2016
-
2017

Compte-Rendu
Projet de 1ère année

PROJET NEURON

BENJAMIN ROZIERE & VALENTIN FERERRE

ISIMA | 2016-2017

I. Objectif du projet

Le but de ce projet est de réaliser un réseau de neurone en mesure de jouer à un jeu de plateforme.

Partant de cet objectif, nous avons pu découper le projet en plusieurs étapes :

- Le jeu de plateforme
    - Le jeu à proprement dit
    - La récupération des données du jeu pour constituer les données de test
    - La connexion avec le réseau de neurones afin de le faire jouer au jeu
- Le réseau de neurones
    - La récupération des données du jeu
    - La normalisation des données
    - La création de jeu d'entrainement et d'essai
    - Le réseau en lui-même
    - L'entrainement avec retour sur sa précision
    - La prédiction de données pour le jeu

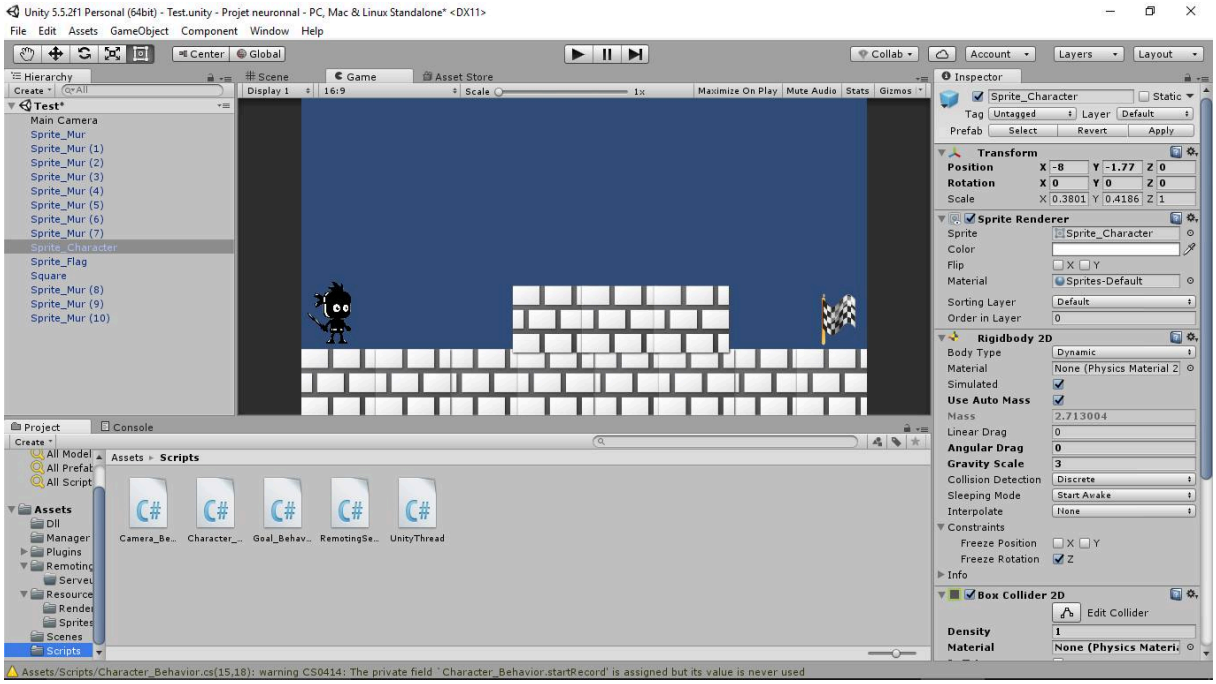
II. Le jeu

La première étape a été de définir sur quel genre de jeux le réseau allait devoir jouer. Afin de garder les choses simples, nous avons choisis de réaliser un jeu de plateforme affiché en 2 dimensions.

Nous avons défini 4 éléments principaux du jeu :

- Le joueur représenté par son avatar
- L'environnement
- Les obstacles sous la forme de carrés gris
- L'objectif sous la forme d'un drapeau

Le jeu n'étant pas la préoccupation centrale du projet, nous avons décidé de le réaliser via Unity (en C#). Unity est une plateforme de développement de jeux vidéo, elle permet de réalise aussi bien des jeux en 2D qu'en 3D très rapidement. Nous avons donc réalisé un jeu très simple, pouvant se jouer à l'aide des flèches du clavier.



