Towards Playing a 3D First-Person Shooter Game Using a Classification Deep Neural Network Architecture

Paulo Bruno S. Serafim, Yuri Lenon B. Nogueira, Creto A. Vidal, Joaquim B. Cavalcante Neto
Department of Computing
Federal University of Ceará
Fortaleza, Brazil
paulobruno@alu.ufc.br; {yuri, cvidal, joaquimb}@dc.ufc.br

***Abstract*—In this work, we present a network architecture to solve a supervised learning problem, the classification of a handwritten dataset, and a reinforcement learning problem, a complex First-Person Shooter 3D game environment. We used a Deep Neural Network model to solve both problems. For classification, we used a Softmax regression and cross entropy loss to train the network. To play the game, we used a Q-Learning adaptation for Deep Learning to train the autonomous agent. In both cases, the input was only the pixels of an image. We show that this single network architecture is suitable for the classification task and is capable of playing the 3D game. This result gives us an insight into the possibility of a general network architecture, capable of solving any kind of problems, regardless of the learning paradigm.**

***Keywords*-first-person 3D environment, visual-based autonomous agent, supervised learning, reinforcement learning, deep neural networks**

I. Introduction

Digital games are often used as environments for testing different algorithms and techniques. Some characteristics that contribute to the popularity of digital games are: complexity, nondeterminism, well defined limited input - the pixels of the screen - and great public exposure. All that increases the interest and importance of research on digital games.

Several research fields such as Computer Graphics, Computer Animation, Virtual Reality and Computational Intelligence use digital games as their testbeds. Computational Intelligence, for example, is concerned with conceiving agents that are capable of playing the game. In this paper, we construct an autonomous virtual agent that is immersed in a 3D virtual environment and plays a First-Person Shooter game. The type of technique developed here can be applied in situations within immersive or semi-immersive environments in which interactions with other characters are needed, for example, in training simulations, where the autonomous agent can be a rival for the human player. In order for that agent to play the game satisfactorily, we need a proper learning strategy.

To solve a problem where an agent has to figure out what to do in a given situation, receiving as feedback a reward for its actions, we use Reinforcement Learning [21], in which the learning process occurs in order to maximize a numerical reward by choosing certain actions for every possible state on the environment. Hence, since a reward is granted to each action, the agent must find out what is the best action to take in a given state.

Another desired characteristic of digital game playing is the input fairness, which means that the input is the same no matter the player is a human or a controller. Using the pixels of the screen as input guarantees input fairness. However, the construction of an autonomous virtual player that receives as input only raw pixels is a very difficult task, for which several solutions have been proposed, including Deep Learning [11]. That solution is a suitable solution, which is now a very well known field of research that is used across many platforms and is recently being used in modeling autonomous game agents. Since its development [5], Deep Learning has been used mainly in classification tasks, a subset of supervised learning problems, where given a certain entry, the model classifies it in accordance with an expected label.

In this work, we present how a Deep Neural Network architecture, which was originally devised for a supervised learning problem, can play a digital First-Person Shooter game (Fig. 1), therefore a reinforcement learning problem. With this approach, we show that a model based on a single network architecture, typically used in a handwritten digit classification task representing the supervised learning problem, can be used as a vision model for an autonomous agent of a First-Person Shooter game. Thus, we show that it is possible to solve different learning problems through a general network architecture.

In section II, we summarize some works which use a Deep Neural Network model to play digital games. Section III brings a description of Q-Learning, the main algorithm used in Reinforcement Learning problems. The description of the performed experiments is shown in section IV. In section V, we show obtained results and a discussion about them. Lastly, some possible future works are suggested in section VI.

II. Related Works

The use of a Deep Learning model in Reinforcement Learning was first proposed by Mnih et al. [14]. The authors trained a Deep Neural Network using a variant of the Q-Learning algorithm. This approach has been used to successfully produce agents capable of playing digital games receiving as input only raw pixels from the screen.

