Reinforcement learning with DQN: Pong

Kamran Thomas Alimagham, Mihai Nipomici, Stefan Birs and Tiberiu-Ioan Szatmari

Abstract— **The following paper details the implementation of a deep learning agent using reinforcement learning, specifically it details the use of deep Q-networks to achieve state of the art results in the context of Pong, the classic Atari 2600 console game. The given solution includes advanced optimization tools such as the epsilon-greedy algorithm, a replay buffer and a target network to further increase the accuracy of the agent. Results show that our model manages to outperform the hard-coded Pong agent, winning with $21 - 0$, the maximum score.**

I. INTRODUCTION

A. Motivation

Reinforcement learning (RL) is a topic that has become increasingly popular over the last few years. It all started in 2015 with the release of AlphaGo by DeepMind, which was the first program that managed to beat a Go grand master [1]. Since then, three more modern and complex models have been created from AlphaGo and they are AlphaGo Master, AlphaGo Zero and AlphaZero [4].
To tackle a wider audience OpenAI presented their own platform for reinforcement learning called OpenAI Gym and in turn made it free to the public to allow for fast creation of RL prototypes, which lead to a research boom in this field. One of the current benchmarks for state of the art models is to be able to solve multiple games on the Atari 2600 console [2].
For the scope of this report, we will go into depth about solving Pong, one of the oldest video games, created in 1972. Pong is a great example of a single agent environment with a large state space, where the player competes against a hard-coded agent, and the goal of the game is to defeat the opponent by being the first to reach the score limit.

B. Pong Environment

The chosen Gym environment, "Pong NoFrameskip-v4", aims to give a baseline on which to train reinforcement models that can beat the hard-coded agent, by being the first to achieve a score of 21 points, drawing the game to an end.

In Pong, the game starts with a paddle for each player and a ball that starts with a set velocity and direction. The paddles can only move on the y axis, where the ball can move in 2D space. By hitting the ball at certain angles the direction and speed of the ball will be changed, which in turn creates a harder environment to solve as the game progresses. Once a point is scored by either of the players, the new round starts in a similar fashion with the ball in the middle of the screen.

The Pong environment provides a partially observable state and by using previous frames from the game we can derive more information about the current state of the game. This can be seen when trying to figure out the direction of either the ball or one of the paddles, which cannot be guessed from a single image. A similar case happens with the speed of the ball, which can be deduced by approximating the distance travelled between multiple frames of the game. As a minimalist approach, the reinforcement agent can solve the Pong game with only three actions: Up, Down and NoOp. To make the agent play continuous games, a new action called Start is introduced and to have the agent generalize better to other Atari 2600 games the agent will have a total of six possible moves to make, which correspond to the controller of the game console. This means that moving the joystick down will make the paddle go down and moving the joystick to the left will also make the paddle go down but at a different speed. The controls are mirrored when trying to move the paddle upwards towards the top of the screen.

C. Purpose

This paper will showcase our approach at trying to solve the Pong challenge. The paper goes into detail about implementing several techniques that were used to create and improve the agent. The agent was built in such a way that it would be feasible to generalize to other Atari games but that is not the main focus of this project.

II. BACKGROUND

A. The Markov Property

The background for reinforcement learning are Markov Decision Processes (MDPs), where an environment fulfills the Markov property: observations from the environment are sufficient for choosing optimal actions, states can be distinguished. Unfortunately, one observation, one image from Pong doesn't capture all necessary information. It is impossible to obtain the speed or direction of the ball or the opponent's paddle. Such an environment is defined as a partially observable MDP, an MDP where the Markov property is broken. To solve this, a stack of k subsequent game frames are joined, and the stack is considered as one observation. The extra information allows for the inference of state dynamics (speed and direction of the ball) [1].

B. Agent vs. Environment

In reinforcement learning, an agent perceives its environment through observations of the states and rewards, and acts upon it through actions [4]. The goal for the agent is

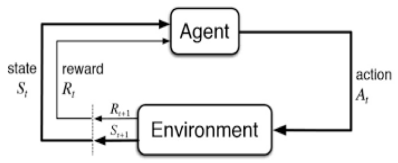

Fig. 1. Agent vs. environment control loop [7].

to learn which actions maximize the reward, given what it learned from the environment.