Le jeu réalisé, nous nous sommes interrogés sur comment abstraire le fait de jouer pour que le réseau de neurone puisse répondre avec des données utilisables en jeu. La solution la plus simple semblait alors de lui donner une prise d'écran du jeu en entré, et de lui demander de prédire quelle action il devait effectuer parmi les suivantes :

Nom	Label
Aller à gauche	0
Aller à droite	1
Sauter	2
Sauter et aller à gauche	3
Sauter et aller à droite	4
Ne rien faire	5

Ce qui peut s'écrire sous la forme d'un vecteur de taille 6 avec la composante correspondant à l'action à 1 et le reste à 0 (ex. $y = [0, 0, 1, 0, 0, 0]$ correspond à sauter.

Afin d'entrainer le réseau de neurones, il nous a fallu constituer un jeu de donné assez important. De plus, ce jeu de donné devait comporter deux éléments : Une capture d'écran du jeu et l'action à effectuer.

III. Le prétraitement

Afin de réaliser ce jeu plus facilement et rapidement, nous avons ajouté un script dans Unity qui observe les évènements du clavier, dès qu'une touche parmi celle décrites plus haut est appuyée, le script réaliser une prise d'écran du jeu et conserve la valeur entre 0 et 5 de l'action réalisée.

Ce script génère un fichier contenant tous les images du set d'entrainement et de test, ce fichier sera ensuite mélanger par un script python pour avoir une répartition plus aléatoire au sein du set de données. Les images générées sont au format 638x359 soit environ 0.7Mo par images, elles sont donc trop volumineuses pour les capacités de notre machine. Avant de confier les images au réseau de neurone un script python va :

- Convertir les images du format RGB au format niveau de gris ce qui diminue la taille de l'image par 3
- Compresser la taille de l'image par 10 avec pour résultat une image de taille 64x36

Grâce à ces deux actions, la taille de l'image a été réduite par 30. Cette optimisation est nécessaire pour obtenir des résultats dans un délai raisonnable.

Une fois le « prétraitement » effectué nous pouvons maintenant nous intéresser au réseau de neurone.

IV. Le réseau de neurone

1. Premier réseau (Bernard I)

Pour résoudre le problème de trouver une action en fonction de l'image source, nous avons tout d'abord tenté d'y répondre de manière simple via une classification. Cela consiste pour le réseau à « ranger » l'image source dans une catégorie d'action à effectuer.

Nous avons développés une première version du réseau de neurone en utilisant la méthode de « Softmax regression ».

Cette méthode vise que pour un vecteur d'entré x, le réseau produira une distribution de probabilité parmi K classes de sorties. Ici nous avons $K = 6$ donc le vecteur de sortie sera de taille 6, et le vecteur d'entrée sera de la taille $64 \times 36 = 2304$. En effet, cette méthode oblige à utiliser un vecteur en entré, ce qui perd malheureusement l'information 2D de l'image.

Pour implémenter ce réseau, nous avons choisis l'outils TensorFlow, il permettait à la fois d'être simple à mettre en oeuvre sur nos machines (python) mais il dispose aussi d'une bonne documentation et communauté. De plus il permet à la fois de comprendre plus en profondeur le fonctionnement des réseaux mais aussi de travailler avec des méthodes de plus haut niveau.

Cette méthode revient à attribuer un poids à chaque pixels de l'image, un poids négatifs correspondant à une preuve que l'image n'appartient pas à une classe, et un poids positif étant une preuve en faveur de l'appartenance de l'image à une classe.

