Deep Learning Project 3
Doom RL

Alejandro Morales-Peña
Costa Rica Institute of Technology
San José, Costa Rica
Email: alejandromp12.4@gmail.com

Javier Cordero Quirós
Costa Rica Institute of Technology
San José, Costa Rica
Email: javiercordero1296@gmail.com

***Abstract*—In this paper, we train and test a deep reinforcement learning model for playing Doom, which is designed with a Duel DQN baseline Architecture and then improved with an Inception module. Incremental experimentation was applied to observe the effects of each addition. Results show improvements in this modification as the agent reaches higher scores for most episodes. We also add an extra preprocessing stage based on segmentation of the important details of each frame, which didn't seem to affect the learning process.**

***Index Terms*—Deep Learning, Reinforcement Learning, Doom, Policy Gradient**

1. Introduction

One of the most common limitations in Machine Learning is having a labeled dataset to use supervised learning techniques, collecting the data and adding labels is not only expensive but time consuming as well. Since the majority of data in the web has no label, the unsupervised learning paradigm becomes important to train deep learning models in a more automated fashion. Although important progress has been achieved for it, there's a third option called **Reinforcement Learning (RL)** which is based on how an intelligent agent gets rewards from the actions it takes in its environment. RL is not considered supervised learning since rewards don't come in input/output pairs and they are not meant to correct sub-optimal actions. The goal is then to find a balance between exploration (of uncharted territory) and exploitation (of current knowledge) [1].

In this paper we present *Doom RL*, a neural network that learns to play the classic first person shooter game Doom through reinforcement learning. We use the concept of policy gradient to create a differentiable model that can be trained without giving previous context of the environment (the game), just by trying and failing. We aim to execute and study 3 different experiments: the first one is training the main architecture that we use a baseline model, then we add preprocessing of the input and finally we modify the architecture to try to improve the network's performance.

The remainder of this paper is organized in 6 sections. Section 2 describes the paradigm of reinforcement learning and brings a general overview of the Doom environment used to train our model. In Section 3, we describe the methodology implemented to tackle down the different scenarios that our agent has to face. Section 4 presents the description of the RL model architecture. The results obtained with the experiments carried out are presented in section 5. In section 6, we perform a deep analysis of the differences between the three experiments. Finally, in section 7 this paper is concluded and future directions for this work are provided.

2. Background

2.1. Doom

Doom is a first-person shooter presented with early 3D graphics. The player controls an unnamed space marine—later termed "Doomguy" through a series of levels set in military bases on the moons of Mars and in Hell. To finish a level, the player must traverse through the area to reach a marked exit room []. Levels are grouped together into named episodes, with the final level focusing on a boss fight with a particularly difficult enemy. While the environment is presented in a 3D perspective, the enemies and objects are instead 2D sprites presented from several preset viewing angles, a technique sometimes referred to as 2.5D graphics with its technical name called ray casting. Levels are often labyrinthine, and a full screen automap is available which shows the areas explored to that point. While traversing the levels, the player must fight a variety of enemies, including demons and possessed undead humans, while managing supplies of ammunition, health, and armor. Enemies often appear in large groups, and the game features five difficulty levels which increase the quantity and damage done by enemies, with enemies respawning upon death and moving faster than normal on the hardest difficulty setting. The monsters have very simple behavior, consisting of either moving toward their opponent, or attacking by throwing fireballs, biting, and clawing. The player can find weapons and ammunition throughout the levels or can collect them from dead enemies, including a pistol, a chainsaw, a plasma rifle, and the BFG 9000. Power-ups include health or armor points, a mapping computer, partial invisibility, a safety suit

against toxic waste, invulnerability, or a super-strong melee berserker status.