The description of each term in Figure 1 can be seen below:

- Agent: Pong AI model that is trained.
- Action: Output of our model.
- Environment: Everything that determines the state of the game.
- State: How the agent perceives the environment (the 4 frames).
- Reward: What the agent gains after taking an action in the environment.
  - If the agent missed the ball (losing): -1
  - If the opponent missed the ball (winning): $+1$

C. Q-Network

In the case of Pong, the number of observable states is extremely large (10^{70802}). A change of one pixel doesn't make a significant difference, which in turn makes it more efficient to treat similar states as a single state. A solution to this problem is to use a non-linear representation that maps both state and action onto a value. This "quality" or $Q^{\pi}(s,a)$ function returns the reward value based on the given state s, action taken a and following policy π - a Q-network - an approximate source of behavior for the learning agent.

D. Exploration vs. Exploitation

The basis of reinforcement learning is interaction with the environment, to obtain training data. In simple environments, random actions, such as moving up or moving down can be a possible solution. However, for Pong the probability of scoring a point by randomly moving the paddle is extremely small, which in turn leads to a very long time for such a situation to occur. Alternatively, if a good Q-network is used as a behavior source, it will provide the agent with relevant data to train on. The issue appears when the Q-network isn't working as intended, such as in the beginning of training. This leads to the agent taking bad actions for some states and being stuck without the option of behaving differently - the **exploration-exploitation dilemma**. The agent needs to explore the environment to build an accurate picture of state and action outcomes, while on the other hand, it should also efficiently interact with the environment. It shouldn't waste time by taking random actions for which the outcome is known. Random behavior is more useful in the beginning of the training and as training progresses it becomes inefficient, moment in which the Q-network should decide how to act. A method of accounting for both situations is the **epsilon-greedy** algorithm which switches between random actions and Q actions by the probability hyper-parameter ϵ. In practice ϵ starts at 100 %, meaning all actions are chosen randomly, and decreases during training until a small value, such as 2%.

E. Deep Q-learning

The core of Q-learning is similar to the process of supervised learning, with the goal of approximating a complex non-linear function $Q(s,a)$ with neural networks by means of stochastic gradient descent (SGD). Supervised learning requires the existence of targets for $Q(s,a)$, which are calculated using the Bellman equation [5]:

$$Q(s,a) = r + \gamma max_{a' \in A} Q_{s',a'}$$

It shows that the value of an action a given a state s is the immediate reward r added with the maximum expected reward from the next state and action, s' and a' - it takes into account the future result, not just the immediate one. However, SGD optimization requires for the training data to be independent and identically distributed (i.i.d). But samples are not independent, belonging to the same episode. Samples are not optimally distributed either. They are the result of the chosen policy, random or the currently learned policy (epsilon-greedy). The solution is using a **replay buffer**, storing past experience and using it for training at certain intervals. Training data is appended to the buffer which results in replacing older experience when full.

Another issue related to the lack of i.i.d data is the iterative nature of updating the Q values. Overriding them when new samples are taken from the environment can lead to unstable training. As such, a blending technique is used, which averages the old and new values of Q using learning rate α, with a value between 0 and 1:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma max_{a' \in A} Q_{s',a'})$$

The value of $Q(s,a)$ is obtained via $Q(s',a')$, and they differ by a single step, resulting in very similar states that are hard to distinguish. Furthermore, when network parameters are updated to improve $Q(s,a)$, $Q(s',a')$ is indirectly changed as well, which translates in highly unstable training. The solution is using a **target network** $\hat{Q}(s,a)$, which is a copy of the main network, to be used for the $Q(s',a')$ value, in the Bellman equation. The difference is, the target network is synchronized with the main network periodically, every N steps.

III. NETWORK ARCHITECTURE

This section starts with a brief overview of the relevant types of neural networks (feedforward and convolutional) to understand the network architecture used in this project to solve Pong problem. Afterwards, our model is described through a simple and detailed diagram, which used two of the main types of neural networks covered in the deep learning course.

A. *Feedforward Neural Network (FFNN)*

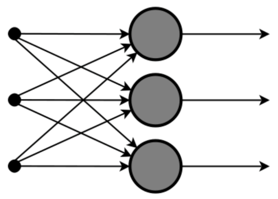

Fig. 2. Feedforward neural network (single layer) [8].