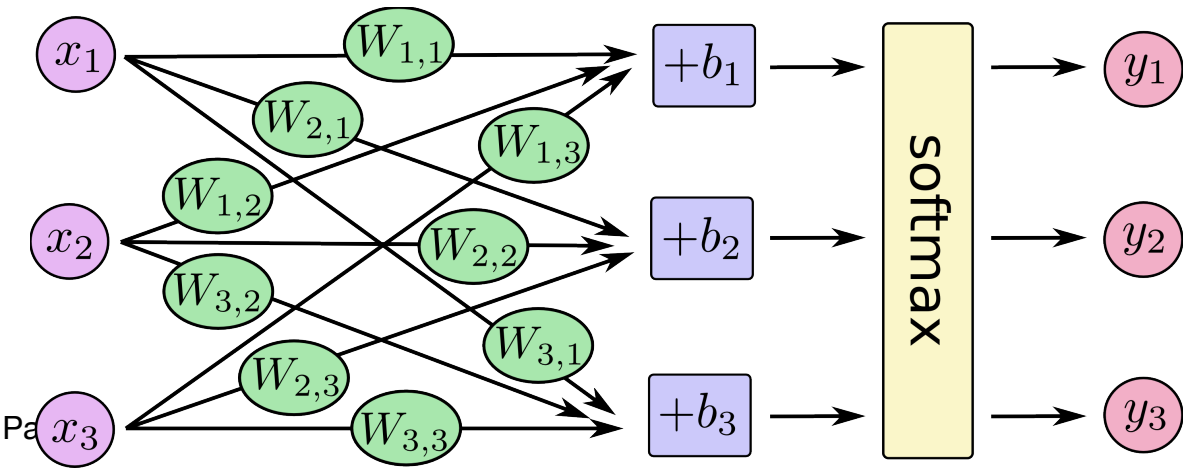
La formule du total de « preuve » étant:

$$preuve_i = \sum_j W_{i,j} x_j + b_i$$

Il nous faut ensuite distribuer cette somme de preuve parmi les K classes via la fonction $softmax$.

$$y = softmax(preuve)$$

Sous la forme d'un réseau de neurones, on obtient donc un réseau de la forme suivante :