Figure 1: Doom gameplay example

2.1.1. ViZDoom Framework. ViZDoom is as framework to help developers train and test Deep reinforcement learning models designed to play the Doom. The created AI bots play only the visual information (the screen buffer). ViZDoom is primarily intended for research in machine visual learning [6]. Some of its features are: API for Python, C++, and Julia. Easy-to-create custom scenarios (visual editors, scripting language and examples available), Async and sync single-player and multi-player modes, Fast (up to 7000 fps in sync mode, single threaded), etc. So basically the framework provides flow control methods such as initializing, starting a new episode, making an action, getting the current state (frame and metainfo), etc. Also there are button settings, game variables, reward settings and others. One of the most important part is the training scenario, the purpose of the scenario is just to check if using this framework to train some AI in a 3D environment is feasible. The basic map is a rectangle with gray walls, ceiling and floor. Player is spawned along the longer wall, in the center. A red, circular monster is spawned randomly somewhere along the opposite wall. Player can only (config) go left/right and shoot. 1 hit is enough to kill the monster. Episode finishes when monster is killed or on timeout. But there are other scenarios with some modifications for instance, the *Deadly Corridor* teaches the agent to navigate towards his fundamental goal (the vest) and make sure he survives at the same time. Other scenarios are *Defend the center*, *Defend the line*, *Health Gathering*, *Take Cover*. All of these have a different teaching objective and use different rewards to do so.

2.2. Policy Gradient

In the reinforcement learning context, the environment is the one that gives rewards to the agent depending on its actions. It is typically stated in the form of a Markov decision, which provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming. So the problem is how to to use past experience to find out which actions lead to higher cumulative rewards. That's where the concept of policy comes in handy, it is a strategy that an agent uses in pursuit of goals. The policy dictates the actions that the agent takes as a function of the agent's state and the environment. In mathematics it is defined as a tuple of the form (S,A,P,R) [2] where the first element S is a set of internal states of the agent, for example it could consist of the position of the agent on a board plus, if necessary, some parameters. The second element is a set A containing the actions of the agent. The actions correspond to the possible behaviors that the agent can take in relation to the environment. Together, the set of all actions spans the action space for that agent. Third, the matrix P containing the probability of transition from one state to another, and the fourth element R comprises the reward function for the agent. It takes as input the state of the agent and outputs a real number that corresponds to the agent's reward. Finally the agent can take several polices into account and evaluate from experience which one gains a higher reward.

3. Methodology

3.1. Data Preprocessing

In the first place, we did a simple reduction to the resolution of the frames extracted from the Doom game. We changed them from $640X480$ to $30X45$.

Lastly, we took advantage of the labels_buffer offered by the ViZDoom framework [6]. The frames rendered under this approach come up with the form detailed in the figure 2. In addition, we implemented a Gaussian Blur filter with the objective of reducing the image noise and details. Once extracted the sample Blur Gaussian matrix from the frame, we used an automatic thresholding method to produce a binary image. We apply the threshold t such that pixels with gray-scale values on one side of t will be turned "on", while pixels with gray scale values on the other side will be turned "off" [7]. This process is depicted with the histogram of the figure 3. For the specific example of the image shown in 2, the threshold $t = 0.466796875$. Therefore, we perform a binary mask to turn all the values bigger than t in the Blur Gaussian matrix ($binary\_mask = blurred\_image > t$), which generates the image presented in the figure 4.

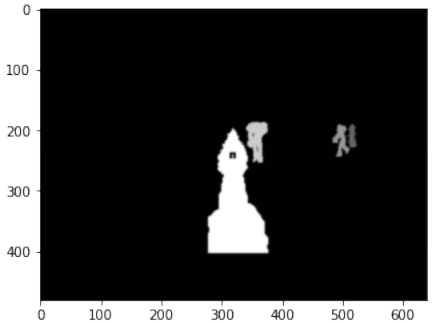

Figure 2: Labels buffer example in gray-scale format.

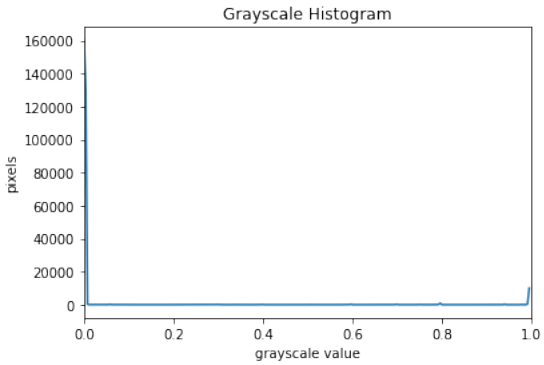

Figure 3: Gray-scale histogram of figure 2.