This is the simplest artificial neural network (ANN) which may have a single layer or many hidden layers. The idea of the ANN is to pass data through the different input nodes till it reached the output node. Also known as forward propagation (hence the name feedforward), which is achieved by using a classifying activation function [8]. As seen in Figure 2, this ANN calculates the sum of the products of the inputs and weights, which is fed to the output.

B. *Convolutional Neural Network (CNN)*

The CNN contains one or more convolutional layers, which can either be completely interconnected or pooled [8]. Before passing the result to the next layer, the convolutional layer uses a convolutional operation on the input. Due to this convolutional operation, the network can be much deeper but with much fewer parameters [8].

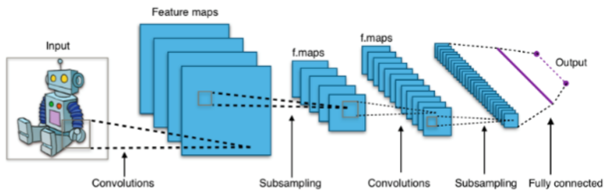

Fig. 3. Example of a convolutional neural network [8].

C. *Our Model*

Our network architecture is based on DeepMind's *Nature* research paper. The first three layers are convolutional then followed by two fully connected ones. Rectified linear Units (ReLU) non-liniarities were added in between each layer. The output of the model consists of Q-values for each possible action in the environment, without ReLU being applied because Q-values can have any value. The idea behind having all Q-values calculated only with one pass through the network is that it helps to increase speed considerably in comparison to treating $Q(s, a)$ literally and feeding observations and actions to the network to obtain the value of the action. The first convolutional layer has 32 channels with a kernel size of 8 and a stride of 4. The second one has 64 channels, kernel of 4 and stride of 2. The last convolutional layer consists of 64 channels, kernel size of 3 and stride 1. We have big kernel and stride sizes in the beginning due to vast amount of data and it is very computationally expensive to explore it in detail. Therefore by each layer we are decreasing this values since we want to capture more complex and useful information from the images.
As described above, the model is composed of two parts: convolution and sequential. In order to be able to use outputted data from the convolutional layers for feedforward ones we had to reshape the 3D tensors into 1D vectors in the **forward()** function of the model.

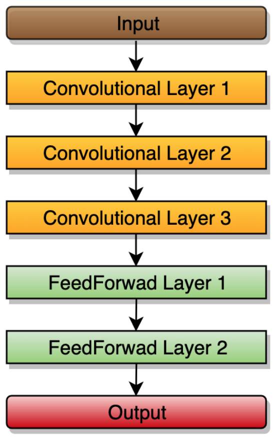

Fig. 4. Simple version of network architecture.

Convolutional Neural Network architecture. Composed of three convolutional layers with increasing depth sizes, two fully connected layers and rectified linear unit activation functions, as seen in Figure 5.

IV. RESULTS

All the training done for the agent was done using Google's Colaboratory that allowed us to accelerate the training using Tesla K80 and Tesla P100 NVIDIA GPUs. The tests we run consisted of the our agent trained using deep Q-networks playing against the hard-coded AI implemented in Pong. To minimize the training load and time we chose to experiment and tune the hyperparameters of our network after the implementation of our core features such as the epsilon-greedy algorithm, the replay buffer and the addition of the target network.

From the initial frame size of the game, the top part has been cropped out because it does not hold any pertinent information for the agent and the image was reshaped to a size of 84x84 to save on the number of computations. The first part of the network was comprised of three convolutional layers with kernels sizes going down from an original size of 8x8 and down to 3x3 and with an increase in the number of channels from 32 to 64 across the three layers. The output of the last convolutional layer had to be flattened to fit the fully connected layers, one comprised of 3136 neurons and a final layer of 512 neurons. The architecture described above was included in Figure 5.
The output of the above mentioned neural network was one of the six actions that the agent could take in the environment. From each of the actions, a reward was computed using the Bellman equation and the agent followed the action that had a maximum reward.