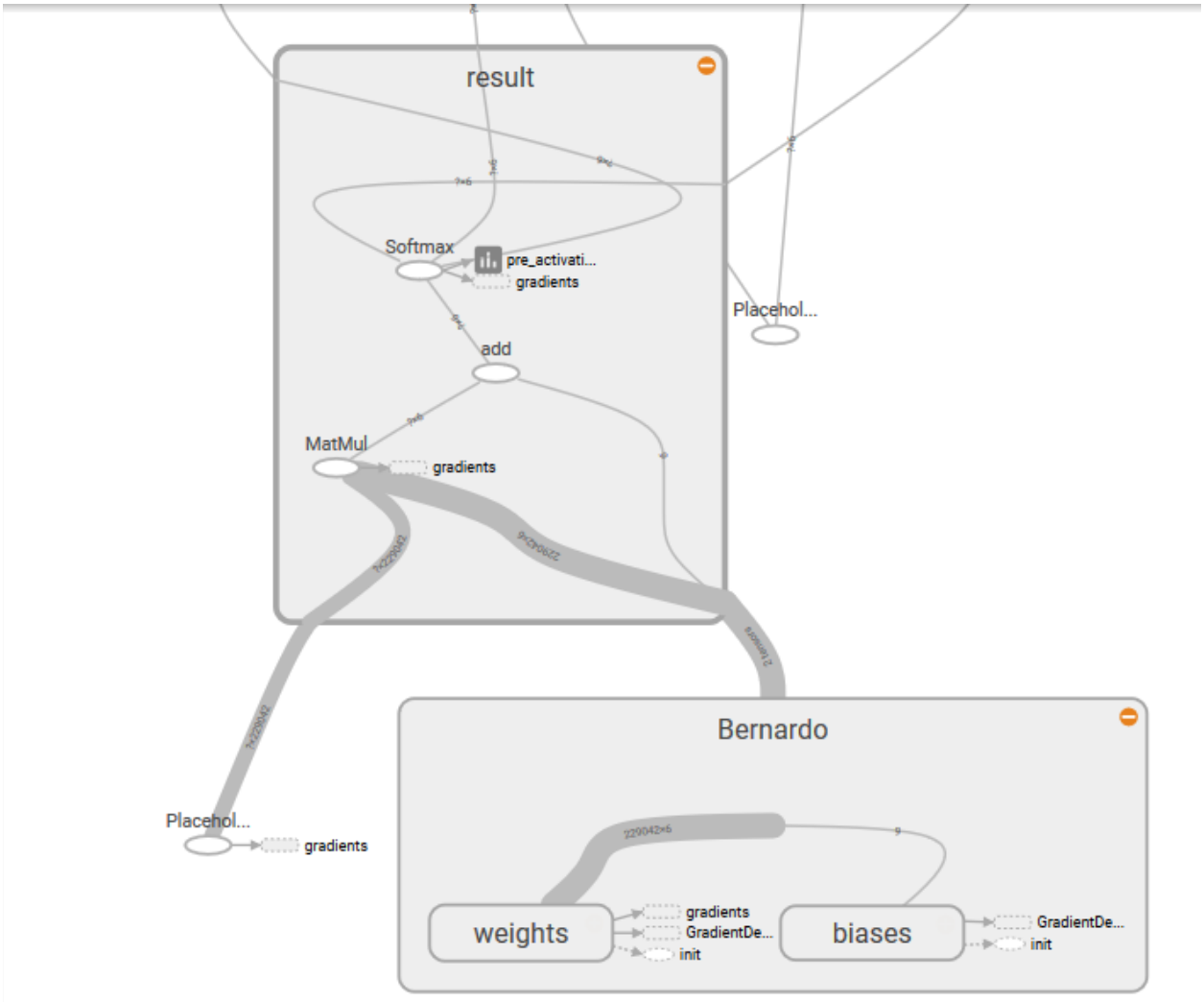


On peut donc généraliser le réseau par la formule matricielle :

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Qui s'écrit donc $y = softmax(Wx + b)$

Nous avons donc implémentés cette formule au sein de TensorFLow sous la forme d'un graphe d'opérations.



ENTRAINEMENT
Pour entrainer ce réseau, nous avons utilisé notre set de données d'entrainement.

Le calcul de l'erreur du réseau est réalisé grâce à l'entropie, soit la formule :

$$H_{y'}(y) = -\sum_i y'_i log(y_i)$$

Où y est le vecteur prédit, et y' le vecteur que l'on devait obtenir.

Pour chaque prédiction durant l'entrainement du réseau, nous calculons l'entropie du résultat, puis nous utilisons la descente de gradient pour minimiser cette entropie, à l'aide d'un algorithme de rétro-propagation pour modifier les poids W et les biais b du réseau.

LIMITATIONS
Une fois le réseau configuré, nous avons pu l'entrainer de nombreuses fois, tout en essayant de maximiser sa précision. Nous avons essayé différentes configurations, à la fois en changeant le nombre d'entrainement, mais aussi le nombre d'images à chaque entrainement. Néanmoins, nous avons constaté qu'une tendance finissait par se dégager de l'entrainement du réseau.

En effet, si le jeu de donné ne constituait qu'un seul niveau ou seul le joueur se déplaçait, le réseau pouvait produire des résultats jusqu'à **80%** de précision. Mais dès que l'objectif ou les obstacles changeaient de place, le réseau soit tombait à 0% de précision, soit il apprenait une seule action par coeur.

Nous en avons conclus que si notre réseau pouvait classifier les images d'un même niveau, le fait d'attribuer un poids à chaque pixel le rendait inefficace dans le cas ou ces zones n'étaient plus utilisées par l'objectif ou par des obstacles. Il ne pouvait donc pas généraliser son apprentissage sur différents niveaux du jeu.

2. Réseau multi-couches (Bernard II)

Afin d'augmenter la précision du réseau, nous avons décidé de changer le type de réseau utilisé. L'image devant être décomposée afin d'identifier les différents éléments avant de prédire l'action à entreprendre, nous avons choisis d'utiliser un réseau de neurones convolutifs.