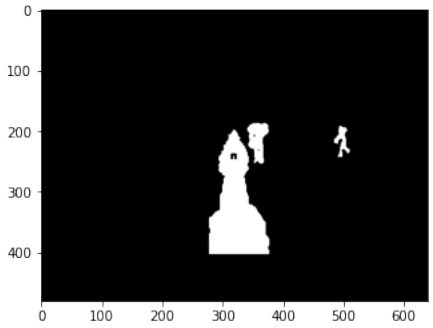

Figure 4: image produced after applying the binary mask to the figure 2, based on the threshold extracted from the figure 3.

3.2. Model Training and Testing

To implement, train and test our Doom RL model we used the Google Collaborator platform with a GPU runtime to accelarate the training process. We loaded the different scenarios using the Kaggle library. To determine the training hyper parameters, we took [4] as a reference, the learning rate was 0.00025, discount factor of 99%, 10 epochs of training, batch size of 64, 100 test episodes per epoch. To teach our model to play we used the scenarios *Basic* , *Defend the center*, *Health Gathering*, *Defend the line* in that order (order is important for a more quick learning in advanced scenarios). Regarding the testing section, we planned 3 incremental experiments: Experiment 1 executes the baseline model called Duel DQN described in section 4.1. Experiment 2 is almost the same as the first one but this time with the Preprocessing described in section 3.1. The Experiment 3 is about changing the architecture to use Inception modules as described in section 4.2. By running these experiments in order, we can compare results incremental results that may indicate the effects of each stage.

4. Doom RL Model Architecture

In this research we divide our work in two different architectures. At first, we implemented a baseline model based on a dueling network (Duel DQN architecture), which explicitly separates the representation of state values and (state-dependent) action advantages [4]. And, second we performed an enhancement to the Duel DQN architecture by adding *1×1* convolutional layers or "inception modules" to increase the network depth [5].

4.1. Duel DQN Architecture (Baseline)

The Duel DQN (D-DQN), is a neural network architecture for model-free reinforcement learning. This dueling network consists of two separate estimators (also called streams): one for the state value function and one for the state-dependent action advantage function. While sharing a common convolutional feature learning module. The main benefit of this factoring is to generalize learning across actions without imposing any change to the underlying reinforcement learning algorithm [4].

The two streams previously mentioned, are combined via a special aggregating layer to produce an estimate of the state-action value function Q as shown in Figure 5 (green output module) [4]. Where, the aggregating layer consists on an implementation of the equation (1), which basically combines both network's output Q-values for each action. The state value function is represented by $V(s;\theta,\beta)$ and $A(s,a;\theta,\alpha)-\frac{1}{\eta}\sum_{a'}A(s,a';\theta,\alpha)$ represents the action advantage function. Here, θ denotes the parameters of the convolutional layers, while α and β are the parameters of the two streams of fully-connected layers, η is the replication factor, and (s,a) are the state and actions of the net [4].

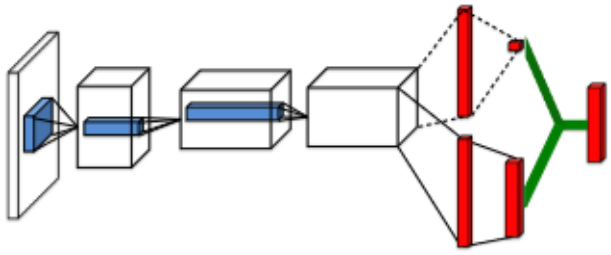

Figure 5: Dueling Q-network architecture [4].

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + [A(s,a;\theta,\alpha) - \frac{1}{\eta}\sum_{a'}A(s,a';\theta,\alpha)] \quad (1)$$

Finally, in our baseline model we worked with gray-scale images (we configured the Doom game to provide frames in gray-scale for simplicity in the processing). Thus the D-DQN network presents the following structure:

- **Layer A:** a convolutional layer with a *3×3* filter size, stride of 2, and batch normalization of 8 resulting in an *3×3×8* output, plus RELU as an activation function.
- **Layer B:** a convolutional layer with a *3×3* filter size, stride of 2, and batch normalization of 8 resulting

in an *3×3×8* output, plus RELU as an activation function.

