Double Trouble Deep Q-Network
Lab 3

1st Nicolaas P. Cawood 2376182
2nd Clarise Poobalan 383321
3rd Shikash Algu 2373769
4th Byron Gomes 0709942R

I. Introduction

Atari games have been used to illustrate the potential for reinforcement learning in many studies over the past decade. We implemented the first deep learning model which successfully learned the control policies from only the video feedback of these games instead of memory. The Deep Q-Network [1] and the improved Double Deep Q-Network [2] were implemented to put test their performances on the classic arcade game of Pong.

II. The training process

Fig. 1. summarizes the training process for the Deep Q Network (DQN). At each time step, experiences, which is a transition tuple consisting of state, action, next state and reward observations, are stored in replay memory [3]. Batches of experiences from replay memory are sampled randomly and are used to train the DQN [3]. By randomly sampling experiences, the high correlation that exists between consecutive samples is reduced which helps to stabilize and improve the DQN training [4]. The mini-batch size and experience replay size control the number of random samples and the size of the memory replay respectively.

The DQN controls two Q networks, viz. the Policy network and the Target network. The Policy network consists of updated (current) weights and the Target network consists of old weights. The current weights from the Policy network is copied to the target network every 1,000 iterations. For every iteration, a random experience is sampled from the replay memory and the current weights from the Policy network are updated by Stochastic Gradient Descent (SGD) in order to minimize the error between the Target network and Policy network by optimizing the loss function [3]. Actions are selected from the Policy network by using a ϵ-greedy approach.

We followed the training loop illustrated by figure 1. The training process was completed in the following order, over 1,000,000 frame steps:

1) An ϵ-greedy action is selected, which gives a higher probability of choosing the greedy action as the number of episodes increase.
2) The action is taken in the environment, and the next state, reward and terminal condition is observed.
3) We add the reward to the total rewards for the current episode, and the state, action, reward and next state transition are stored in the experience replay memory.
4) We use stochastic gradient descent to optimise our policy network weights from a 32 sample mini-batch from the experience replay memory. The TD loss we optimise are computed as the absolute element-wise error between the policy network and the target network using the Huber loss, which is less sensitive to outliers than the mean square error.
5) The target network's weights are updated to the Policy network's weights every 1,000 steps.

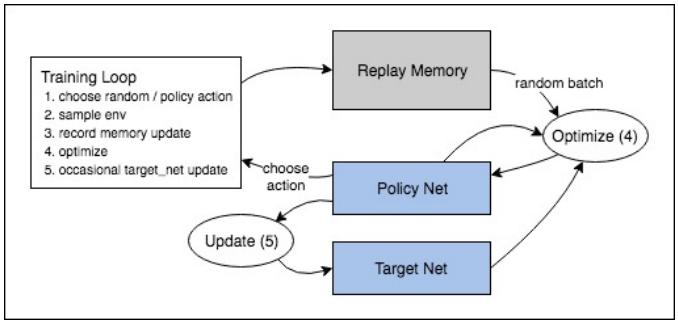

Fig. 1. The Deep Q-Network training loop [5].

III. Hyperparameters

We experimented with the "PongNoFrameskip-v4" environment from the OpenAI Gym API. The hyperparameters are presented in Appendix A. We followed the approach used for the training as per the original paper [1], which include

- The rewards were clipped at 1 and all the negative rewards to -1, and
- we treated both a win and a loss as the end of an episode.
- Mini-batches of size 32 were used as the history of each optimisation step.
- An ϵ-greedy behaviour policy was used with ϵ annealing from 1.0 to 0.1 over the first million frames, and fixing it to 0.1 afterwards, which reduces the exploration of the agent as the training gets closer to the end.

The RMSProp algorithm was replaced with an Adam optimizer, which is a stochastic gradient decent algorithm. We used the Huber Loss to compute the TD error which is minimized. We also excluded the frame-skipping technique that was used in the original paper.

A. Q-Network architecture

The Q Network architecture implemented is identical to the architecture from the original paper [6]. The DQN was implemented using PyTorch. The model was programmed as a linear stacking of layers, each separated by Rectifier Linear Units (ReLu) and the architecture used followed:

- The input layer: a convolution layer, with 32 8x8 filters and a stride of 4.
- The first hidden layer : a convolution layer, with 64 4x4 filters, and a stride of 2.
- The second hidden layer : a convolution layer, with 64 3x3 filters, and a stride of 1.
- The third hidden layer : a fully connected linear layer, with an output length of 512.
- The output layer: a fully-connected linear layer, with a single output for each action.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2015.

[3] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[4] D. Lizard, "Deep Q-Learning, Combining Neural Networks And Reinforcement Learning," uRL: https://deeplizard.com/learn/video/wrBUkpiRvCA. Last visited on 2020/09/29.

[5] "Reinforcement learning (dqn) tutorial¶." [Online]. Available: https://pytorch.org/tutorials/intermediate/reinforcementqlearning.html

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

APPENDIX A
MODEL HYPERPARAMETERS

Hyperparameter	**Value**	**Description**
learning rate	1e-4	The rate of descent at which the Q-network parameters are updated using stochastic gradient descent.
mini-batch size	32	The number of experience samples used to perform the Q-learning updates.
target network update	1,000	The number of iterations before each target network update.
experience replay size	1e6	The memory from which the mini-batches are sampled for the optimisation.
discount factor	0.99	The gamma parameter, or discount factor, for the Q-learning update.
initial exploration	1.0	Initial value for epsilon.
final exploration	0.1	The final value for epsilon.
exploration frames	1e6	The number of frames that the epsilon value linearly anneals over.
no-op max	30	Maximum number of ”do-nothing” actions by the agent during the start of an episode.
action repeat	4	The number of frames an action is repeated.