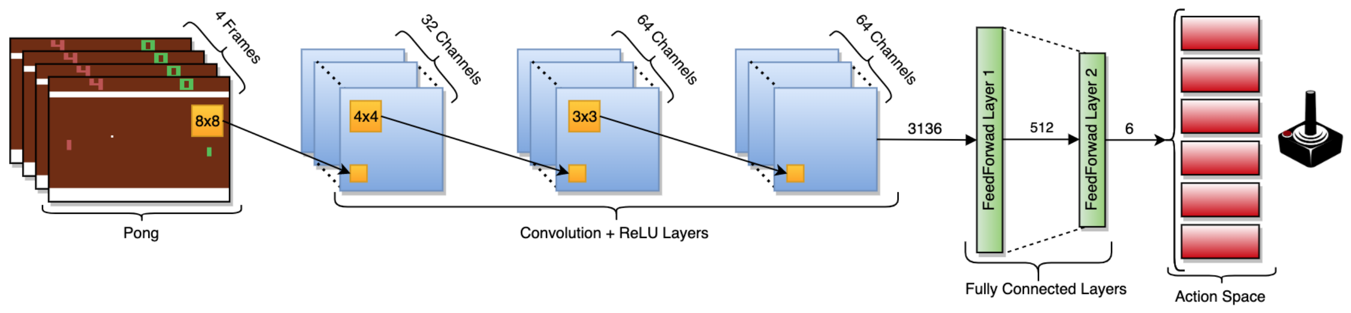

Fig. 5. Detailed version of network architecture.

Our best model managed to achieve a maximum reward of 19.52 after playing 382 games against the hard coded agent in Pong. This result was achieved using the epsilon greedy algorithm that linearly decreased the chance to explore the environment. As it can be seen in Figure 6, after the first 262k frames, the agent had only a small chance to explore and it was mostly focusing on exploiting the knowledge that it has gathered in order to win the round at hand.

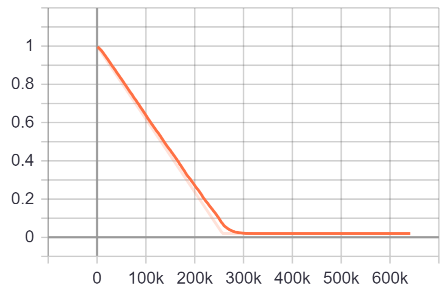

Fig. 6. Linear decrease of the epsilon value during the training loop

We introduced a buffer that was filled with the last 10 000 frames in order to try and break the correlation between steps and force the agent to apply new actions. Another major help was the implementation of the scheduler during the training of the network. The scheduler started to reduce the learning rate when on a plateau, which in turn helped the network overcome local minimum points and further increased the accuracy of the network. The local reward was plotted in Figure 7 and it can clearly be seen that the reward fluctuates at different time stamps, but in Figure 8 we can see that the mean reward, which was defined as the average reward over the last 100 frames has a smooth curve and converges at point corresponding to our reward value of 19.52.

TABLE I

NETWORK HYPERPARAMETERS USED DURING THE TRAINING OF THE AGENT USING REINFORCEMENT LEARNING AND DEEP Q NETWORKS.

Nr.	Training Games	Epsilon Decay	Batch size	Buffer Size	Scheduler	Mean Reward
1.	347	200k	32	10000	Yes	14.56
2.	470	200k	64	7000	No	16
3.	342	100k	32	10000	No	16.69
4.	382	262k	32	15000	Yes	19.52

In Table I we have included three more models that we created during the project period, where model number four

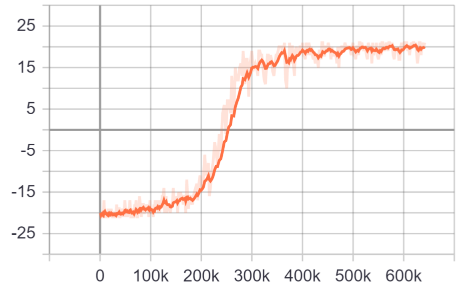

Fig. 7. Reward value plotted against the frame numbers.

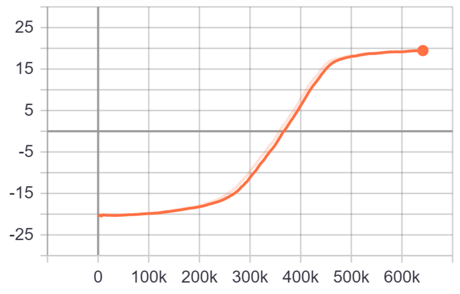

Fig. 8. Mean reward value plotted against the frame numbers.

in the table corresponds to the model that was discussed in this section. At a first glance we can see that 19.52 was the highest accuracy that we achieved and that was a result of a higher range of exploration given by the epsilon value and a double batch size compared to model number 1 and number 3. The finer tuned scheduler that did not kick in from the beginning of the training loop also played a good part in achieving such a high accuracy. During the development phase of the model we encountered a number of issues where the learning rate would decrease so fast that the model would not learn anything very early and as such we decided to only allow the schedule to decrease the learning rate further on in the training loop.

