Deep Learning In Video Games: Related Studies

A. Audren, B. Phan, University of Cincinnati, deep learning (fall 2023) students

I. Introduction

This document is the result of recent (2023) researches on the use of deep learning in video games. Especially the use of Q-learning for performances in Atari-like games. This paper summarizes state of the art models and performances achieved when trying to beat humans in video games. The reader will find classical approaches for use of deep learning in arcade, platform, strategic and mazed-based video games. A first part will introduce concepts of intelligent player agent with base principles. In a second time we will dive into a conceptual approach which includes optimization-based decision. Eventually, we will look at most commonly used models designed for simple games.

II. Base principles when learning to play video games

Deep learning can be applied to many types of games, considering arcade games, racing games, first person shooter games, open world games and many others. The choice of deep learning approach depends on the nature of the game, the available data, and the specific objectives of the intelligent agent. As an example, open world games do not have a clearly defined goal to achieve, so it might be hard to define for the agent. On the other hand, arcade games have a score or objective that makes it easier to define for the agent. Furthermore, there are many ways to define the state of the game as input for our agent. It can be the image displayed by the game, the position or velocity of the character.[1]
This is the reason why we choose to focus on arcade games, we can use as the state of the game the image displayed and the goal are well defined for most of them.

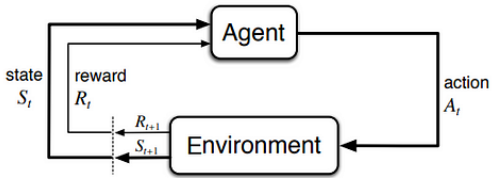

Figure 1 : agent-environnement interaction model

In reinforcement learning, the problem can be represented by a Markov Decision Process (MDP) as shown above. For our case in video games, the agent will decide which action it will make based on the current state of the game. Consequently, the state of the game will change and if the action helped him to get closer to finishing the game, then it will be rewarded. [2]

The main goal of a reinforcement learning (RL) agent is to maximize the cumulative reward it receives over time. This means that the agent aims to make a series of decisions and take actions in an environment in such a way that it accumulates as much total reward as possible throughout its interactions. Nonetheless, in some case we might want the agent to maximize immediate reward or futur reward and this is why the cumulative reward can be defined with a discount factor (γ between 0 and 1). The value of the discount factor will affect the behavior of the agent. With a lower value it will give less importance of the future reward and with a high value it will act for long term reward.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + .. = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad (2.1)$$

III. Conceptual approach – Optimization-based decisions

A. Dynamic optimization

Given a state in the game, we want our agent to perform the best possible action i.e., the one which is the most long-term rewarding. Even though we cannot forecast the future we can try to predict it by learning from patterns. If we succeed to predict the "long-term value" of a pair action-state, the agent will simply realize at each step of the game the action which has the maximum value. We will call this value the utility $q(s, a)$ of an action-state pair. This utility can be approached using the Bellman optimality principle: [3]

$$q^*(s_t, a_t) = E[R_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1})] \qquad (3.1)$$

The reward at R_{t+1} is given by the game at each step. While the utility of a pair action-

state has to be learned by the model. To make it learns, we can use the following equation (Q-learning):

$$q_{new}(s_t, a_t) = (1-\alpha)q_{old}(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1})] \qquad (3.2)$$

This approach is nothing else than reinforcement learning with α as learning rate. We are updating these utility values with agent's gaming experiences.

B. *Q-table approach*

If the game we are playing remains simple, we can use a Q-table that reassemble all the values for each pair action-state. This is a simple approach that works well when the number of states is relatively low. [3]

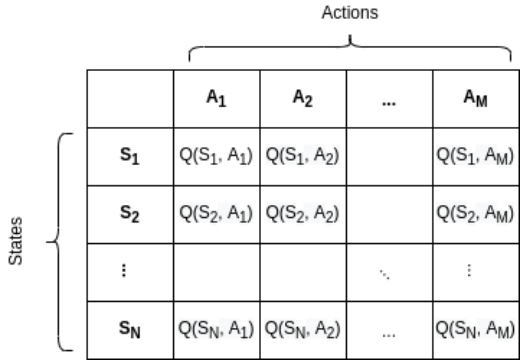

Figure 2: Q-table approach

At each step, the agent looks at the Q-table and choose the action with the maximum value in the row of its current state. The game then gives us feedback with a reward and a new state. From past gaming experiences, we can update that table using (3.2). We were in the state s_t and realized action a_t, the game gave us the reward R_{t+1} and we are now in the state s_{t+1}. We can estimate the expected value $E[\max_{a_{t+1}} q^*(s_{t+1}, a_{t+1})]$ using that same Q-table by looking for the maximum value in the row corresponding to state s_{t+1}.

C. *Deep neural network approach*

For most of video games, the previous approach cannot be used. Indeed, the number of states does not enable us to use a Q-table. Assuming we are playing a game with 60 frames per second (FPS), then there are 60 different states each second. The solution is to use a deep neural network to predict the utility of each available action given a state. We can use a convolutional neural network to transform each frame into a state and return us a vector of utility for each action $[q(s_t, a_{t,1}), q(s_t, a_{t,2}), \dots, q(s_t, a_{t,M})]$, if we have M possible actions at step t. [4]

