Computer Games Development SE607 Software Requirements Specification Year IV

Adrien Dudon
C00278154

26 April 2023

DECLARATION

Work submitted for assessment which does not include this declaration will not be assessed.

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) Adrien Dudon

Student Number(s): C00278154

Signature(s): _____

Date: 26 April 2023

Please note:

a) *Individual declaration is required by each student for joint projects.
b) Where projects are submitted electronically, students are required to submit their student number.
c) The Institute regulations on plagiarism are set out in Section 10 of Examination and Assessment Regulations published each year in the Student Handbook.

1. Contents

2. Acknowledgements

The completion of this project would not have been possible without the invaluable assistance of several individuals, whom I would like to acknowledge. Firstly, I would like to express my sincere gratitude to my supervisor, Dr Oisin Cawley, for providing me with expert guidance, constructive feedback, and unwavering support throughout the project. His insights and recommendations were invaluable in shaping my research and completing this project.

I would also like to extend my thanks to Dr Lei Shi, a lecturer at SETU Carlow, who generously provided me with the necessary hardware to produce the data required to obtain significant results. Their kind support and assistance are greatly appreciated.

3. Introduction

3.1. Intended audience and Reading suggestions

This document is intended for individuals seeking to understand the development goals of the software associated with the research project. The document provides a high-level description of the intended functionality of the code.

The software described in this document supports the research project titled "*What Advantages Does Using a Vision Transformer Model Offer Over Convolutional Neural Network in Playing Video Games?*" (Dudon 2023). The project aims to compare two neural network architectures in the context of Reinforcement Learning and determine the effectiveness and benefits of one over the other. For a more detailed description of the project, please refer to the project report.

3.2. Purposes

The software for this project implements the Double Deep Q Network algorithm (Hasselt et al. 2016)using PyTorch. The goal is to train an agent to play different Atari games using neural networks that focus on image recognition, specifically Convolutional Neural Networks and Swin Transformer models.

The purpose of this software is to support the comparison of the two neural network architectures and the assumptions made and explained in the research paper. The software should generate data, charts, and trains a Deep Reinforcement Learning agent to provide insights into the performance, benefits, and effectiveness of the neural networks.

4. Functional Specification

4.1. Technologies

The programming language chosen for this project is Python due to its popularity in the machine learning community and the vast number of technical resources available in Python. The project will primarily be developed using Jupyter Notebook, a library that allows interactive code cells and enhances code readability compared to standard Python files.

PyTorch, a widely-used neural network's framework, will be used for developing the neural network architectures. Although experimentation was conducted with TensorFlow, it did not yield conclusive results. PyTorch's API provides an easy-to-use development environment for any neural network architecture and offers performance benefits by compiling the code to run on the GPU.

To validate the implementation of the Double Deep Q Network algorithm, the Stable Baselines3 library will be used. This library provides reliable implementations of Reinforcement Learning algorithms in PyTorch (Raffin et al. 2021).

The environment will be emulated using the Python library Gymnasium, which provides a standard API for reinforcement learning and includes a collection of reference environments, including the famous Atari environment that will be used for this project (OpenAI and Farama Foundation 2021).

4.2. Reinforcement Learning Algorithm

The project requires the implementation of a variant of the Deep Q Network algorithm, specifically the Double Deep Q Network algorithm, which is a well-known Reinforcement Learning algorithm. The pseudocode of the algorithm is presented in *Algorithm 1*. An adaptation of this pseudocode is required to make it compatible with PyTorch and Gymnasium.

The Double Deep Q Network algorithm is an extension of the Deep Q Network algorithm (Mnih et al. 2015), which aims to reduce overestimation in the Q-values estimation by using two separate neural networks instead of one. One of the networks is used to estimate the action-value function, while the other is used to estimate the target action-value function. The target network is updated periodically, using the weights of the main network. This technique helps to stabilize the training of the agent and improve its performance.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\hat{\theta} = \theta$
For t = 1 to T **do**
  Reset the environment
  while game not done **do**
    t += 1
    if random(0,1) < ϵ greedy **do**
      Select a random action a
    else
      Select an action through policy network $a = argmax(Q(s,a;\theta))$
    end if

    Execute action a
    Store $(s,a,s',r,done)$ into replay buffer D
    Sample random minibatch of $(s,a,s',r,done)$ from D

    if t % train_frequency = 0:
      $target = r + (s',a;\theta) * (1 - done)$
      Compute loss $L(Q(s,a;\theta), target)$
      Update $Q(s,a;\theta)$ by loss L with gradient descent
    end if
  end while

  if t % target_update_interval = 0:
    Update $\hat{Q}$ with $\hat{\theta} = \theta$
  end if
end for

Algorithm 1: Double Deep Q Network (Pseudocode)

4.3. Neural networks

The Q-function in the DQN algorithm will be approximated using neural networks. For this purpose, two models, namely the Convolutional Neural Network (CNN) and the Swin Transformer, will be used interchangeably by the DQN algorithm with no major modifications required. The choice of these networks is based on their performance in image recognition tasks, which is the focus of this project. The architecture and hyperparameters of these networks will vary depending on the network used for training.

The CNN architecture used in this project will be the same as the one developed by Mnih et al. (2015). On the other hand, the Swin Transformer's parameters used should be the same as those described by Meng et al. (2022). A schematic representation of these two models is presented in *Figure 1* and *2*, respectively. More details regarding the neural network architectures and hyperparameters will be provided in the Technical Design Document.