V. OTHER WORK

A number of other approaches were explored. They are included here to present other possible ways of solving the Pong challenge:

- **DeepMind**: A model that's similar to the presented solution, capable of learning successful policies from the pixel input using RL. It surpassed the performance of previous algorithms and achieved a level comparable to a professional human player across 49 Atari games, using the same DQN algorithm, architecture and hyperparameters. Epsilon-greedy, replay buffer and target network were used, with a neural architecture made of three convolutional layers and two fully connected layers, with a single output for every valid joystick action and ReLU as activation functions [3].

- **Policy Gradients**: A simpler model, proposed by Andrej Karpathy. Although policy gradients require more fine-tuning, it was shown that this method can achieve better results than Q-learning when tuned well. The main advantage is that it is an end-to-end solution: there is an explicit policy and a principled approach to directly optimize the reward. The network is defined by two fully connected layers, with ReLU and Sigmoid activation functions [9].

VI. FUTURE WORK

As previously discussed in Section I and Section V, current state of the art reinforcement learning models are able to solve multiple games with the same agent. One such example is the famous agent created by DeepMind, which is able to solve up to 49 games. This leads us to the conclusion that this should be the next step taken in this project, trying to adapt our agent to play different games such as Bomberman or shooters. The first step required would be to standardize the reward the agent receives because, as we have seen, the reward in pong is either one or minus one for winning or losing a point respectively. On the other hand, in a shooter game, the reward could be 100 per defeated enemy. Another issue which was already addressed in this report consists of the generalization of the actions to the controller of the Atari 2600 console. We have chosen six actions as the output of our neural network, whereas Pong could be solved with only three. This was done to better generalize the network to play multiple Atari games.

Currently we have a attempted implementations where our RL agent plays, and beats, the hard-coded AI agent in Pong and where the player can try imitation learning to have the agent train on a particular set of moves. A reasonable extension of this would be to have the player test his skills against the trained reinforcement agent.

VII. CONCLUSION

As stated in Section I, the main goal of this project was to create a fully functional agent with the help of reinforcement learning in order to solve Pong. In Section II we have detailed the exploration vs. exploitation dilemma and in turn the epsilon-greedy algorithm that addresses it. We have also explained what Q-learning means and how the Q-function can be an approximate source of behaviour, such as applying specific actions to generate new states. Moving on to Section III, the entire neural network was discussed in detail, showing that the architecture chosen by us was a CNN that read the raw pixels from the Pong game and generated a reward for each action available at the current state. Lastly, in Section IV we have covered the main outcome of this project. In Table I, it can clearly be seen that considerable improvements in the mean reward were achieved, compared to our original architecture. This was due to multiple iterations over the contents of the network and a fine tuning of the parameters to promote a high reward, as well as a good generalization of the agent. As such, the main goal of this report was achieved during the project period and clear goals are set for the project to be further expanded.

APPENDIX

Link towards the GitHub repository: Repository

REFERENCES

[1] Lapan, M., 2018. Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more. Packt Publishing Ltd.

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

[3] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529– 533 (2015) doi:10.1038/nature14236

[4] Géron, A., 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

[5] Vedpathak, O., 2019, Playing Pong using Reinforcement Learning, https://towardsdatascience.com/intro-to-reinforcement-learning-pong-92a94aa0f84d, Medium

[6] Trazzi, Michaël, Spinning Up a Pong AI With Deep Reinforcement Learning, https://blog.floydhub.com/spinning-up-with-deep-reinforcement-learning/ , FloydHub Blog, 2019, May

[7] Bhatt, Shweta, Reinforcement Learning 101, https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292 , Towards Data Science, 2019, April

[8] Anukrati, A Comprehensive Guide to Types of Neural Networks, https://www.digitalvidya.com/blog/types-of-neural-networks/ , 2019, December

[9] Karpathy, A., 2016, Deep Reinforcement Learning: Pong from Pixels, https://karpathy.github.io/2016/05/31/rl/, Github