In that pioneer work, a Deep Q-Learning model learned to play seven Atari 2600 games using as input only

978-1-5386-3588-9/17 $31.00 © 2017 IEEE
DOI 10.1109/SVR.2017.24

Figure 1. ViZDoom, a Doom-based First-Person Shooter 3D environment.

raw pixel data, i.e., a brute high-dimensional input. The authors also used an adaptation of Q-Learning called Experience Replay, which improved the efficiency of data use, increased the learning speed and led to a better choice of parameters to avoid local minima. In 2015, Mnih et al. [15] increased the size of their previous work, by applying the technique to 49 games, and reaching human-level performance in 29 of them.

Inspired by these works, games from Atari 2600 became vastly used as testbeds, and new techniques for Deep Learning based models arose. Hasselt et al. [3] created a variant of Q-Learning called Double Q-Learning, in which two simultaneous value functions were learned, resulting in two different sets of weights. Besides, it improved the performance over the traditional algorithm. They also tested it in 57 Atari 2600 games, eight more than in Mnih et al. [15].

Another related advance was made by Nair et al. [16], which proposed a modification of the technique presented in the work of Mnih et al. [14] [15], by introducing a distributed system architecture. The parallelization improved the performance of an agent, in comparison with previous approaches. The model was tested in the same 49 Atari 2600 games and in 41 of them the technique outperformed the non-distributed models.

Wang et al. [24] used a new approach, dividing a network to represent two separate estimators: one to be used for the state-value function; and the other to be used for the state-dependent action advantage function. In practice, the first estimator was concerned with the result yet to come, while the second was concerned with immediate actions.

Parisotto et al. [19] proposed a new method of training a single Deep Network of actions over a set of related tasks. The method consists in learning on decision making of an autonomous agent through the orientation of expert teachers. They called this model *Actor Mimic*, and showed that this technique played several Atari 2600 games in the same level as an expert, all simultaneously.

A new model using asynchronous gradient descent was proposed by Mnih et al. [13] for controller optimization. The authors presented variations of four Reinforcement Learning techniques and showed that their method outperformed the state-of-the-art techniques. Moreover, the computation effort was greatly reduced, because the model was trained in a single multi-core processor in half the time of conventional training.

Other games were also used as testbeds in Deep Learning models. Hausknecht and Stone [4] created a new model to play Half-Field offense, one of the domains of the RoboCup 2D Soccer Simulation League. An offensive agent was trained using the model, in order to score as many goals as possible. The results showed that the agent outscored the 2012 RoboCup 2D Soccer in percentage of goals made.

First-Person Shooter games were also used as testbeds. Lample and Chaplot [8] used an architecture of a Deep Recurrent Network connected to a Long-Short Term Memory neural network. During training time, they explored information of game features, such as presence or absence of an enemy and items, to simultaneously learning them while minimizing the objective function. This aspect greatly improved the training speed and the performance of the agent.

III. Background

In this section, we present a short summary of the Deep Q-Network model used to train an autonomous agent in a Reinforcement Learning problem [8] [14].

A. Deep Q-Network

Reinforcement Learning tasks are sequential decision problems in which the objective is to find the best policy in order to maximize the sum of received rewards. A policy is a set of actions that should be taken by the agent for every possible state. The agent analyzes the current state s, and decides what action, a, to take according with the policy, π. The goal of the agent is then to find the best policy in which the expected sum of discounted rewards, R_t, is maximum

$$R_t = \sum_{i=t}^{T} \gamma^{\,i-t} r_i \tag{1}$$

where T is the last iteration, t is the current iteration and γ is the discount factor, which varies in the interval [0,1]. The discount factor trades-off the importance of future rewards, so that immediate rewards are more important than later ones.

The value of an action a in a state s under a given policy π, the Q-function, is defined as the expected return

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a] \tag{2}$$

The optimal value of $Q^{\pi}(s,a)$, Q^{*}, is defined as the highest expected return by following any strategy

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{3}$$

An optimal policy is then derived from the optimal values by choosing the actions with highest value in each state. Instead of trying to learn all action values, it is

easier to learn a parametrized value function Q_θ, in which Q^π is close to the optimal Q-function Q^*. Therefore, the algorithm tries to find a θ such that $Q_\theta(s,a) \approx Q^*(s,a)$.