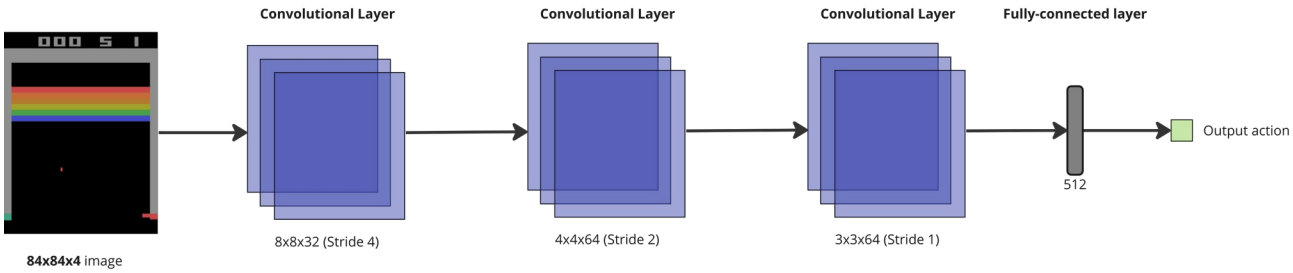

Figure 1: Convolutional Neural Network architecture of the Double DQN

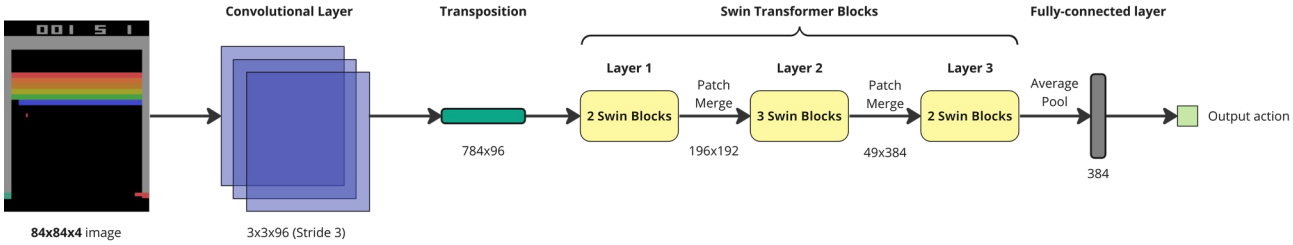

Figure 2: Swin Transformer architecture of the Double DQN

4.4. Environment

The environment simulation used for training the Reinforcement Learning agent is provided by the Gymnasium library. The agent will be trained on Atari environments, with a focus on Pong and Breakout. Specifically, the agent will be trained on version 5 of the Atari environment, which is the latest version available at the time of writing. Similar to the neural networks, the environment should be interchangeable without requiring any changes to the algorithm code.

4.5. Training specifications

The DQN algorithm parameters for training the Reinforcement Learning agents are specified in Table 1.

Table 1: DQN Parameters

Input	4x84x84
Optimizer	Adam
Adam learning rate	0.0001
Loss function	Smooth L1
Max timesteps	10,000,000
Target update interval	1,000
Learning starts	100,000
Train frequency	4
Replay buffer size	10,000

Batch size	32
Discount rate γ	0.99
Exploration fraction	0.1
Final exploration rate ε	0.01

The gradient descent step is performed every four time steps, as determined by the train frequency parameter. To prevent memory shortage issues on the training computer, the replay buffer size is limited to 10,000. The agent will be trained for 10,000,000 time steps, which strikes a balance between training time and performance.

4.6. Input

Due to the architecture of the neural network employed, the input to the DQN model must be in a specific format. This format requires the pre-processing of images in a manner similar to that described by Mnih et al. (2015) in their publication. Specifically, the images must be in greyscale and reduced to a size of 84x84 pixels. After each step taken within the environment, four consecutive frames must be stacked together to form an input of four 84x84 images. This stacking is necessary to enable the network to detect changes in the environment, as a single image is insufficient to provide a complete understanding of the dynamics. Thus, the input format should be a 4-stacked image of 84x84 dimensions, resulting in an array of 4x84x84 dimensions.

4.7. Data output

The software aims to gather meaningful data for comparison purposes. At the end of each episode in the environment, the mean reward of the last 100 episodes must be calculated and correlated to the time step number. Furthermore, the episodic return, which represents the score achieved by the agent at the end of each game, should be recorded. To achieve this goal, the software will use TensorBoard to store the training output, as it enables the direct addition of new scalar data to visualize through charts. Additionally, it is essential to capture a video of the training at reasonable intervals, such as every 500,000 timesteps. This step results in a total of 20 videos, allowing to observe the agent's performance visually and monitor its progression more intuitively than through the use of charts.

5. References

Dudon, A. (2023). *What advantages does using a Vision Transformer model offer over Convolutional Neural Network in playing video games?* [unpublished]. Project Report, Carlow: South East Technological University.

Hasselt, H. van, Guez, A. and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* [online], 30(1). Available from: https://ojs.aaai.org/index.php/AAAI/article/view/10295 [accessed 23 April 2023].

Meng, L., Goodwin, M., Yazidi, A. and Engelstad, P. (2022). Deep Reinforcement Learning with Swin Transformer. , 30 June 2022. Available from: http://arxiv.org/abs/2206.15269 [accessed 23 April 2023].

Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), pp.529–533.

OpenAI and Farama Foundation. (2021). *Gymnasium Documentation* [online]. *Gymnasium Documentation* [online]. Available from: https://gymnasium.farama.org/ [accessed 25 April 2023].

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M. and Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268), pp.1–8.