Balloma Video Game playing Reinforcement Learning Agent.

Luis Rojas Aguilera
Udacity
Capstone Project Proposal

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Balloma Video Game playing Reinforcement Learning Agent.

Abstract

This work presents a capstone project proposal to achieve a Nanodegree in Machine Learning from Udacity. The objective of this work is to construct a video game playing robot, through the use of Deep Reinforcement Learning techniques. Also it should provide a methodology for test automation on game development process. This document provide descriptions on: *a)* problem to be solved, *b)* objectives, *c)* context, *d)* tools, *e)* metrics and *f)* techniques.

I. Introduction

Balloma is a single-player, android video game, developed by Black River Studios[1] at brasilian SIDIA[2]. During the game development process, after any new features are integrated, regression and functional tests must be executed in order to validate if new functionalities are properly working and side effects caused by new code have not appeared.

Currently such tests are executed manually by humans, thus as functionalities' stack increases also does testing complexity and workload. In order to aim testing process this proposal presents a Proof Of Concept method for test automation using Deep Reinforcement Learning techniques. Next sections presents further details on Balloma video game, the RL framework and how to use it for game automatic controlling.

II. Deep Reinforcement Learning

Reinforcement Learning (RL) is a field of Machine Learning that studies autonomous interactions among software agents and environments, and the way an agent can enhance its behavioral rules (policy) in order to maximize the reward it obtains from an environment. That is, at time step t, agent takes action a_t on a given environment E with transition dynamics $p(s_{t+1}|s_t, a_t)$ and current state s_t obtaining a reward r_t from a reward function $r(s_t, a_t)$ and transforming current state into s_{t+1}.

Currently RL proposes two main approaches to represent an agent behavioral rules: a) Value-based and b) Policy-based algorithms.

---

[1] http://blackriverstudios.net/

[2] https://www.sidia.com

Fig. 1. Balloma splash screen.

In value-based algorithms it is used a function called action-value ($q_*(s, a)$) that contains the expected cumulative reward of performing a given action while in a given environment's state. Such function can be used by the agent to decide which action to take in a given state by selecting from q the action that maximizes reward with a given probability ϵ that allows a desired exploration behavior so as it is applicable for non-deterministic environment dynamics. This approach is constrained to discrete action and state spaces.

Policy-based algorithms directly maps states and actions by a approximated policy function that represents the underlying stochastic distribution presented by environment's dynamics. While in the Value-based approach an heuristic should be defined to provide exploration behavior and avoid conditional randomness influenced by agent greediness, in Policy-based such heuristic is expressed in the policy function and is not necessarily fixed for each possible action-space setup. This approach is applicable to continuous action and state spaces.

For both approaches, agent behavior is guided by an optimal policy $\pi_*(s, a, \theta)$ evaluated by objective function $J(\theta) = \mathbb{E}[R(\tau)]$ of policy parameters θ, based on state-action-reward expectations on trajectory $\tau = S_0, A_0, R_1, S_1, ...$ drawn from a probability distribution $p(s', r|s, a)$. In such case is convenient using the **Bellman Expectation Equation** [5] to represent expected return of taking an action a_t while in state s_t and following a policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{1}$$

In this proposal I intend to apply a hybrid approach for RL called Deep Deterministic

Policy Gradients (DDPG) [1]. It follows the actor-critic [2] algorithm DPG [3] in which the agent's behavior is represented by function $\mu(s|\theta^{\mu})$, called the actor, that is evaluated by the critic, a non-linear action-value function $Q(s,a)$ parametrized by θ^{Q}, adjusted by minimizing the loss:

$$L(\theta^{Q}) = \mathbb{E}_{s_t\sim\rho^{\beta},a_t\sim,r_t\sim E}[(Q(s_t,a_t|\theta^{Q}) - y_t)^2] \tag{2}$$

where

$$y_t = r(s_t,a_t) + \gamma Q(s_{t+1},\mu(s_{t+1})|\theta^{Q}) \tag{3}$$

Actor function $\mu(s|\theta^{\mu})$ specifies the agent's policy by deterministically mapping states to a specific action. It is adjusted to maximize the expected reward from a start distribution $J = \mathbb{E}_{r_i,s_i\sim E,a_i\sim\pi}[R_1]$. That is attained by updating parameters θ^{μ} following the policy's performance gradients, expressed as:

$$\begin{aligned} \nabla_{\theta^{\mu}} &\approx \mathbb{E}_{s_t\sim\rho^{\beta}}[\nabla_{\theta^{\mu}} Q(s,a|\theta^{Q})|_{s=s_t,a=\mu(s_t|\theta^{\mu})}] \\ &= \mathbb{E}_{s_t\sim\rho^{\beta}}[\nabla_a Q(s,a|\theta^{Q})|_{s=s_t,a=\mu(s_t)} \nabla_{\theta_{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}] \end{aligned} \tag{4}$$

Both actor and critic functions are expressed by neuronal networks, updated following an approximation to a supervised learning approach. For that matter, target networks $\mu'(s|\theta^{u'})$ and $Q'(s,a|\theta^{Q'})$ are created, which are soft copies of actor and critic networks respectively. That soft copy is made from updated parameters as follows: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$

In order to provide exploration capabilities, during training, a different policy than that of the agent is used, obtained by adding noise sampled from a noise process η to the actor function. Such exploration policy is expressed as follows:

$$\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + \eta \tag{5}$$

Training follows an episodic process similar to Q-Learning. In each episode a set of agent-environment interactions, represented as a transition tuple (s_t, a_t, r_t, s_{t+1}) are stored in a replay buffer R [4]. A set of tuples are sampled from R and used to apply the updates previously explained in this section. This iterative process can be truncated based on optimization or time constraining conditions are met.