The optimal Q-function verifies the Bellman optimality equation

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \quad (4)$$

where s' is the reached state and a' represents the following set of actions. Therefore, if $Q_\theta(s,a) \approx Q^*(s,a)$, it is natural to think that Q_θ should be close to verify the Bellman optimality equation, which will lead to the loss function

$$L_i(\theta_i) = \mathbb{E}[(y_t - Q_{\theta_t}(s,a))^2|s,a,r,s'] \quad (5)$$

where s is the current state, a is the action taken, r is the reward received, s' is the state reached and

$$y_t = r + \gamma \max_{a'} Q^*(s',a') \quad (6)$$

The Q-learning updates, using the loss function estimations of (5), are stable and perform well in practice [15].

IV. Experiment

A. MNIST

For the classification task, we use the MNIST database [10]. MNIST is a set of handwritten digits used as a testbed for classification models. Some examples of MNIST entries are presented in Fig. 2. It is well established as an indicative of the quality of a classification algorithm. In this work, we use the MNIST dataset to evaluate the proposed neural network architecture as a classification model, which will indicate that the architecture is well suited for supervised learning problems.

1) Hyperparameters: The MNIST database [10] contains 70000 images of handwritten digits. In our case, 55000 images are used for training and 10000 are used for testing. All the hyperparameters were chosen arbitrarily, as follows. We used a learning rate of 0.0001. In order to prevent overfitting, we use Dropout [20], a technique that randomly removes (or "drop") a certain number of neurons, so that the network will not adapt to the inputs. Here, we use a dropout probability of 0.5.

2) Training Regime: The training occured using mini-batches of size 50. The Softmax regression function, also known as multinomial logistic regression, was the model used for digit recognition. This function is used in multi-class classification problems, and gives the probability of the input to belong to a certain class, with all probabilities summing to 1 [7]. To accelerate the learning process, cross-entropy [18] was used as the loss function. The expected result is that the classifier identifies the most number of digits correctly.

Figure 2. Examples of digits from MNIST dataset.

B. ViZDoom

In the Reinforcement Learning problem, we build an autonomous game character to play a 3D First-Person Shooter game. We used the API developed by Kempka et al. [6], ViZDoom, a research platform based on the game Doom (Fig. 1). This tool gives access to the game engine and enables the construction of a custom agent. We used a Deep Neural Network model and Q-Learning [25] as the learning algorithm used to train the autonomous agent.

1) Hyperparameters: All the following parameters were chosen arbitrarily. For Q-Learning we used a discount factor, γ, of 0.99. The learning rate was equal to 0.00025. In order to reduce correlation between consecutive frames, we used Experience Replay [12]. A replay memory keep track of the latest ten thousand frames and at every update a randomly chosen sample is passed to the network.

The network was trained using the RMSProp algorithm, with mini-batches of size 64. The RMSProp is an adaptation of Mini-Batch Gradient Descent which divides the gradient by a running average of its recent magnitude [22]. Lastly, the Dropout technique [20] was also used, with probability of 0.8.

All network weights were initialized with random values and all bias were initialized with a value of 0.1. To balance the trade-off between exploration and exploitation, we used an ϵ-greedy policy [25], with linear decay from 1.0 to 0.1 along all the lifespans. This means that the agent has an ϵ probability of taking a random choice rather than the current best action.

2) Scenarios: Two scenarios were used in this work, which are:

Basic. In this scenario (Fig. 3), the agent plays against one circular enemy controlled with a built-in behaviour. The enemy is randomly spawned in the opposing wall. The player can take one of three actions: move left, move right or shoot. One shot is enough to kill the monster. An episode starts with the player in one side of the map and one enemy in the opposing side. Episode finishes when the monster is killed or on timeout.