- **Layer C:** a convolutional layer with a *3×3* filter size, stride of 1, and batch normalization of 8 resulting in an *3×3×8* output, plus RELU as an activation function.
- **Layer D:** a convolutional layer with a *3×3* filter size, stride of 1, and batch normalization of 16 resulting in an *3×3×16* output, plus RELU as an activation function.
- **Layer E:** a fully-connected layer for the state value function.
- **Layer F:** a fully-connected layer for the state-dependent action advantage function
- **Topology:** $\rightarrow A - B - C - D - |^{\rightarrow F}_{\rightarrow E}$

4.2. Inception Module Enhancement

We decided to "go deeper with convolutions" to improve the D-DQN network architecture used in our baseline model. Thus, we introduced a new model based upon the "inception" architecture approach described in [5].

The new architecture follows the fundamental structure of D-DQN, which is described in section 4.1. But the convolutional layers were modified, and two new more with filter sizes of *1×1* were added to follow the "inception" approach. Where the *1×1* convolutions, are used to compute reductions before the expensive 3×3 convolutions. Besides, being used as reductions, they also include the use of rectified linear activation which makes them dual-purpose [5]. Furthermore, some dropout layers were implemented in order to avoid overfitting problems due to the depth of the network.

Finally, as we did with the baseline model. In our inception module enhancement we worked with gray-scale images. As a result, our network presents the following structure:

- **Layer A:** a convolutional layer with a *1×1* filter size, stride of 2, and batch normalization of 8 resulting in an *1×1×8* output, plus RELU as an activation function.
- **Layer B:** a convolutional layer with a *3×3* filter size, stride of 2, and batch normalization of 8 resulting in an *3×3×8* output, plus RELU as an activation function.
- **Layer C:** a dropout layer with 0.5 probability of an element to be zero-ed.
- **Layer D:** a convolutional layer with a *3×3* filter size, stride of 1, and batch normalization of 8 resulting in an *3×3×8* output, plus RELU as an activation function.
- **Layer E:** a dropout layer with 0.5 probability of an element to be zero-ed.
- **Layer F:** a convolutional layer with a *1×1* filter size, stride of 1, and batch normalization of 16 resulting in an *1×1×16* output, plus RELU as an activation function.
- **Layer G:** a convolutional layer with a *3×3* filter size, stride of 1, and batch normalization of 16 resulting in an *3×3×16* output, plus RELU as an activation function.
- **Layer H:** a dropout layer with 0.5 probability of an element to be zero-ed.
- **Layer I:** a convolutional layer with a *3×3* filter size, stride of 1, and batch normalization of 16 resulting in an *3×3×16* output, plus RELU as an activation function.
- **Layer J:** a dropout layer with 0.5 probability of an element to be zero-ed.
- **Layer K:** a fully-connected layer for the state value function.
- **Layer L:** a fully-connected layer for the state-dependent action advantage function
- **Topology:**
  $\rightarrow A - B - C - D - E - F - G - H - I - J - |^{\rightarrow K}_{\rightarrow L}$

5. Results

In this section the results of the three experiments carried out for this study are presented. A comparison between the experiments will be addressed in the section 6.

All the results were obtained by using a GPU, $epochs = 10$, $learning - rate = 0.00025$, $learning - steps - per - epoch = 2000$, and $discount - factor = 0.99$. Furthermore, the values obtained consists on testing the trained agent in the **defend_the_center.cfg** scenario offered by the ViZDoom Framework [6].

The purpose of the scenario **defend_the_center.cfg** is to teach the agent that killing the monsters is GOOD and when monsters kill you is BAD. The agent is rewarded only for killing monsters (+1 for killing a monster) so he has to figure out the rest for himself. The map is a large circle (see figure 6 for getting in context). The player is spawned in the exact center. 5 melee-only, the monsters are spawned along the wall, and can be killed after a single shot. After dying each monster is re-spawned after some time. Episode ends when the player dies (death penalty = 1) [6].

Figure 6: Example of the scenario: defend_the_center.cfg.

5.1. Experiment 1

This experiment consisted on testing the baseline model under the conditions described in section 5. The results are presented in the table 1.