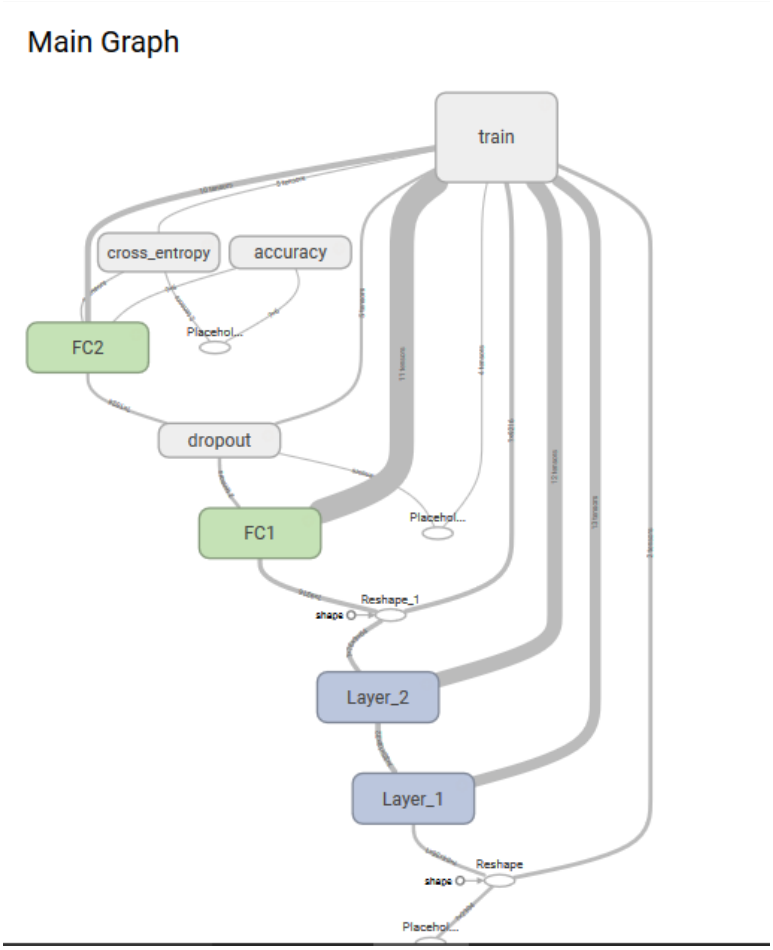
Un réseau convolutif (CNN) est composé de plusieurs réseaux de convolution et de plusieurs réseau densément connectés. Cela permet de décomposer l'image pour en extraire les différentes composantes (formes, lignes) et analyser ces composantes dans son ensemble (conservation des informations 2D) contrairement au premier réseau.

Dans la pratique, un réseau de convolution va découper l'image en petites parties (patch) afin d'en extraire l'information. L'information extraite est ensuite redimensionnée via l'opération de pool. Les niveaux densément connectés sont quant à eux identiques à celui présent dans le premier réseau et permettent

d'appliquer la méthode softmax aux informations extraites de l'image pour obtenir la distribution finales parmi les classes K.

LIMITATIONS

Nous avons implémentés ce nouveau réseau avec Tensorflow, malheureusement nous nous sommes heurtés à une première limitation technique. La multiplication des couches du réseau, et la taille des patch ainsi que le nombre d'informations par couches à extraire augmente considérablement le nombre de neurones et donc le nombre d'opérations. Sur une configuration avec 16Go de mémoire et les calculs effectués par le CPU, la durée de l'entrainement se comptait en heures et les calculs devenaient impossibles par manque de mémoire.



Nous avons du réduire le nombre de couches de convolution à 2 (Layer 1 et 2) et 2 couches dense (FC1 et 2). Ces deux réseaux utilisaient des patchs de 5 par 5 et extrayaient 32 et 64 informations respectivement, impossible de travailler sur plus avec notre matériel.

ENTRAINEMENT

Nous avons donc utilisé le même set de données pour entrainer et tester notre réseau. Nous avons pus constater que après plusieurs essais, nous atteignons une précision d'environ **35%** avec le jeu d'essais complet, ce qui est une amélioration par rapport au premier réseau. Néanmoins, cela reste faible pour l'utilisation que nous souhaitions en faire, et nous sommes limités à la puissance de notre matériel pour entrainer notre réseau.

Conclusion

Les objectifs de notre projet ne sont pas tous remplis mais sont encourageants. Nous avons pu extraire des données depuis le jeu ainsi que réaliser un réseau capable de jouer à un niveau. Nous avons ensuite pu l'améliorer afin de le faire jouer à n'importe quel niveau. Néanmoins il est impossible de valider ce dernier point, de par le manque de ressources.

De plus nous n'avons pas réalisé le code nécessaire pour que le réseau puisse jouer au jeu en temps réel. Ce code n'étant pas lié directement au réseau, la priorité était de réaliser un réseau fonctionnel en mesure de prédire correctement les actions du jeu.

Ce sujet nous à permis d'apprendre beaucoup de notions sur les réseaux de neurones, les algorithmes utilisés pour la reconnaissance et la classification des images. Nous avons pu parcourir de nombreuses documentations et articles sur cette technologie qui est actuellement en pleine ébullition.

Ne soyez pas surpris si la classe du réseau de neurone est appelée Bernard, c'est un client d'oeil à Bernard Low, le personnage de la série Westworld qui nous a inspiré dans le choix de ce projet.

Sources

- https://www.tensorflow.org
- http://colah.github.io
- http://ufldl.stanford.edu

Notes

Les sources du réseau sont disponibles sur Github : https://github.com/croziere/Bernard