Defend the Line. As presented in Fig. 1, this environment has two different types of enemy. The first enemy is a movable red worm which will move towards the player and then hurt him, and the second is a static brown snake-like monster which can shoot a fireball in the direction of the player. The monster are spawned in the opposing wall of the map.

An episode starts with six enemies, three of every type, in the opposing side. Initially, the enemies are killed with a single shot. For every enemy killed, the player receives

Figure 3. ViZDoom's Basic environment. A single enemy is spawned on the other side of the room. The player should kill it as soon as possible.

one point. After dying, every monster will respawn after some time and they will need more shoots to be killed.

Every time the agent is hit by the movable enemy or shot by the static one, it will lose health. Episode finishes when the player dies. This is inevitable because the player has limited ammunition. When this happens, he will receive a reward equal to the number of enemies killed. The goal of the agent is to maximize his reward and therefore the number of enemies killed.

3) Evaluation Metrics: Every scenario has a different evaluation:

Basic. The player is rewarded a score of 101 when he kills the monster. He loses 5 points for missing a shot and loses 1 point every step he remains alive. In other words, the agent must kill the enemy, without missing a single shot and as quickly as possible. The timeout occurs after 300 steps have passed.

Defend the Line. Just one reward is given to the player, he receives one point after killing an enemy. However, since he loses after being killed, he must find a way to kill as many enemies as possible before dying. The player is hurt after being shot or touched, but these are not given to him as a numerical feedback, only in the screen.

4) Training regime: Training occurred within five thousand learning steps for each epoch. Every learning step means a run of Q-Learning and an update of network weights. Therefore, the training was the moment in which the agent actually learned. In Basic environment, the agent trained for 40 lifespans, but in Defend the Line environment, he trained for 90 lifespans.

C. Network Architecture

The network architecture is summarized in Fig. 4. The input is a matrix of floating point numbers, representing a grayscale image. It is given to a convolutional layer with a patch size of 5 by 5, a stride of 1 and zero padded, this way the size of the output of this layer is the same as the input. This first convolutional layer computes 32 features. Developed by LeCun et al. [9], a convolutional layer has the aspect of detecting a specific feature according to the spatial position of the input. In other words, it automatically filters some characteristics of the input. The result of this layer is then passed through a ReLU activation function [17] [1].

After that, a max pool is applied in every feature. With a shape size of 2 by 2, this max pooling layer halves the size of the given input. This way, the input of the layer is down-sampled, thus reducing the complexity of the network. However, this reduction will lead to loss of information, such as spatial position and orientation. Although these characteristics are not necessarily fundamental in classification tasks, they are very important for a game agent to handle a certain view.

Then, another convolutional layer with patch size of 5 by 5, a stride of size 1 and zero padded is employed to compute 64 features, followed by a ReLU activation function. Next, another max pooling layer is used, also with size of 2 by 2, halving the given input. The outputs of