III. BALLOMA VIDEO GAME

Balloma is a single-player video game that runs on Android devices. It is composed of several scenes that initially appears locked and gets unlocked as the user completes unlocked scene's objective.

Each scene contains a constrained little terrain world with elements arranged and boundaries that if crossed makes the scene end. The scene's main element is a ball the player should control and carry to a remarked target position in the scene. To control such ball the player should touch and swipe the device's screen inputing a direction and speed he wants the ball to take. Once the ball is positioned at target, scene ends, a score is given to user and next scene is unlocked.

Fig. 2. Balloma's scene sample. Elements of scene are: a) The ball (blue), b) Floating elements (green), c) Score records (red), d) Target position (yellow)

While the user carries the ball to target he can make the ball go trough floating elements in the scene which makes the score increase. Also a timer is included in the scene which influences the final score, lower scene's completion running times provides higher rewards.

Figure 2 presents the game's first scene which appears unlocked by default. In Figure 2 the ball is bounded by a blue box. The blue diamonds bounded with a green box appears floating in the scene, if reached by the ball makes the score (upper left red box) increase. Target position is remarked inside a yellow box. Also there is a timer (lower left corner red box) that influences the final score.

IV. Environment

An interface is implemented to provide a RL suitable environment by which the agent can interact with the game to construct the optimized policy. Since the game runs in Android,

the environment should provide a controlling interface with an android physical or virtual device. Such functionality can be attained through the Android Debug Bridge (adb[3]).

A. States

In this proposal the states are represented by raw scene frames of 240x240. It means at each time step t the environment is at state $s_t \in \mathbb{R}^3$ formatted as a tensor of shape $(240, 240, 3)$. A similar approach was presented in [1] and [4], however scenes in those works are 2D spaces and in this case it is 3D. Frames are cropped to exclude score indicators and focus on more important pixels.

B. Actions

On each time step the agent can chose to take an action $a_t \in \mathbb{R}^3$. Actions are presented as a tuple $a_t = (\vartheta, \varkappa, \xi)$ that describes the swipe action: *a)* $vartheta$ size of the vector drawn by the swipe, *b)* $\varkappa$ angle with respect to screen horizontal of the vector drawn by the swipe and *c)* xi speed at which swipe is drawn.

Each episode starts with the first swipe inputted by the agent. If the balls falls off the terrain world or get to target position, environment is set into a terminal state. Also it will be constrained in time to avoid long episodes.

V. Reward function

At each time step, after the agent choses an action a_t and such action is applied to environment, a reward r_t is observed. If an episode gets into a failed terminal state (e.g ball falls out) reward of that step is computed as -1. On the other hand if episode ends successfully (i.e the ball hits the target), reward of that step is computed as +2.

For others time steps, reward is computed taking into account two coefficients: *a)* ratio between floating diamonds currently gathered over total (ω) and *b)* episode time passed since it started (φ). Also each parameter is multiplied by a weight coefficient α and β that regulates its importance, it permits experimenting different parameter's level of impact. In such cases reward is computed as:

$$r(s_t, a_t) = \alpha \frac{\omega_t}{\omega_N} - \beta \frac{\varphi_t}{\varphi_N} \tag{6}$$

There exists in the scene an object providing information on how many diamonds has been collected. During training, after an action is inputed into the scene a frame is captured and processed to extract the information needed to compute reward. In order to extract

[3]https://developer.android.com/studio/command-line/adb

that data, which is used by the reward function, in this work was implemented a picture matching procedure in which crops of predetermined portions of the scene are compared with previously tagged crops of the digits as it is rendered in the scene.

For each digit from 0 to 9, crops were previously gathered and tagged. Such digit images are compared with the pixels cropped from scene at specific coordinates it is expected the element with current scoring appears, in order to determine diamonds gathered at each step, during training.

Before comparison, crop's color space is transformed into binary. Then the *height*X*weight* shaped crops are subtracted and bits in the result are summed. If the result is below a predefined threshold, the crop is classified with the digit of the crop it was matched against.

Since it is necessary before hand the exact position of the elements in screen, this approach is dependent on the device's screen size, thus should be manually adjusted for each device it is used on. The code accompanying this report contains coordinates and crops compatible with a Samsung S8+ device screen. This method worked properly in 100% of cases.

VI. Benchmark Model

Previous researches have addressed video game playing agents with reinforcement learning. Mnih et al. [4] created Atari video game playing robots trough the use of an adaptation to Deep Q-Learning. This approach is considered as Deep Reinforcement Learning since makes use of neuronal networks to map states to actions directly. The method was tested on 49 Atari games assessing reward evolution throughout training. It was achieved professional human players comparable performance.

