/.

[6] « I2DL ». https://niessner.github.io/I2DL/.

[7] « Bullet Real-Time Physics Simulation | Home of Bullet and PyBullet : physics simulation for games, visual effects, robotics and reinforcement learning. » https://pybullet.org/wordpress/.

[8] « urdf/Tutorials -Robot Operating System Wiki ». http://wiki.ros.org/urdf/Tutorials/.

[9] « CIFAR-10 and CIFAR-100 datasets ».http://www.cs.toronto.edu/~kriz/cifar.html.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller, *Playing Atari with Deep Reinforcement Learning*, https://arxiv.org/pdf/1312.5602.pdf

[11] C. Premebida, R. Ambrus, et Z.-C. Marton, « Intelligent Robotic Perception Systems », Appl. Mob. Robots, nov. 2018, doi : 10.5772/intechopen.79742.

[12] D. Djian, "Contribution à l'analyse de scènes par vision active : utilisation de réseaux Bayesiens," Thèse de doctorat, Institut national de recherche en informatique et en automatique, centre de documentation, Sophia Antipolis, Alpes-Maritimes, France, 1997.

[13] Coralie Bernay-Angeletti. Stratégie de perception active pour l'interprétation de scènes. Autre. Université Blaise Pascal - Clermont-Ferrand II, 2016. Français. ffNNT : 2016CLF22710ff. fftel-01420115 [En ligne]. Disponible sur : https://tel.archives-ouvertes.fr/tel-01420115/document

[14] D. Grover et B. Toghi, « MNIST Dataset Classification Utilizing k-NN Classifier with Modified Sliding-Window Metric », in Advances in Computer Vision, Cham, 2020, p. 583-591.

[15] R. Bajcsy, Y. Aloimonos, et J. K. Tsotsos, « Revisiting active perception », Autonomous Robots, vol. 42, no 2, p. 177-196, févr. 2018, doi : 10.1007/s10514-017-9615-3.

[16] Z. Jie, X. Liang, J. Feng, X. Jin, W. F. Lu, et S. Yan, « Tree-Structured Reinforcement Learning for Sequential Object Localization », arXiv :1703.02710 [cs], mars 2017, Disponible sur :http://arxiv.org/abs/1703.02710.

[17] R. Bajcsy, « Active perception », Proceedings of the IEEE, vol. 76, no 8, p. 966-1005, août 1988, doi : 10.1109/5.5968.

[18] R. Cheng, A. Agarwal, et K. Fragkiadaki, « Reinforcement Learning of Active Vision for Manipulating Objects under Occlusions », ArXiv181108067 Cs, févr. 2019. [En ligne]. Disponible sur : http://arxiv.org/abs/1811.08067.

[19] Philip Thomas et Emma Brunskill, « Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning », in Proceedings of The 33rd International Conference on Machine Learning, juin 2016, p. 2139-2148, [En ligne]. Disponible sur : http://proceedings.mlr.press/v48/thomasa16.html.

[20] A. Kendall et Y. Gal, « What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision ? », ArXiv170304977 Cs, oct. 2017, Consulté le : avr. 27, 2021. [En ligne]. Disponible sur : http ://arxiv.org/abs/1703.04977