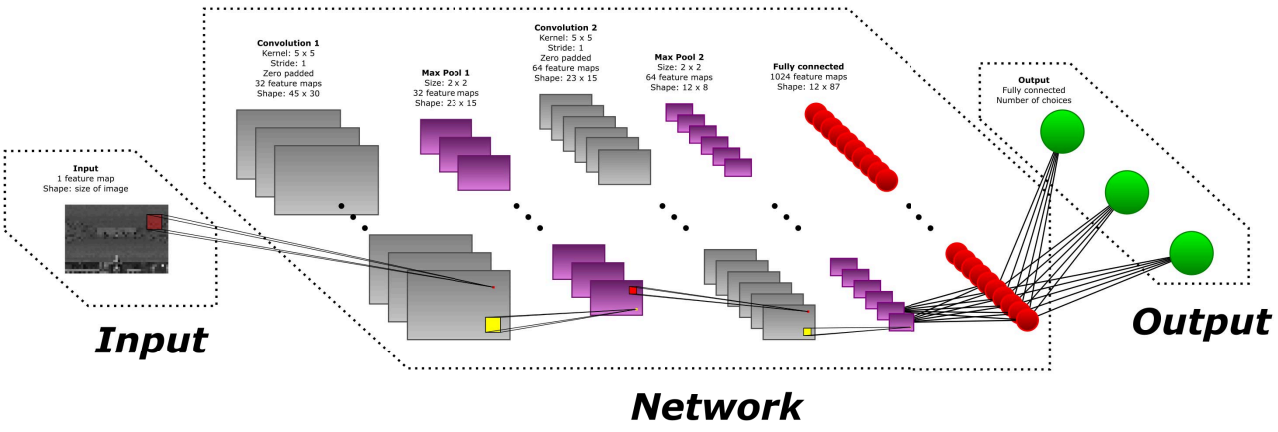

Figure 4. An illustration of the Network Architecture used. **Input**: a matrix of floating point numbers, represented by a grayscale image. **Network**: the input image is given to a convolutional layer to compute 32 features, followed by a max pooling layer which halves the dimension of the image; the resulting output is passed to a second convolutional layer to compute 64 features, followed by another max pooling layer to halve the received input; the output is then fed into a fully connected layer, which is fully connected to the output layer. **Output**: the result of the network according to the task.

the past layer are then fed into a fully connected layer with 1024 neurons. This layer has the objective to put together all the previous features found. A dropout is then applied before the readout layer, in order to reduce overfitting. Lastly, the output layer is fully connected to the previous layer.

1) Input and Output: Both input and output are different for every case:

MNIST. The input is a 28 by 28 pixels gray image (Fig. 5). Specifically, a 28 by 28 matrix of values in the interval [0,1]. The output layer has 10 neurons, one for every digit class from zero to nine, to represent the percentage of the image belonging to that class. The neuron with the highest value indicates the desired answer.

ViZDoom application. The raw color image is preprocessed into a smaller one with reduced size and color so that the input of the network is a 45 by 30 pixels gray scale image (Fig. 6). Therefore, the input layer has 45 by 30 neurons with values in the interval [0,1]. The output of the network consists of 3 neurons, one for each possible action on the environment: move or turn left, move or turn right, and shoot. Every output neuron value means the q-value for each action.

V. Results and Discussion

A. MNIST

The executions of tests were made using 200 mini-batches of size 50 from the 10000 testing images. Such as in the literature [23], we obtained a percentage of success higher than 99%. This result is sufficient to show that this model is adequate for purposes of classification.

B. ViZDoom

For tests in the ViZDoom Environment, the learning was disabled and the player chose only the best action already learned in from the network. Ten executions of tests were made in each scenario for each epoch. To see examples of the results described below, please refer to the accompanying video.

1) Basic: The optimal expected behavior is the agent to move towards the direction of the enemy and then shoot. If the enemy is spawned close to the player, this optimal behaviour will give him a score of about 95. If the enemy is spawned far from the player, the optimal behaviour will give him a score of about 50 to 75. The player is hugely penalized for a missed shot and will receive a score less than 50.

In the left side of Fig. 7, we can see that in training the score of the player grows over time. This shows that the algorithm is in fact learning to play the environment. However, in testing, the score starts as high as it ends. What explains this high score is that in the first epoch of training, the agent plays 315 episodes. Being a very simple scenario, this number is enough for the algorithm to find out the best actions. While in testing the chosen action is the one with the highest value, in training, the ϵ-greedy strategy is still choosing random actions with high probability. So, in training, the results are not great, except

Figure 5. Example of a MNIST input. An image is passed to the network as a matrix of 28 by 28 floating point numbers, with a value varying from 0 to 1.

a few lifespans ahead, when the ϵ decay increases the chance of choosing the best action.

This result shows that this single network architecture, designed for classification tasks, is indeed capable of playing a 3D First-Person Shooter game. It is a strong result, but the environment is very simple, and therefore we want to test the network in other scenarios.

2) Defend the Line: This environment is a lot harder than the previous one. Here, we do not have a clear expected behavior. However, we want to see the player turning towards an enemy and killing him as soon as possible. Considering that the only feedback the agent has is one point received when he kills an enemy, he should discover all the motion and shooting behavior by himself.