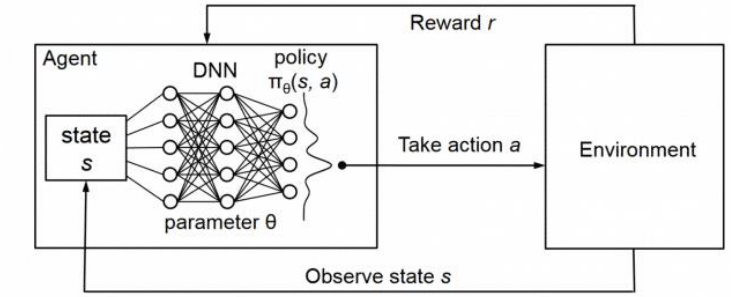

Figure 3: Deep Q-neural network approach

"Deep Q-Network (DQN) is based on Q-learning, where a neural network model learns to approximate $q^*(s_t, a_t)$ that estimates the expected return of taking action a_t in state s_t while following a behavior policy π. A simple network architecture consisting of two convolutional layers followed by a single fully-connected layer was used as a function approximator". [1]

IV. MODELS OVERVIEW FOR ATARI – LIKE GAMES

There are many different models using deep neural network to perform in Atari-like games but there are all built on top of each other's. First using Q-learning and adding over the years many implementations to make the agent learn better and faster. We are briefly presenting some of the most popular: [1]

- Deep Recurrent Q-Learning (DRQN)

Extends the DQN architecture with a recurrent layer before the output and works well for games with partially observable states.

- Double Deep Q-Networks (DDQN)

Instead of using one DNN to compute both the Q-values and the Q-targets, we use two DNNs. The first one computes the Q-values and the second one the Q-targets.

- Bootstrapped DQN

Improves exploration by training multiple Q-networks. A randomly sampled network is used during each training episode and bootstrap masks modulate the gradients to train the networks differently.

- Multi-threaded asynchronous DQN

Utilize multiple CPU threads on a single machine, reducing training roughly linear

to the number of parallel threads.

The performances have been summarized using human-normalized scores.

Results	Mean	Median	Year and orig. paper
DQN [161]	228%	79%	2013 [97]
Double DQN (DDQN) [161]	307%	118%	2015 [155]
Dueling DDQN [161]	373%	151%	2015 [161]
Prior. DDQN [161]	435%	124%	2015 [123]
Prior. Duel DDQN [161]	592%	172%	2015 [123]
A3C [63]	853%	N/A	2016 [96]
UNREAL [63]*	880%	250%	2016 [63]
NoisyNet-DQN [56]	N/A	118%	2017 [35]
Distr. DQN (C51) [9]	701%	178%	2017 [9]
Rainbow [56]	N/A	223%	2017 [56]
IMPALA [30]	958%	192%	2018 [30]
Ape-X DQN [61]	N/A	434%	2018 [61]

Table 1: Human-normalized scores of reinforcements learning models on Atari games.[1]

Architectures and functioning of models used for arcade games have also been summarized to create an overview of existing implementations.

Game	Method	Input			Architecture				Training					Miscellaneous				
		Features	Pixels	Text	CNN	RNN	Ext. Memory	MLP	Supervised	Q-learning	Actor-critic	ES	GA	Auxiliary Learning	Hierarchical	Intrinsic Motivation	Transfer Learning	Distribute
Atari 2600 (ALE)	DQN [97], [98]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	DRQN [51]	○	●	○	●	●	○	○	○	●	○	○	○	○	○	○	○	○
	UCTtoClassification [42]	○	●	○	●	○	○	○	●	○	○	○	○	○	○	○	○	○
	Gorila [100]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	●
	Double DQN [155]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	Prioritized DQN [123]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	Dueling DQN [161]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	Bootstrapped DQN [106]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	A3C / A2C [96]	○	●	○	●	●	○	○	○	○	●	○	○	○	○	○	○	●
	ACER [160]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	Progressive Networks [120]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	●	●
	UNREAL [63]	○	●	○	●	●	○	○	○	○	●	○	○	●	○	○	○	●
	Scalable Evolution Strategies [121]	○	●	○	●	○	○	○	○	○	○	●	○	○	○	○	○	●
	Distributional DQN (C51) [9]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	NoisyNet-DQN [35]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	NoisyNet-A3C [35]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	Rainbow [56]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	ACKTR [166]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	Deep GA [139]	○	●	○	●	○	○	○	○	○	○	○	●	○	○	○	○	●
	NS-ES / NSR-ES [24]	○	●	○	●	○	○	○	○	○	○	●	○	○	○	○	○	●
	IMPALA [30]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	TRPO [128]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	PPO [129]	○	●	○	●	●	○	○	○	○	●	○	○	○	○	○	○	●
	DQfD [57]	○	●	○	●	○	○	○	●	●	○	○	○	○	○	○	○	○
	Ape-X DQN [61]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	●
	Ape-X DQfD [112]	○	●	○	●	○	○	○	●	●	○	○	○	○	○	○	○	●
Ms. Pac-Man	Hybrid Reward Architecture (HRA) [156]	●	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
Montezuma's Revenge	Hierarchical-DQN (h-DQN) [77]	●	●	○	●	○	○	○	○	●	○	○	○	○	●	○	○	○
	DQN-CTS / DQN-PixelCNN [8]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	●	○	○

Table 2: Overview of models used for arcade games. [1]

REFERENCES

Online documents:

[1] Deep Learning for Video Game Playing
Niels Justesen1 , Philip Bontrager2 , Julian Togelius2 , Sebastian Risi1 1 IT University of Copenhagen, Copenhagen 2New York University, New York
arXiv:1708.07902v3 [cs.AI] 18 Feb 2019

[2] Reinforcement learning: Markov decision process (Part 1) – Blackburn Jul 18, 20219
https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision process-44c533ebf8da

[3] Q-learning algorithm: From explanation to implementation – Amrani Amine Dec 12, 2020
https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187

[4] Deep Q-networks: From theory to implementation - Amrani Amine Dec 19, 2020
https://towardsdatascience.com/deep-q-networks-theory-and-implementation-37543f60dd67