Episode	Total score
1	1.0
2	3.0
3	0.0
4	2.0
5	1.0
6	-1.0
7	0.0
8	3.0
9	3.0
10	1.0
AVG	1.3

TABLE 1: Results of the experiment 1.

5.2. Experiment 2

This experiment consisted on testing the baseline model along with the preprocessing described in section 3.1, and under the conditions described in section 5. The results are presented in the table 2.

Episode	Total score
1	-1.0
2	2.0
3	1.0
4	1.0
5	0.0
6	1.0
7	0.0
8	0.0
9	0.0
10	2.0
AVG	0.6

TABLE 2: Results of the experiment 2.

5.3. Experiment 3

This experiment consisted on testing our new architecture designed for this work (see section 4) along with the preprocessing described in section 3.1, and under the conditions described in section 5. The results are presented in the table 3.

Episode	Total score
1	2.0
2	3.0
3	5.0
4	2.0
5	1.0
6	2.0
7	4.0
8	2.0
9	3.0
10	2.0
AVG	2.6

TABLE 3: Results of the experiment 3.

6. Analysis

Certainly the best results were obtained in the experiment 3, where we achieved a total score of 5.0 in the episode 3, a minimum of 1.0, and an average of 2.6. In contrast, the worst results were collected in the experiment 2, where we improved the preprocessing of the frames used for training the agent, and obtained a maximum total score of 2.0, a minimum of -1.0, and an average score of 0.6.

From our perspective, the results that we acquired for the experiment 2 are contradictory since we expected to improve the behavior of the agent when using the baseline model with a better image preproccesing. However, these results help to came up with an interesting point due to we demonstrated that the baseline architecture was not capable of properly deal with the new form of the data used to train the agent. Thus, to take advantage of frame buffers that have segmentation format, this network must be improved with more convolutional layers as we did for our model.

7. Conclusions and Future Work

In this study, we introduced a deep reinforcement learning model for playing Doom, which was designed with a Duel DQN baseline Architecture and then improved with an Inception module. The results analysis suggest that this last addition improved the bot score in most of the testing episodes, so it seems that going deeper in convolution layers and using 1x1 kernel filters is an appropriate approach for the Doom reinforcement learning task. We also came to the conclusion that in order to use the preprocessing described in section 3.1 with a model that is being trained to learn to play Doom, it must have a robust architecture, which at least implements a deeper network than the baseline used in this study. Finally we noticed that the incremental experimentation was useful to distinguish the effects of each change to the training workflow.

In terms of future work, we propose to use more scenarios to make the bot more robust and improve its performance in unknown maps or even other first person shooters with similar objectives and button configurations. Furthermore, we think that adding an Attention layer to the architecture would be a great attempt of improving the decision making of our model since the locality issue of convolution is decreased.

References

[1] Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). *"Reinforcement Learning: A Survey"*. Journal of Artificial Intelligence Research. 4: 237–285. arXiv:cs/9605103. doi:10.1613/jair.301. S2CID 1708582.

[2] De Luca, G. *What is a Policy in Reinforcement Learning?* 2020. Extracted from: https://www.baeldung.com/cs/ml-policy-reinforcement-learning

[3] ”Doom, the original and best first-person shooter, is 20 years old today - ExtremeTech”. Archived from the original on October 22, 2017.

[4] Ziyu, Wang; Tom, Schaul; Matteo, Hessel; Hado, van Hasselt; Marc, Lanctot; Nando, de Freitas. (2016).*“Dueling Network Architectures for Deep Reinforcement Learning”*. http://arxiv.org/abs/1511.06581

[5] Szegedy, Christian; Liu, Wei; Jia, Yangqing; Sermanet, Pierre; Reed, Scott E.; Anguelov, Dragomir; Erhan, Dumitru; Vanhoucke, Vincent; Rabinovich, Andrew. (2014).*“Going Deeper with Convolutions”*. http://arxiv.org/abs/1409.4842

[6] Wydmuch, Marek; Kempka, Michał; Jaśkowski, Wojciech. (2018). *ViZDoom Competitions: Playing Doom from Pixels*. IEEE Transactions on Games. http://arxiv.org/abs/1605.02097

[7] Image Processing with Python. (2021). *Thresholding*. https://datacarpentry.org/image-processing/07-thresholding/