From the right side of Fig. 7, it becomes clear that the learning is occurring, denoted by the growth of the training curve. Also, the behavior of the agent is improving over time. Therefore, we can see that training phase is improving the choice of actions of the agent.

After 90 lifespans of training, the mean score stabilized. In that moment, the agent was capable of killing about seventeen enemies. This number shows that the agent learned how to play the game in that scenario. Moreover, not only he learned how to kill enemies, but also how to survive long enough to kill several of them. In fact, when we see him playing, his behavior matches exactly what we expected. He turns towards an enemy an shoots him very quickly.

Notice that the network architecture, intended for solving a supervised learning problem, managed to discover a complex behavior in a reinforcement learning problem. All that was achieved without knowing any intermediate steps to take.

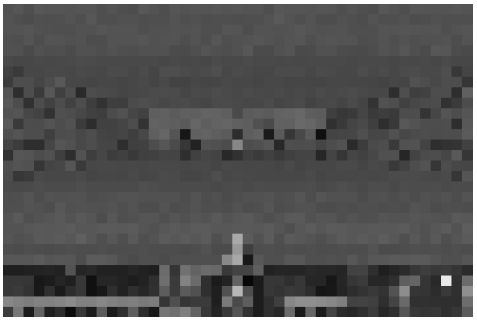

Figure 6. Input to both ViZDoom's environment. A 45 by 30 pixels grayscale image, where every pixel represents a floating number in the interval [0,1].

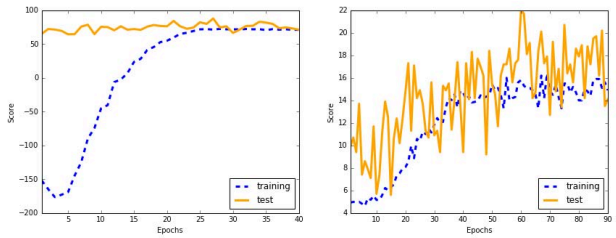

Figure 7. Training and testing results for each ViZDoom scenario played. The dashed line shows the evolution in training and the continuous line shows the evolution in testing. **Left**: Basic scenario: 40 lifespans were executed with a mean score of 75 points. **Right**: Defend the Line scenario: after 90 lifespans of execution, the agent had a mean score of about 17 points.

VI. Future Works

A. Increasing neural network size

The size of the network, showed in section IV-B, is relatively small. Increasing its size can lead to better results, because more features could be learned and passed throughout the network. This can be done by increasing the number of neurons in every layer, adding more layers or receiving more information as input. This last option is particularly useful when the network receives a preprocessed input, as in the case of ViZDoom in this paper, because this could make it easier for the model to identify features of the game.

B. Changing learning algorithms

As discussed in section IV-A, the learning in the MNIST classification problem occurs through a multinomial logistic regression and uses the cross-entropy function. On the other hand, the learning in the MNIST problem occurs using an adapted version of vanilla Q-Learning. Testing other algorithms can give better results. For MNIST, the work from Wan et al. [23], which achieved the best result so far in the task, seems to be the starting point. In ViZDoom side, some variations of Q-Learning could lead to better results, such as the Double Q-Learning algorithm [2], [3]. These adaptations will involve only the learning algorithms, the network will remain the same.

C. Different architecture settings

Notice that the network architecture presented in section IV-C has a very simple setting. This fact makes the results even more remarkable. A natural next step is to change the architecture, increasing its complexity. Generally, complex behaviors reflect internal dynamics of a complex network.

Some changes can be made to the presented architecture: the use of colored images, which could improve the agent's distinction of game features; the removal of max pooling layers, to decrease the loss of information, as seen in section IV-C; the elimination of dropout before the readout layer; and insertion of intermediate layers specialized in distinguishing characteristics of the game.

All of those changes will impact both supervised and reinforcement learning problems. Making changes to improve the performance of the game agent will increase the complexity of the network for the classification task too, and not necessarily receiving some of its benefits. This trade-off makes this work very hard to balance.