Deep Deterministic Policy Gradients [1], the method presented in this work, was used by its authors to create an agent that plays Torc, a racing video game. They compared effectiveness of the method when applied on low-dimensional features like acceleration, braking and steering and with high-order features like pixel representations (i.e game raw frames), assessing reward evolution throughout training. It was demonstrated by experimentation that similar performance is attained for both features presentations.

VII. Results

It was implemented a Reinforcement Learning setup composed by an agent, represented by a Convolutional Neuronal Network (ConvNet) and an environment, representative of Balloma Video Game, driven by an iterative procedure in which at every step the agent interacts with the environment and information from that interaction is used to adjust the agent's behavior with the goal of incrementing the reward the agent obtains when interacting with the environment.

The RL setup follows and Actor-Critic approach in which game actions are inferred by a ConvNet adjusted from experiences obtained by interaction with Balloma Video Game. Each

experience is compound of an action, reward and game scene frame, obtained by inputting the inferred action into the video game running in an android device and capturing the rendered frame. Another Neuronal Network, called the Critic, evaluates how well the Actor is performing and outputs a gradient that is used to adjust the Actor towards minimizing the loss and increasing the reward obtained from environment.

A. Actor ConvNet

The Actor Convolutional Neuronal Network architecture is presented in Table I. A similar architecture was proposed in the original paper of Deep Deterministic Policy Gradient [1].

Layer (type)	Output Shape	Param #
$states(InputLayer)$	(None, 84, 296, 9)	0
$conv2d\_1(Conv2D)$	(None, 83, 295, 32)	1184
$conv2d\_2(Conv2D)$	(None, 82, 294, 32)	4128
$conv2d\_3(Conv2D)$	(None, 81, 293, 32)	4128
$dense\_1(Dense)$	(None, 81, 293, 200)	6600
$dense\_2(Dense)$	(None, 81, 293, 200)	40200
$raw\_actions(Dense)$	(None, 81, 293, 3)	603
$global\_average\_pooling2d\_1$	(None, 3)	0
$actions(Lambda)$	(None, 3)	0
Total params: 56,843		
Trainable params: 56,843		
Non-trainable params: 0		

TABLE I. AGENT'S ACTOR CONVNET ARCHITECTURE.

The Actor's ConvNet architecture consists of 3 Conv layers of 32 filters each, without pooling layers. It is followed by two fully connected layers with 200 units. It was used Adam [7] for learning the network parameters with a learning rate of 10^{-4}. It is used the rectified non-linearity [8] for all layers. The final output layer of the actor is a $tanh$ layer, to bound the actions. The final layer weights and biases are initialized from a uniform distribution

B. Critic Neuronal Network

The Critic's Neuronal Network architecture is presented in Table II. A similar architecture was proposed in the original paper of Deep Deterministic Policy Gradient [1].

The Critic's network architecture consists of two inputs for states and actions respectively. Since states are tensors, that input is flattened before hidden layers. Following there are three dense layers in a row with 32, 32 and 64 units respectively.

It was used Adam [7] for learning the network parameters with a learning rate of 10^{-4}. It is used the rectified non-linearity [8] for all layers. The final layer weights and biases are initialized from a uniform distribution

Layer (type)	Output Shape	Param #	Connected to
$states(InputLayer)$	(None, 84, 296, 9)	0	
$flatten\_1(Flatten)$	(None, 223776)	0	$states$[0][0]
$actions(InputLayer)$	(None, 3)	0	
$dense\_1(Dense)$	(None, 32)	7160864	$flatten\_1$[0][0]
$dense\_3(Dense)$	(None, 32)	128	$actions$[0][0]
$dense\_2(Dense)$	(None, 64)	2112	$dense\_1$[0][0]
$dense\_4(Dense)$	(None, 64)	2112	$dense\_3$[0][0]
$add\_1(Add)$	(None, 64)	0	$dense\_2$[0][0] $dense\_4$[0][0]
$activation\_1(Activation)$	(None, 64)	0	add_1[0][0]
$q\_values(Dense)$	(None, 1)	65	$activation\_1$[0][0]

Total params: 7,165,281
Trainable params: 7,165,281
Non-trainable params: 0

TABLE II. AGENT'S CRITIC NEURONAL NET ARCHITECTURE.

REFERENCES

[1] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[2] Konda, Vijay R., and John N. Tsitsiklis. "Actor-critic algorithms." Advances in neural information processing systems. 2000.

[3] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14), Eric P. Xing and Tony Jebara (Eds.), Vol. 32. JMLR.org I-387-I-395.

[4] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Humanlevel control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[5] (2011) Bellman Equation. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA

[6] LeCun, Y. & Cortes, C. (2010), 'MNIST handwritten digit database'.

[7] Kingma, D. P., and J. L. Ba. "Adam: A method for stochastic optimization. arXiv 2014." arXiv preprint arXiv:1412.6980 (2014).

[8] Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier networks. In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W& CP Volume, volume 15, pp. 315–323, 2011.