D. Different tasks

A further level of test with a much higher difficulty is to use the same network on other problems, such as: other games, with different scenarios, visual aspects or gender; other virtual environments; other learning tasks like unsupervised learning; or different reinforcement learning problems. All of those problems have their own idiosyncrasies and must give different levels of solutions. If a single architecture can lead to good results in some of those tasks, it could mean that a general network solution is closer.

However, to achieve good results in such different problems is challenging. As discussed in section V, the presented network performed well in both tasks, but did not achieved the best results presented in the literature. Hence, two observations must be taken in consideration: first, the difficulty will probably be much higher and then good results will have a higher cost; and second, an extraordinary result for all types of problems does not need to be the best result for every one of them.

VII. Conclusion

We showed that a single network architecture was capable of solving a supervised learning problem and a complex reinforcement learning problem. The supervised learning task was represented by the classification of handwritten digits, giving a small error percentage. Two environments of a 3D First-Person Shooter were given as the reinforcement learning tasks: a basic scenario, which was quickly solved with the expected behavior, and a more complex scenario, where the agent learned how to kill several enemies without knowing the intermediate steps to achieve this result. This gives us a perspective on the possibility of solving problems from different environments or learning paradigms with a single network architecture, which would lead us to a potentially generalized solution and an approach to handle complex tasks.

Acknowledgment

The authors would like to thank CNPq, the National Council for Scientific and Technological Development, and the Graduate program (MS and PhD) in Computer Science at the Federal University of Ceará (MDCC/UFC) for their financial support.

References

[1] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323.

[2] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2613–2621. [Online]. Available: http://papers.nips.cc/paper/3964-double-q-learning.pdf

[3] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 2094–2100. [Online]. Available: http://dl.acm.org/citation.cfm?id=3016100.3016191

[4] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *Proceedings of the International Conference on Learning Representations (ICLR)*, May 2016.

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006. [Online]. Available: http://dx.doi.org/10.1162/neco.2006.18.7.1527

[6] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "Vizdoom: A doom-based AI research platform for visual reinforcement learning," *CoRR*, vol. abs/1605.02097, 2016. [Online]. Available: http://arxiv.org/abs/1605.02097

[7] B. Krishnapuram, L. Carin, M. A. T. Figueiredo, and A. J. Hartemink, "Sparse multinomial logistic regression: fast algorithms and generalization bounds," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 6, pp. 957–968, June 2005.

[8] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," *CoRR*, vol. abs/1609.05521, 2016. [Online]. Available: http://arxiv.org/abs/1609.05521

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec 1989.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[11] Y. LeCun, Y. Bengio, G. Hinton, L. Y., B. Y., and H. G., "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1993, uMI Order No. GAX93-22750.

[13] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv*, vol. 48, pp. 1–28, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[16] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De 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:1507.04296*, p. 14, 2015. [Online]. Available: http://arxiv.org/abs/1507.04296

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 807–814. [Online]. Available: http://www.icml2010.org/papers/432.pdf

[18] G. E. Nasr, E. A. Badr, and C. Joun, "Cross entropy error function in neural networks: Forecasting gasoline demand," in *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*. AAAI Press, 2002, pp. 381–384. [Online]. Available: http://dl.acm.org/citation.cfm?id=646815.708603

[19] E. Parisotto, L. J. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," *CoRR*, vol. abs/1511.06342, 2015. [Online]. Available: http://arxiv.org/abs/1511.06342

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[21] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[22] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, 2012.

[23] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, S. Dasgupta and D. Mcallester, Eds., vol. 28, no. 3. JMLR Workshop and Conference Proceedings, May 2013, pp. 1058–1066. [Online]. Available: http://jmlr.org/proceedings/papers/v28/wan13.pdf

[24] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling Network Architectures for Deep Reinforcement Learning," *arXiv*, no. 9, pp. 1–16, 2016. [Online]. Available: http://arxiv.org/abs/1511.06581

[25] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/ chrisw/new$_thesis.pdf$